# BAB 2
# TINJAUAN PUSTAKA

## 2.1. *Game*

*Game* merupakan salah satu media hiburan yang populer untuk semua kalangan usia. Sejak pertama kali *game* ditemukan sampai saat sekarang, teknologi *game* telah mengalami kemajuan yang terbilang pesat. Hal ini ditandai dengan berkembangnya jenis *game*, produk, alat dan interaksi *game* dengan penggunaan yang semakin beragam bentuknya. [7]

### 2.1.1. Pengertian *Game*

Menurut Andang Ismail terdapat dua pengertian *game*. Pertama *game* adalah sebuah aktifitas bermain yang murni mencari kesenangan tanpa mencari menang dan kalah. Kedua, *game* diartikan sebagai aktifitas bermain yang dilakukan dalam rangka mencari kesenangan dan kepuasan, namun ditandai dengan menang dan kalah.

Berdasarkan representasinya, *game* dapat dibedakan menjadi 2 jenis yaitu *game* 2D (2 dimensi) dan 3D (3 dimensi). *Game* 2D adalah *game* yang secara matematis hanya melibatkan 2 elemen koordinat kartesius yaitu x dan y, sehingga konsep kamera pada *game* 2D hanya menentukan gambar pada *game* yang dapat dilihat oleh pemain. Sedangkan *game* 3D adalah game yang selain melibatkan elemen x dan y juga melibatkan elemen z pada perhitungan sehingga konsep kamera pada *game* 3D benar-benar menyerupai konsep kamera pada kehidupan nyata. [7]

### 2.1.2. Sejarah *Game*

Sejak pertama kali ditemukan sampai saat ini, *game* telah melewati beberapa fase evolusi penting dimana yang berperan besar terhadap perkembangan *game* sudah semakin pesat sekarang ini.

Game pertama kali dibuat pada tahun 1966 oleh Ralph Baer bersama dengan timnya yang berjumlah 500 orang terdiri dari para insinyur dan teknisi. Setelah sebulan bekerja keras, Bear berhasil menampilkan dua titik putih yang berkejar-kejaran dilayar. Pada tahun 1972 muncul nama baru dalam dunia *game*.

Nolan Bushel mendirikan perusahaan Atari dan membuat *game Arcade* Pong. Pada era baru dalam perkembangan dunia *game* terjadi pada tahun 1988 yang didomininasi oleh perusahaan jepang. Nitendo yang awalnya hanya memproduksi mesin *fotocopy*, dengan *video game* sistemnya telah mencapai omset miliaran dolar. Setelah masa awal perkembangan *game* ini, dunia *game* telah melalui beberapa fase yaitu fase *game* PC, fase *game console* dan fase *game online*. Sekarang ketiga fase game tersebut sudah semakin banyak variannya dan masing-masing memliki kelompok penggemar sendiri. [7]

### 2.1.3. Elemen Dasar *Game*

Menurut Teresa Dillon [8] elemen-elemen dasar sebuah *game* adalah :

1. *Game Rule*

   Game rule merupakan aturan perintah, cara menjalankan, fungsi objek dan karakter di dunia permainan. Dunia *game* bisa berupa pulau, dunia khayal, dan tempat-tempat lain yang sejenis yang dipakai sebagai setting tempat dalam permainan *game*.

2. Plot

   Plot biasanya berisi informasi tentang hal-hal yang akan dilakukan oleh player dalam *game* dan secara detail, perintah tentang hal yang harus dicapai dalam *game.*

3. Thema

   Di dalam biasanya ada pesan moral yang akan disampaikan.

4. *Character*

   Pemain sebagai karakter utama maupun karakter yang lain yang memiliki ciri dan sifat tertentu.

5. *Object*

   Merupakan sebuah hal yang penting dan biasanya digunakan pemain untuk memecahkan masalah, adakalanya pemain harus punya keahlian dan pengetahuan untuk bisa memainkannya.

6. *Text*, *Grafik* dan *Sound*

   Game biasanya merupakan kombinasi dari media teks, grafik maupun suara, walaupun tidak harus semuanya ada dalam permainan *game*.

7. Animasi

   Animasi ini selalu melekat pada dunia *game*, khususnya untuk gerakan karakter, karakter yang ada dalam *game*, properti dari objek.

*8. User Interface*

   Merupakan fitur-fitur yang mengkomunikasikan user dengan *game*.

### 2.1.4. Jenis-jenis *Game*

Berikut ini beberapa jenis *game* berdasarkan cara pembuatannya, cara pemasarannya dan mesin yang menjalankannya [9]. Jenis-jenis *game* tersebut adalah :

1. *Game* PC

   *Game* PC adalah *game* yang dimainkan pada PC (*Personal Computer*) yang memiliki kelebihan yaitu tampilan antarmuka yang baik untuk *input* maupun *output*. *Output* visual berkualitas tinggi karena layar komputer biasanya memiliki resolusi yang jauh lebih tinggi dibandingkan dengan layar televisi biasa. Kekurangannya adalah spesifikasi komputer yang sangat bervariasi antar satu komputer dengan komputer yang lainnya menyebabkan beberapa *game* dapat ditampilkan dengan baik pada satu komputer tetapi tidak berjalan dengan baik pada komputer yang lainnya.

2. *Game Console*

   *Game console* adalah *game* yang dijalankan pada suatu mesin spesifik yang biasanya tersedia di rumah seperti Xbox, Nintendo Wii dan lain-lain.

3. *Game Arcade*

   *Game arcade* adalah *game* yang dijalankan pada mesin dengan *input* dan *output* audio visual yang telah terintegrasi dan tersedia ditempat-tempat umum.

4. *Game Online*

   *Game online* adalah *game* yang hanya dapat dumainkan secara *online* melaui LAN atau internet.

### 2.1.5. *Genre Game*

Berdasarkan *genre, game* dapat dibagi menjadi beberapa *genre* yaitu [9] :

1. *Action Game*

*Action game* dikategorikan sebagai *game play* dengan model pertarungan. Berikut beberapa macam *game* yang termasuk dalam *genre action game* yaitu :

a. *Action Adventure Game*

*Genre game* yang berfokus pada eksplorasi dan biasanya mempunyai unsur *item gathering*, penyelesaiaan *puzzle simple* dan pertarungan. Contoh *game* dari *genre* ini adalah *The Legend Of Zelda series* dan *Metroid series*.

b. *Stealth Game*

Termasuk dalam *genre* terbaru, biasanya digolongkan dalam mata-mata yang bias melakukan aksinya secara rahasia. Contoh *game* dari *genre* ini adalah *Metal Gear series.*

c. *Survival Horror Game*

*Genre game* yang berusaha membuat pemain menjadi tegang dan takut dengan elemen-elemen horror. Contoh *game* dari *genre* ini adalah *Resident Evil series* dan *Alone in The Dark*.

d. *Beat'em Up Game*

*Genre game combat* dimana satu orang melawan banyak musuh yang telah disediakan. Contoh *game* dari *genre* ini adalah *Dynasty Warrior series* dan *Final Fight.*

e. *Fighting Game*

*Game* pertarungan dua pemain dengan jurus-jurus yang biasa dikeluarkan dengan menekan beberapa tombol pada *keyboard* dengan urutan tertentu. Contoh *game* dari *genre* ini adalah *Street Fighter* dan *Tekken series.*

f. *Maze Game*

*Genre game* yang membutuhkan kecepatan berpikir dan bereaksi serta berunsur ketepatan menavigasi. Contoh *game* dari *genre* ini adalah *Pac-Man.*

g. *Platfrom Game*

*Genre Game* dengan *game play* berlari, melompat, mengayun dan sebagainya. Contoh *game* dari *genre* ini adalah *Donkey Kong* dan *Ray Man.*

h. *Shooter*

a) *First Person Shooter Game*

*Genre game* yang mengutamakan *shooting* dan *combat* dari perspektif langsung mata karakter yan bertujuan untuk memberikan pemain perasaan berada ditempat itu dan bisa fokus menembak.

b) *Massively Multiplayer Online First Person Shooter Game*

*Genre game* yang mengkombinasikan *game play first person shooter* dengan dunia virtual dimana banyak *player* juga ikut bermain melaui internet. Contoh *game* dari *genre* ini adalah *Counter Strike Online*.

c) *Third Person Shooter Game*

*Genre game* yang sama seperti *first person shooter game* yaitu mengutamakan *shooting* dan *combat* dari perspektif karakter yang bertujuan untuk memberikan pemandangan yang lebih luas dan gerakan yang lebih banyak.

d) *Tactical Shooter Game*

*Genre* yang mengutamakan perencanaan dan kerja sama tim untuk memenangkan *game*. Contoh *game* dari *genre* ini adalah *Tom Clancy's Ghost Recon series.*

e) *Light Gun Game*

*Genre* dengan lebih banyak pada *arcade* dengan peralatan tertentu seperti senjata mainan yang mempunyai sensor khusus terhadap layar. Contoh *game* dari *genre* ini adalah *Time Crisis* dan *Duck Hunt.*

f) *Shoot'em Up Game*

*Genre* dengan ciri khas gambar 2D dan *scrolling playing area*. Contoh *game* dari *genre* ini adalah *Star Fox series*.

2. *Adventure Game*

*Adventure game* dikategorikan sebagai *game play* yang mengharuskan pemain memecahkan bermacam-macam teka-teki melaui interaksi dengan orang lingkungan dalam *game* tersebut.

a. *Text Adventure*

Pemain akan menggunakan *keyboard* untuk mengetikkan berupa perintah dan komputer akan menganalisa perintah tersebut lalu menjalankan karakter sesuai perintah tersebut.

b. *Graphical Adventure Game*

*Genre* yang merupakan perkembangan dari *text adventure*. Pemain dapat menggunakan *mouse* untuk menggerakkan karakter.

c. *Visual Novel Game*

*Genre* yang memberikan keleluasaan untuk memilih jalan ceritanya sendiri.

d. *Interactive Movie Game*

*Genre game* dengan rangkaian *live action* dari karakter yang dimainkan pemain. Contoh *game* dari *genre* ini adalah Space Ace.

e. *Dialog Game*

Pada *genre* ini, pemain akan mengalami kemajuan tergantung pada apa yang mereka katakan. Contoh *game* dari *genre* ini adalah *Law And Order : The Vangeful Heart.*

3. *Role Playing Game*

*Role playing game* adalah *game* yang memiliki *game play* dimana karakter milik *player* akan berpetualang dengan *skill combat* dalam cerita *game*.

a. *Action Role Playing Game*

*Genre game* yang memasukkan unsur *action game* dan *action adventure game*. Contoh *game* dari *genre* ini adalah Diablo 1 & 2.

b. *Massively Multyplayer Online Role Playing Game*

Konsep dari *genre* ini terkombinasi dengan *genre-genre* lainnya yang berupa fantasi. Contoh *game* dari *genre* ini adalah Rising *Force Online*.

c. *Tactical Role Playing Game*

Dalam *genre* ini, pemain akan diberikan giliran masing-masing untuk menentukan langkah-langkah yang akan dilakukan oleh karakter. Contoh *game* dari *genre* ini adalah *Final Fantasy Tactics.*

4. *Simulation Game*

*Genre* ini bertujuan untuk memberikan pengalaman simulasi kepada pemain.

a. *Construction and Management Simulation Game*

*Genre* ini merupakan bagian dari *economic simulation game*. Contoh *game* dari *genre* ini adalah *Sims City series*.

b. *Economic Simulation Game*

*Genre* ini berupa simulasi keadaan ekonomi dimana pemain mengontrol keadaan ekonomi dari *game* tersebut. Contoh dari *genre* ini adalah Monopoly Tycoon.

c. *God Game*

Dalam *genre* ini tidak ada tujuan akhir yang membuat pemain memenangakan *game*. Contoh *game* dari *genre* ini adalah *The Sims series.*

d. *Government Simulation Game*

*Genre game* yang memasukkan unsur kepolisian, pemerintahan atau politik sebuah negara.

5. *Strategy Game*

*Genre strategy game* berfokus pada *game play* dimana dibutuhkan pemikiran yang tepat agar dapat meraih kemenangan.

a. *Real Time* Strategi

Dalam *real time* strategi, *action* dilakukan dalam waktu yang bersamaan oleh masing-masing pihak dimana *action* dimainkan per*ronde* atau bergiliran. Contoh *game* dari *genre* ini adalah Warcraft series.

b. *Tactical Game*

Dalam *genre* ini pemain harus menggunakan bermacam-macam taktik dan strategi untuk mencapai kemenangan. Contoh *game* dari *genre* ini adalah Dark Omen.

c. *4X Game*

*Genre* ini berarti penjelajahan, menjajah dan memusnahkan. Contoh *game* dari *genre* ini adalah Galactic Civizations.

d. *Artillery Game*

*Genre game* ini biasanya mengikutkan *combat* dengan tank atau tentara militer. Contoh *game* dari *genre* ini adalah Tanarus.

6. *Vehicle Simulation Game*

*Genre* ini merupakan simulasi yang memberikan pemain sebuah pengalaman realistik dalam mengendarai kendaraan tertentu.

a. *Flight Game*

Dalam *genre* ini,pemain tidak hanya bersimulasi mengontrol pesawat terbang tetapi juga bisa *combat* di udara. Contoh *game* dari *genre* ini adalah Falcon 4.0.

b. *Racing Game*

*Genre* yang menempatkan pemain sebagai *driver* dengan kendaraan seperti mobil. Contoh *game* dari *genre* ini adalah Need For Speed series.

c. *Space Game*

*Genre* ini bersifat pertarungan di angkasa luar. Contoh *game* dari genre ini adalah Star Wars dan Homeworld.

d. *Train Game*

*Genre* ini mensimulasikan yang berhubungan dengan transportasi kereta. Contoh dari *genre* ini adalah Rail Simulator.

### 2.1.6. *Game* 3D

Pada saat kemampuan proses pada komputer semakin cepat, para developer juga senantiasa menciptakan game yang semakin canggih sehingga muncul *engine* dengan grafik 3D (3 dimensi). Maka dari itu, *game* 3D dengan hitungan *polygon* yang sangat besar dan pencahayaan yang sudah canggih, juga tekstur maping mulai diproduksi. *Game* 3D merepresentasikan objek dalam bentuk 3 dimensi sehingga objek akan terlihat lebih nyata seperti dalam kehidupan nyata.

*Game* bertipe 3 dimensi merupakan *game* dengan grafis yang baik dalam penggambaran secara realita, kebanyakan *game-game* ini memiliki perpindahan kamera (*angle*) hingga 360 derajat sehingga dapat melihat secara keseluruhan dunia *game* tersebut [10].

### 2.1.7. 3D Grafis

Grafik komputer 3 dimensi biasa disebut 3D atau adalah bentuk dari benda yang memiliki panjang, lebar, dan tinggi. Grafik 3 Dimensi merupakan teknik penggambaran yang berpatokan pada titik koordinat sumbu x (datar), sumbu y (tegak), dan sumbu z (miring). Representasi dari data geometrik 3 dimensi sebagai hasil dari pemrosesan dan pemberian efek cahaya terhadap grafika komputer 2D.

Tiga dimensi, biasanya digunakan dalam penanganan grafis. 3D secara umum merujuk pada kemampuan dari sebuah *video card*. Saat ini *video card* menggunakan variasi dari instruksi-instruksi yang ditanamkan dalam *video card* itu sendiri (bukan berasal dari *software*) untuk mencapai hasil grafik yang lebih realistis dalam memainkan *game* komputer.

Grafik 3D merupakan perkembangan dari grafik 2D. Didalam grafika komputer, 3D merupakan bentuk grafik yang menggunakan representasi data geometri tiga dimensi. Suatu objek rangka 3D apabila disinari dari arah tertentu akan membentuk bayangan pada permukaan gambar. Proses pembuatan grafik komputer 3D dapat dibagi ke dalam tiga fase, yaitu 3D modeling yang mendeskripsikan bentuk dari sebuah objek, *layout* dan *animation* yang mendeskripsikan gerakan dan tata letak sebuah objek, dan 3D rendering yang memproduksi *image* dari objek tersebut. Istilah atau pengertian grafik 3D adalah sebuah gambar, garis, lengkungan dan sebagainya yang memiliki titik-titik yang menghubungkan menjadi sebuah bentuk 3D di dalam dunia *game*, 3D secara umum merujuk pada kemampuan dari sebuah *video card.* Saat ini *video card* menggunakan variasi dari instruksi-instruksi yang ditanamkan dalam *video card* itu sendiri (bukan berasal dari *software*) untuk mencapai hasil grafik yang lebih realistis dalam memainkan *game* komputer [11].

### 2.1.8. Klasifikasi Permainan

Berikut ini adalah klasifikasi permainan berdasarkan pembatasan usia dari permainannya, menurut ESRB (*Entering software Ratting Board*) adalah sebagai berikut [5]:

1. *Early Childhood* (3+) memiliki isi yang mungkin cocok untuk usia 3 keatas. Tidak mengandung materi yang tidak pantas seperti adegan kekerasan.
2. *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.
3. *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.
4. *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.
5. *Mature* (17+) memiliki konten yang mungkin cocok untuk orang usia 17 dan lebih tua. Judul-judul dalam kategori ini mungkin berisi konten kekerasan, darah, konten seksual dan bahasa yang kuat.
6. *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.

## 2.2. *Artificial Intelligence* (Kecerdasan Buatan)

Kecerdasan buatan atau *Artficial Intelligent* (AI) merupakan cabang dari ilmu komputer yang berhubungan dengan pengautomatisan tingkah laku cerdas. Kecerdasan buatan didasarkan pada teori suara (*sound theoretical)* dan prinsip-prinsip aplikasi dari bidangnya. Prinsip-prinsip ini meliputi struktur data yang digunakan dalam representasi pengetahuan, algoritma yang diperlukan untuk mengaplikasikan pengetahuan tersebut serta bahasa dan teknik pemrograman yang digunakan dalam mengimplementasikannya [12].

Berdasarkan sudut pandang, Ai dapat dipandang sebagai berikut :

1. Sudut pandang kecerdasan, AI adalah bagaimana membuat mesin cerdas dan dapat melakukan hal-hal yang sebelumnya hanya dilakukan oleh manusia.
2. Sudut pandang bisnis, AI adalah sekelompok alat bantu yang berdayaguna dan metodologi yang menggunakan alat-alat bantu tersebut untuk menyelesaikan masalah-masalah bisnis.
3. Sudut pandang pemrograman, AI meliputi studi tentang pemrograman simbolik, pemecahan masalah dan proses pencarian.
4. Sudut pandang penelitian :
    a. Riset tentang AI dimulai pada awal tahun 1960-an, percobaan pertama adalah membuat program permainan catur, membuktikan teori dan *general problem solving*
    b. AI adalah nama pada akar sari studi area.

### 2.2.1. Teknik Dasar Pencarian

Pencarian atau pelacakan merupakan salah satu teknik untuk menyelesaikan permasalahan AI. Keberhasilan suatu sistem salah satunya ditentukan oleh kesuksesan dalam pencarian dan pencocokan. Teknik dasar pencarian memberikan suatu kunci bagi banyak sejarah penyelesaian yang penting dalam bidang AI. Ada beberapa aplikasi yang menggunakan teknik pencarian ini, yaitu :

1. Papan *game* dan *puzzle* (tic-tac-toe, catur, menara hanoi).
2. Penjadwalan dan masalah *routing* (*travelling salesman problem*).
3. Parsing bahasa dan inteprestasinya (pencarian struktur dan arti).
4. Logika pemrograman (pencarian fakta dan implikasinya).
5. Komputer *vision* dan pengenalan pola.
6. Sistem pakar bebasis kaidah (*rule based expert sistem*).

Pencarian adalah proses mencari solusi dari suatu permasalahan melalui sekumpulan kemungkinan ruang keadaan (*state space*). Ruang keadaan merupakan suatu ruang yang berisi semua keadaan yang mungkin. Kondisi suatu pencarian meliputi :

1. Keadaan sekarang atau awal.
2. Keadaan tujuan-solusi yang dijangkau dan perlu diperiksa apakah telah mencapai sasaran.
3. Biaya atau nilai yang diperoleh dari solusi.

Solusi merupakan suatu lintasan dari keadaan awal sampai keadaan tujuan. Secara umum, proses pencarian dapat dilakukan seperti berikut :

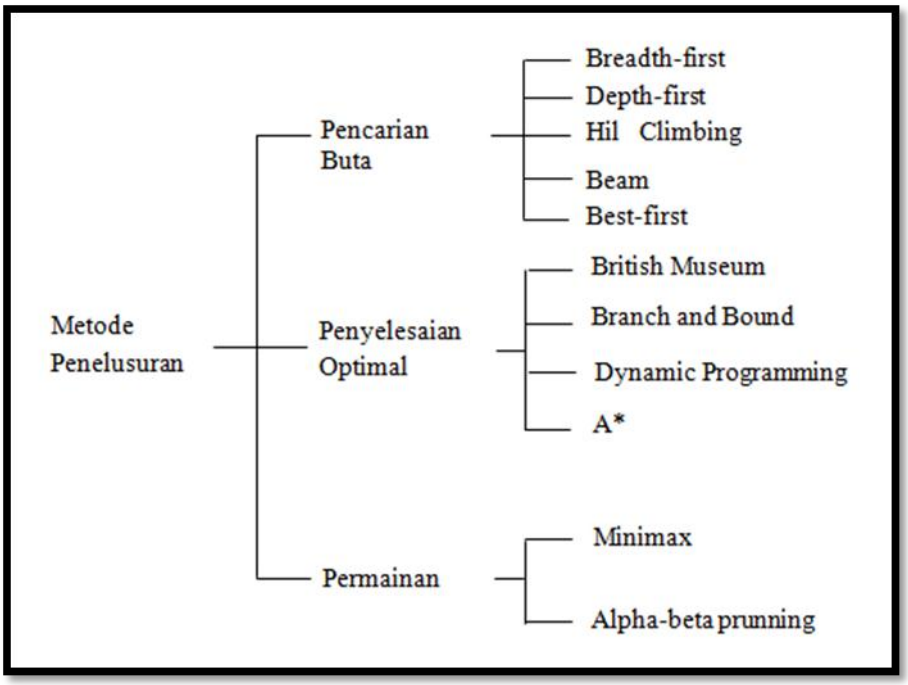
1. Memeriksa keadaan sekarang atau awal.
2. Mengeksekusi aksi yang dibolehkan untuk memindahkan ke keadaan berikutnya.
3. Memeriksa jika keadaan baru merupakan solusinya. Jika tidak, keadaan baru tersebut menjadi keadaan sekarang dan proses ini diulangi sampai solusi ditemukan atau ruang keadaan habis terpakai [12].

### 2.2.2. Algoritma Pencarian

Permasalahan pencarian adalah merupakan yang sering dijumpai oleh peneliti di bidang kecerdasan buatan. Permasalahan ini merupakan hal penting dalam menentukan keberhasilan sistem kecerdasan buatan. Metode pencarian yang pertama adalah metode yang sederhana yang hanya berusaha mencari kemungkinan penyelesaian. Metode yang termasuk pada bagian ini adalah *dept-first search*, *hill climbing*, *breadth-first search*, *beam search* dan *best-first search*.

Yang kedua, adalah metode yang lebih kompleks yang akan mencari jarak terpendek. Metode ini adalah *British Museum Procedure*, *Branch and Bound*, *Dynamic Programming* dan A*. Metode-metode ini digunakan pada saat harga perjalanan untuk mencari kemungkinan menjadi perhitungan.

Yang ketiga, adalah beberapa prosedur/metode yang terapkan saat berhadapan dengan musuh. Prosedur ini adalah *minimax search*, *alpha-beta prunning*. Metode ini banyak digunakan pada program-program permainan seperti catur dsb. Dalam Gambar 2.1 terdapat bagan untuk Metode *Searching*.




**Gambar 2.1 Metode Pencarian [12].**

Menurut cara algoritma mengembangkan node dalam proses pencarian, gambar bagan metode penulusuran dibagi menjadi dua golongan, yakni pencarian buta (*blind search*) dan pencarian terbimbing (*heuristic search*).

1. Pencarian Buta (*Blind Search*)
2. Pencarian Terbimbing (*Heuristic Search*)

### 2.2.3. Pencarian Buta (*Blind Search)*

Pencarian buta (*Blind Search)* adalah pencarian solusi tanpa adanya informasi yang dapat mengarahkan pencarian untuk mencapai *goal state* dari *current state.* Informasi yang ada hanyalah definisi *goal state* itu sendiri sehingga algoritma dapat mengenali *goal state.* Apabila tidak ada informasi maka pencarian buta dalam kerjanya akan memeriksa *node-node* secara tidak terarah dan kurang efisien untuk kebanyakan kasus karena banyaknya *node* yang dikembangkan. Beberapa contoh algoritma yang termasuk *blind search* adalah *Breadth First Search, Uniform Cost Search, Depth First Search, Depth Limited Search, Interative Deepening Search* dan *Bidirectional Search* [12].

### 2.2.4. Pencarian Terbimbing (*Heuristic Search*)

Pencarian terbimbing (*Heuristic Search)* mempunyai informasi tentang biaya untuk mencapai *goal state* dan *current state.* Pencarian terbimbing dapat melakukan pertimbangan untuk mengembangkan atau memeriksa *node-node* yang mengarah ke *goal state.* Pencarian terbimbing untuk menghitung *cost* ke *goal state* digunakan fungsi *heuristic*. Fungsi *heuristic* berbeda dari pada algoritma dimana *heuristic* lebih merupakan perkiraan untuk membatu algoritma dan tidak harus *valid* setiap waktu. Beberapa contoh algoritma pencarian yang menggunakan metode *heuristic search* adalah *Best First Search, Greedy Search, A* (A star) dan Hill Climbing Search* [12].

### 2.2.5. Algoritma A* (A *Star*)

Algoritma A* merupakan perbaikan dari metode BFS dengan memodifikasi fungsi *heuristic.* A* akan meminimumkan total biaya lintasan. Pada kondisi yang tepat, A* akan memberikan solusi yang terbaik dalam waktu yang optimal [10].

Pada pencarian rute kasus sederhana, dimana tidak terdapat halangan pada peta, A* bekerja secepat dan seefisien BFS. Pada kasus pera dengan halangan, A* dapat menemukan solusi rute tanpa “terjebak” oleh halangan yang ada.

1. Setiap sisi mempunyai "*cost"* yang berbeda-beda, seberapa besar *cost* untuk pergi dari satu simpul ke simpul yang lain.
2. *Cost* dari setiap simpul ke simpul tujuan bisa diperkirakan. Ini membantu pencarian, sehingga lebih kecil kemungkinan untuk mencari kearah yang salah.

*Cost* untuk setiap simpul tidak harus berupa jarak. Karena *cost* bisa saja berupa waktu yang dapat mencari jalan dengan waktu tercepat untuk dilalui. Sebagai contoh, bila dengan berkendaraan melewati jalan biasa bisa saja merupakan jarak terdekat, tetapi dengan berkendaraan melewati jalan tol biasanya memekan waktu lebih sedikit.

Algoritma A* berkerja dengan prinsip yang hampir sama dengan BFS kecuali dengan dua perbedaan, yaitu :

1. Simpul-simpul di *list* "terbuka" diurutkan oleh *cost* keseluruhan dari simpul awal ke simpul tujuan, dari *cost* terkecil sampai *cost* terbesar. Dengan kata lain, menggunakan *priority queue* (antrian prioritas). *Cost* keseluruhan dihitung dari *cost* dari simpul awal ke simpul sekarang (*current node*) ditambah cost perkiraan menuju simpul tujuan.
2. Simpul di *list* "tertutup" bisa dimasukan ke *list* "terbuka" bila jalan terpendek (*cost* lebih kecil) menuju simpul tersebut ditemukan.

Karena *list* "terbuka" diurutkan berdasarkan perkiraan *cost* keseluruhan, algoritma mengecek simpul-simpul yang mempunyai perkiraan *cost* yang paling kecil terlebih dahulu, jadi algoritmanya mencari simpul-simpul yang kemungkinan mengarah ke simpul tujuan. Karena itu, lebih baik perkiraan *cost*-nya, lebih cepat pencariannya. *Cost* dan perkiraannya ditentukan dengan nilai yang bebas. Bila *cost*-nya adalah jarak, akan menjadi mudah untuk menentukan nilainya.

*Cost* antara simpul adalah jaraknya, dan perkiraan *cost* dari suatu simpul ke simpul tujuan dalah penjumlahan jarak dari simpul tersebut ke simpul tujuan. Atau untuk lebih mudah dimengerti, berikut rumusnya :

$$F(n) = G(n) + H(n) \quad (2.1)$$

Dengan :

F(n) = fungsi evaluasi

G(n) = biaya (*cost*) yang sudah dikeluarkan dari keadaan sampai keadaan n

G(n) = estimasi biaya atau perkiraan biaya untuk sampai pada suatu tujuan mulai dari n

*Node* dengan nilai terendah merupakan solusi terbaik untuk diperikas pertama kali pada G(n) + H(n). dengan fungsi *heuristic* yang memenuhi kondisi tersebut maka pencarian dengan algoritma A* dapat optimal. Keoptimalan dari A* cukup langsung dinilai optimal jika H(n) adalah *admissible heuristic* yaitu nilai H(n) tidak akan memberikan penilaian lebih pada *cost* untuk mencapai tujuan. Salah satu contoh dari a*dmissible heuristic* adalah jarak dengan menarik garis lurus karena jarak terdekat dari dua titik adalah dengan menarik garis lurus [12].

Beberapa terminologi dasar yang terdapat pada algoritma ini adalah *starting point*, *current node*, simpul, *neighbor node*, *open list*, *closed list*, harga (*cost)*, *walkability,* target *point.*

1. Simpul awal adalah sebuah terminologi untuk posisi awal sebuah benda.
2. *Current node* adalah simpul yang sedang dijalankan dalam algoritma pencarian jalan terpendek.
3. Simpul adalah petak-petak kecil sebagai representasi dari area *pathfinding*. Bentuknya dapat berupa persegi, lingkaran, maupun segitiga.
4. *Neighbor node* adalah simpul-simpul yang bertetangga dengan *current node*.
5. *Open List* adalah tempat menyimpan data simpul yang mungkin diakses dari *starting point* maupun simpul yang sedang dijalankan.
6. *Closed List* adalah tempat menyimpan data simpul sebelum *current node* yang juga merupakan bagian dari jalur terpendek yang telah berhasil didapatkan.

7. *Parent* adalah tempat menyimpan data ketetanggaan dari suatu simpul, misalnya y *parent* x artinya *neighbor node* y dari *current node* x.
8. Harga (F) adalah nilai yang diperoleh dari penjumlahan nilai G, jumlah nilai tiap simpul dalam jalur terpendek dari *starting point* ke *current node*, dan H, jumlah nilai perkiraan dari sebuah simpul ke target *point.*
9. Target *point* yaitu simpul yang dituju.
10. *Walkability* adalah sebuah atribut yang menyatakan apakah sebuah simpul dapat atau tidak dapat dilalui oleh *current node.*

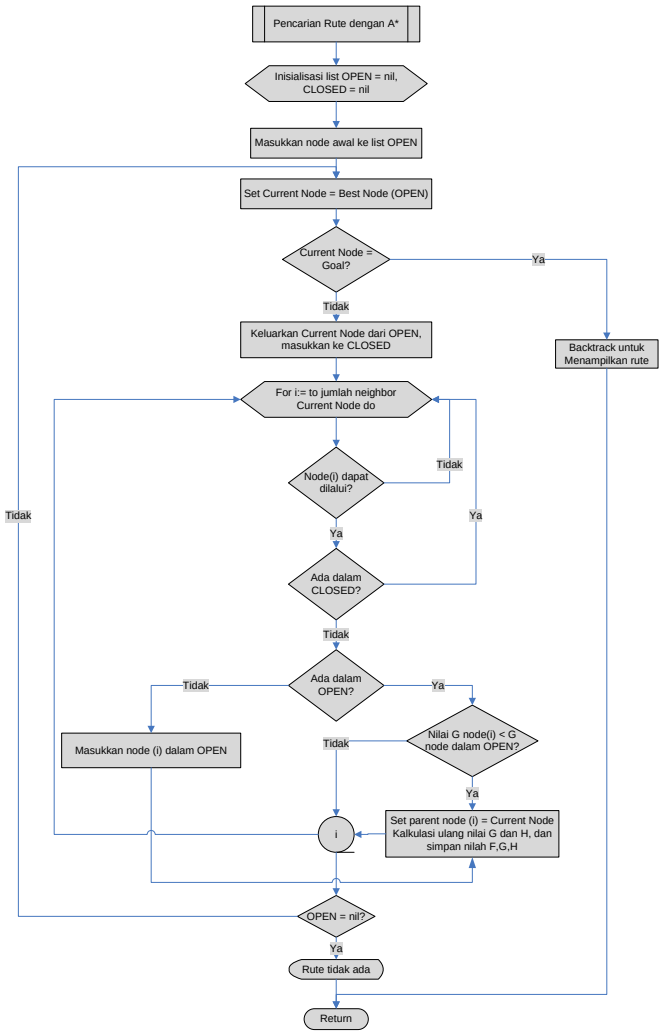
Algoritma A* secara ringkas langkah demi langkahnya adalah sebagai berikut:

1. Tambahkan simpul awal ke dalam open list.
2. Ulangi langkah berikut sampai pencarian berakhir:
    1) Carilah simpul n dengan biaya F(n) paling rendah, dalam *open list.* Simpul dengan biaya F terendah kemudian disebut *current node.*
    2) Keluarkan *current node* dari *open list* dan masukkan ke dalam *closed list.*
    3) Untuk setiap 8 simpul (*neighbor node*) dari *current node* lakukan langkah berikut:
        a. Jika sudah terdapat dalam *closed list* atau tidak *walkable*, maka abaikan, jika tidak lanjutkan.
        b. Jika belum ada pada *open list*, tambahkan ke *open list*. Simpan *current node* sebagai *parent* dari *neighbor node* ini. Simpan harga F masing-masing simpul.
        c. Jika sudah ada dalam *open list*, periksa apakah ini jalan dari simpul ini ke *current node* yang lebih baik dengan menggunakan biaya G sebagai ukurannya. Simpul dengan biaya G yang lebih rendah berarti bahwa ini adalah jalan yang lebih baik. Jika demikian, buatlah simpul ini (*neighbor node*) sebagai *parent* dari *current node*, dan menghitung ulang nilai G dan F dari simpul ini.

d. Setelah mencapai target *point*, jika masih ada *neighbor node* yang memiliki nilai yang lebih kecil, maka simpul tersebut akan terus dipilih sampai bobotnya jauh lebih besar atau mencapai target *point* dengan bobot yang lebih kecil dibanding dengan simpul sebelumnya yang telah mencapai target *point.*

e. Pada saat pemilihan simpul berikutnya, nilai F(n) akan dievaluasi, dan jika terdapat nilai F(n) yang sama maka akan dipilih berdasarkan nilai G(n) terbesar.

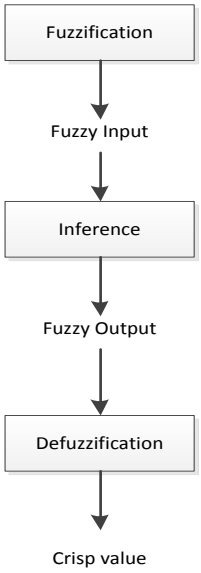
3. Simpan rute atau jalan. Bekerja mundur dari target *point*, pergi dari masing-masing simpul ke simpul *parent* sampai mencapai *starting point.* Secara umum dapat digambarkan dengan *flowchart* pada gambar berikut ini :




**Gambar 2.2 *Flowchart* Algoritma A* (Astar) [13].**

### 2.2.6. *Fuzzy Logic*

Konsep tentang logika *fuzzy* diperkenalkan oleh prof. Lotfi Astor Zadeh pada tahun 1962. Logika *fuzzy* adalah metologi sistem kontrol pemecahan masalah, dapat diterapkan pada perangkat keras, perangkat lunak, atau kombinasi keduanya. Dalam logika klasik dinyatakan bahwa segala sesuatu bersifat biner, artinya adalah hanya mempunyai duakemungkinan, "Ya atau Tidak", "Benar atau Salah", "Baik atau Buruk" dan lain-lain. Oleh karena itu, semua ini dapat mempunyai nilai keanggotaan 0 dan 1. Artinya, bias saja suatu keadaan mempunyai nilai "Ya dan Tidak", "Benar atau Tidak", "Baik dan Buruk" secara bersamaan, namunbesar nilainya tergantung pada bobot keanggotaan yang dimilikinya. Dalam banyak hal, logika *fuzzy* digunakan sebagai suatu cara untuk memetakan permasalahan dari *input* menuju ke *ouput* yang diharapkan, Konsep himpunan *fuzzy* memiliki 2 atribut, yaitu [13] :




**Gambar 2.3 Struktur Sistem *Fuzzy* [13]**

Keterangan:

1) *Fuzzification*

   Masukan-masukan yang nilai kebenarannya bersifat pasti (*crispt input*) dikonversi ke bentuk *fuzzy input*, yang berupa nilai *linguistik* yang semantiknya ditentukan berdasarkan fungsi keanggotaan.

2) *Inference*

Memperhitungkan semua aturan yang ada dalam basis pengetahuan. Hasil dari proses inference dipresentasikan oleh suatu *fuzzy* set untuk setiap variabel bebas. Derajat keanggotaan untuk setiap nilai variabel tidak bebas menyatakan ukuran kompatibilitas terhadap variabel bebas.

3) *DeFuzzyfication*

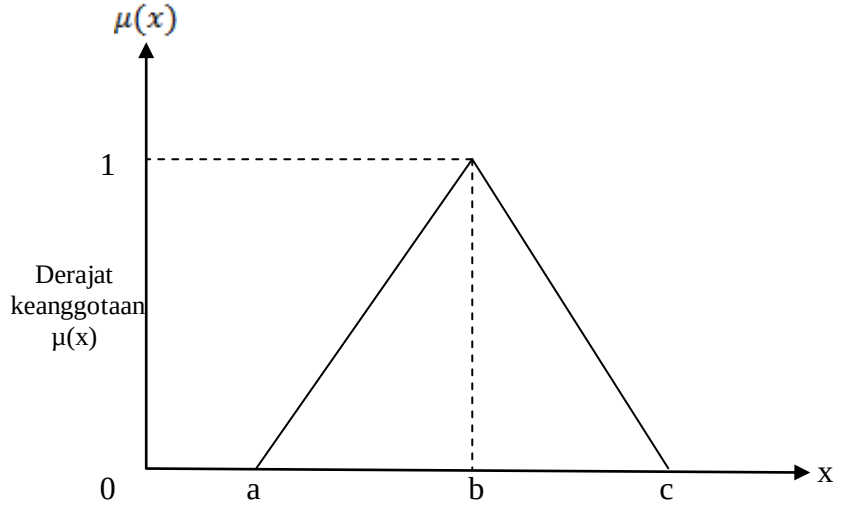
merupakan proses mengubah *output fuzzy* yang diperoleh dari mesin inferensi menjadi nilai tegas menggunakan fungsi keanggotaan yang sesuai dengan saat dilakukan *fuzzyfikasi.*

**2.2.6.1 Fungsi Keanggotaan**

Pada *fuzzy system*, fungsi keanggotaan memainkan peran yang sangat penting untuk merepresentasikan masalah dan menghasilkan keputusan yang akurat. Terdapat banyak sekali fungsi keanggotaan yang dapat digunakan, diantaranya adalah [14] :

1. Fungsi Segitiga

Fungsi segitiga hanya terdapat satu nilai x yang memiliki derajat keanggotaan sama dengan 1, yaitu ketika x = b. Tetapi, nilai-nilai disekitar b memiliki derajat keanggotaan yang turun cukup tajam (menjauhi 1). Grafik dan notasi matematika dari fungsi segitiga dapat dilihat pada gambar 2.4.



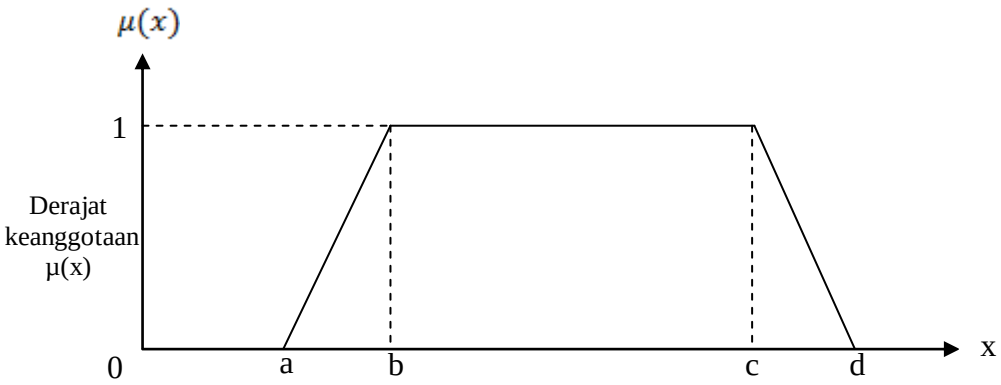

**Gambar 2.4 Grafik Fungsi Segitiga [14]**

Fungsi keanggotaan :

$$\mu(x) = \begin{cases} 0;\ x \leq a\ atau\ x \geq c \\ (x-a)/(b-a);\ a \leq x \leq b \\ (c-x)/(c-b);\ b \leq x \leq c \end{cases} \quad (2.2)$$

2. Fungsi Trapesium

Berbeda dengan fungsi segitiga, pada fungsi ini terdapat beberapa nilai x yang memiliki derajat keanggotaan sama dengan 1, yaitu ketika $b \leq x \leq c$. Tetapi, derajat keanggotaan untuk $c \leq x \leq d$ memiliki karakteristik yang sama dengan fungsi segitiga. Grafik dan notasi matematika dari fungsi Trapesium dapat dilihat pada gambar 2.5.




**Gambar 2.5 Grafik Fungsi Trapesium [14]**

Fungsi Keanggotaan :

$$\mu(x) = \begin{cases} 0; & x \leq a \text{ atau } x \geq d \\ (x-a)/(b-a); & a \leq x \leq b \\ 1; & b \leq x \leq c \\ (d-x)/(d-c); & c \leq x \leq d \end{cases} \quad (2.3)$$

### 2.2.6.2 Metode Sugeno

*Fuzzy* metode sugeno merupakan metode inferensi *fuzzy* untuk aturan yang direpresentasikan dalam bentuk IF – THEN, dimana *output* (konsekuen) sistem tidak berupa himpunan *fuzzy*, melainkan berupa konstanta atau persamaan linear. Metode ini diperkenalkan oleh Takagi-Sugeno Kang pada tahun 1985. Model Sugeno menggunakan fungsi keanggotaan Singleton yaitu fungsi keanggotaan

yang memiliki derajat keanggotaan 1 pada suatu nilai *crisp* tunggal dan 0 pada nilai *crisp* yang lain. Penalaran dengan metode Sugeno hampir sama dengan penalaran Mamdani, hanya saja *output* (konsekuen) sistem tidak berupa himpunan *fuzzy*, melainkan berupa konstanta atau persamaan linear. Michio Sugeno mengusulkan penggunaan singleton[14].

Tahapan-tahapan dalam metode Sugeno yaitu sebagai berikut:

1. Pembentukan himpunan *Fuzzy*

   Pada tahapan ini variabel *input* (*crisp*) dari sistem *fuzzy* ditransfer ke dalam himpunan *fuzzy* untuk dapat digunakan dalam perhitungan nilai kebenaran dari premis pada setiap aturan dalam basis pengetahuan. Dengan demikian tahap ini mengambil nilai-nilai *crisp* dan menentukan derajat di mana nilai-nilai tersebut menjadi anggota dari setiap himpunan *fuzzy* yang sesuai.

2. Aplikasi fungsi implikasi

   Tiap-tiap aturan (proposisi) pada basis pengetahuan *fuzzy* akan berhubungan dengan suatu relasi *fuzzy*. Bentuk umum dari aturan yang digunakan dalam fungsi implikasi adalah sebagai berikut:

   IF x is A AND y is B THEN z is f(x,y) (2.4)

   Dengan x dan y adalah skalar, dan A dan B adalah himpunan *fuzzy*. Proposisi yang mengikuti IF disebut sebagai antesenden sedangkan proposisi yang mengikuti THEN disebut konsekuen. Proposisi ini dapat diperluas dengan menggunakan operator *fuzzy* seperti,

   IF(x1 is A1) o (x2 is A2) o (x3 is A3) o…o (xN is AN) THEN y is B (2.5)

   dengan o adalah operator (misal: OR atau AND).

   Fungsi ini akan menskala *output* himpunan *fuzzy*. Pada metode Sugeno ini , fungsi implikasi yang digunakan adalah fungsi min.

3. Defuzzifikasi ( *Defuzzification* )

   *Input* dari proses defuzzifikasi adalah himpunan *fuzzy* yang dihasilkan dari proses komposisi dan *output* adalah sebuah nilai (*crisp*).

Defuzzifikasi pada metode Sugeno lebih sederhana, karena hanya menghitung *center of single-ton* atau titik pusat nilai *crisp* dengan metode rata-rata (*average*)

$$z^* = \frac{\Sigma \mu_i . z_i}{\Sigma \mu_i} \quad . \qquad (2.6)$$

dengan $z^*$ = titik pusat nilai *crisp*

$\mu_i$ = derajat keanggotaan masing-masing variabel

$z_i$ = nilai domain pada variabel linguistik $z$

## 2.3. *Object Oriented*

### 2.3.1. *Object Oriented* Analisis

Berorientasi objek merupakan teknik atau cara pendekatan dalam melihat permasalahan dan sistem (sistem perangkat lunak atau sistem lainnya). Pendekatan berorientasi objek akan memandang sistem yang akan dikembangkan sebagai suatu kumpulan objek yang berkorespondensi dengan objek-objek dunia nyata.[15]

Keuntungan menggunakan metode berorientasi objek adalah sebagai berikut :

1. Meningkatkan Produktivitas

   Karena kelas dan objek yang ditemukan dalam suatu masalah masih dapat dipakai ulang untuk masalah lainnya yang melibatkan objek tersebut (*Reusable*).

2. Kecepatan Pengembangan

   Karena sistem yang dibangun dengan baik dan benar pada saat analisis dan perancangan akan menyebabkan berkurangnya kesalahan pada saat pengkodean.

3. Kemudahan Pemeliharaan

   Karena dengan model objek, pola-pola yang cenderung tetap dan stabil dapat dipisahkan dan pola-pola yang mungkin sering berubah-ubah.

4. Adanya Konsistensi

   Karena sifat pewarisan dan penggunaan notasi yang sama pada saat analisis, perancangan maupun pengkodean.

5. Meningkatkan Kualitas Perangkat lunak

   Karena pendekatan pengembangan lebih dekat dengan dunia nyata dan adanya konsistensi pada saat pengembangannya, perangkat lunak yang dihasilkan akan mampun memenuhi kebutuhan pemakai serta mempunyai sedikit kesalahan.

### 2.3.2. OOP (*Object Oriented Programming*)

OOP (*Object Oriented Programming*) atau yang dikenal dengan Pemrograman Berorientasi Objek 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.

Dengan menggunakan OOP maka dalam melakukan pemecahan suatu masalah tidak harus melihat bagaimana cara menyelesaikan suatu masalah tersebut (terstruktur) tetapi objek-objek apa yang dapat melakukan pemecahan masalah tersebut. Sebagai contoh suatu departemen yang memiliki *manager*, sekretaris, petugas administrasi data dan lainnya. Misal manager 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 *manager* tidak harus mengetahui bagaimana cara mengambil data tersebut tetapi manager 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 [16].

#### 2.3.2.1. Istilah-Istilah OOP

Istilah OOP sudah sering didengar dalam ilmu pemrograman. Berikut ini istilah-istilah yang sering kali terdengar dalam teknik pemrograman OOP :

1. ***Object***

Untuk mempermudah pemahaman, maka disini akan dijelaskan melalui analogi. Pada dasarnya semua benda yang ada di dunia nyata dapat dianggap sebagai objek. Misalnya rumah, mobil, sepeda, motor, gelas, komputer, meja, sepatu, dll. Setiap objek memiliki atribut sebagai status (*state*) dan tingkah laku sebagai *behavior*. Contoh objek : Motor. Maka *atribute*- nya (*state*) adalah pedal, roda, jeruji, *speedometer*, warna, jumlah roda. Sedangkan tingkah laku (*behavior*) adalah kecepatan menaik, kecepatan menurun dan perpindahan gigi motor. Analogi pemrograman berorientasi objek sama dengan penggambaran pada dunia nyata seperti contoh di atas. Dalam OOP, *state* disimpan pada variabel dan tingkah laku disimpan pada *method*. Dalam bahasa teoretis OOP, Objek berfungsi untuk membungkus data dan fungsi bersama menjadi satu unit dalam sebuah program komputer. Objek merupakan dasar dari modularitas dan struktur dalam sebuah program komputer berorientasi objek.

2. ***Class***

Definisi *class* yaitu *template* untuk membuat objek. *Class* merupakan prototipe atau *blue prints* yang mendefinisikan variabel-variabel dan *method-method* secara umum. Objek merupakan hasil instansiasi dari suatu *class*. Proses pembentukan objek dari suatu kelas disebut sebagai *instantiation*. Objek disebut juga sebagai *instances*. Dalam bahasa teoritis OOP, *class* merupakan kumpulan atas definisi data dan fungsi-fungsi dalam suatu unit untuk suatu tujuan tertentu. Sebagai contoh *'class of dog'* adalah suatu unit yang terdiri atas definisi-definisi data dan fungsi-fungsi yang menunjuk pada berbagai macam perilaku/turunan dari anjing. Sebuah *class* adalah dasar dari modularitas dan struktur dalam pemrograman berorientasi object. Sebuah *class* secara tipikal sebaiknya dapat dikenali oleh seorang *non-programmer* sekalipun terkait dengan domain permasalahan yang ada dan kode yang terdapat dalam sebuah *class* sebaiknya (relatif) bersifat mandiri dan independen (sebagaimana kode tersebut digunakan

jika tidak menggunakan OOP). Dengan modularitas, struktur dari sebuah program akan terkait dengan aspek-aspek dalam masalah yang akan diselesaikan melalui program tersebut. Cara seperti ini akan menyederhanakan pemetaan dari masalah ke sebuah program ataupun sebaliknya.

**3. *Attributes***

Atribut adalah data yang membedakan antara objek satu dengan yang lainnya. Contoh Objek : Volcano Robot (*a volcanic exploration vehicle*), mempunyai atribut sebagai berikut :

*Status* → exploring, moving, returning home

*Speed* → in miles per hour

*Temperature* → in Fahrenheit degrees

Dalam *class*, atribut sering disebut sebagai variabel. Atribut dibedakan menjadi dua jenis yaitu *Instance Variable* dan *Class Variable*. *Instance variable* adalah atribut untuk tiap objek dari kelas yang sama. Tiap objek mempunyai dan menyimpan nilai atributnya sendiri. Jadi, tiap objek dari *class* yang sama boleh mempunyai nilai yang sama atau berbeda. *Class Variable* adalah atribut untuk semua objek yang dibuat dari *class* yang sama. Semua objek mempunyai nilai atribut yang sama. Jadi semua objek dari *class* yang sama mempunyai hanya satu nilai yang *value* nya sama.

1. *Behavior*

   *Behavior*/tingkah laku adalah hal-hal yang bisa dilakukan oleh objek dari suatu *class*. Behavior dapat digunakan untuk mengubah nilai atribut suatu objek, menerima informasi dari objek lain, dan mengirim informasi ke objek lain untuk melakukan suatu tugas (*task*).

   Contoh behavior untuk objek Volcano Robot:

   a. *check current temperature*
   b. *begin a survey*
   c. *report its current position*

Dalam *class*, behavior disebut juga sebagai *methods. Methods* sendiri adalah serangkaian *statements* dalam suatu *class* yang menangani suatu *task* tertentu. Cara objek berkomunikasi dengan objek yang lain adalah dengan menggunakan *method*.

2. Abstraksi

Abstraksi adalah kemampuan sebuah program untuk melewati aspek informasi yang diproses olehnya, yaitu kemampuan untuk memfokus pada inti. Setiap objek dalam sistem melayani sebagai model dari "pelaku" abstrak yang dapat melakukan kerja, laporan dan perubahan keadaannya dan berkomunikasi dengan objek lainnya dalam sistem, tanpa mengungkapkan bagaimana kelebihan ini diterapkan. Proses, fungsi atau metode dapat juga dibuat abstrak dan beberapa teknik digunakan untuk mengembangkan sebuah pengabstrakan.

### 3.2.3.2. Konsep-Konsep OOP

Konsep dasar dari Pemrograman Berorientasi Objek Pemrograman orientasi-objek menekankan konsep berikut:

1. **Enkapsulasi (*Encapsulation*)**

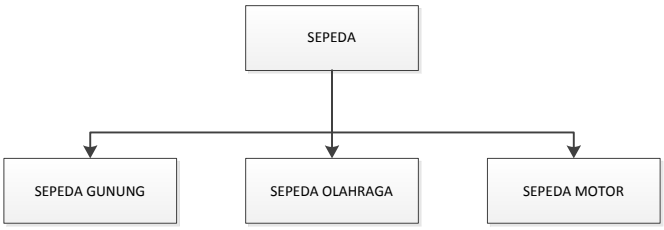
Definisi enkapsulasi adalah Pembungkusan variabel dan *method* dalam sebuah objek yang terlindungi serta menyediakan *interface* untuk mengakses variabel tersebut. Variabel dan method yang dimiliki oleh suatu objek, bisa ditentukan hak aksesnya. Dalam OOP, konsep enkapsulasi sebenarnya merupakan perluasan dari struktur dalam bahasa C. Contoh: jam tangan. Dalam hal ini, penting sekali untuk mengetahui waktu, sedangkan cara jam mencatat waktu dengan baik antara jam bertenaga baterai atau bertenaga gerak tidaklah penting untuk diketahui. Dengan kata lain enkapsulasi berfungsi untuk memastikan pengguna sebuah objek tidak dapat mengganti keadaan dalam/dari sebuah objek dengan cara yang tidak layak, hanya metode dalam objek tersebut yang diberi izin untuk mengakses keadaannya. Setiap objek mengakses *interface* yang menyebutkan bagaimana objek lainnya dapat berinteraksi dengannya. Objek

lainnya tidak akan mengetahui dan tergantung kepada representasi dalam objek tersebut.

**2. Pewarisan (*Inheritance*)**

Pewarisan merupakan pewarisan atribut dan method dari sebuah *class* ke *class* lainnya. *Class* yang mewarisi disebut *superclass* dan *Class* yang diwarisi disebut *subclass*. *Subclass* bisa berlaku sebagai *superclass* bagi *class* lainya, disebut sebagai *multilevel inheritance*. Contoh : terdapat *class* sepeda dan sepeda gunung. Sepeda termasuk superclass. Sepeda gunung termasuk *subclass*. Hal ini dikarenakan sepeda gunung memiliki variabel dan *method* yang dimiliki oleh sepeda. Prinsip dasar *inheritance* yaitu persamaan-persamaan yang dimiliki oleh beberapa kelas dapat digabungkan dalam sebuah *class* induk sehingga setiap kelas yang diturunkannya memuat hal-hal yang spesifik untuk kelas yang bersangkutan.

Contoh Pewarisan :




**Gambar 2.6 Contoh pewarisan**

Keuntungan pewarisan

a. *Subclass* menyediakan *state/behaviour* yang spesifik yang membedakan dengan *superclass*, sehingga memungkinkan *programmer* untuk menggunakan ulang *source code* dari *superclass* yang telah ada.
b. *Programmer* dapat mendefinisikan *superclass* khusus yang bersifat generik, yang disebut *abstract class* (abstraksi), untuk mendefinisikan *class* dengan tingkah laku dan *state* secara umum.

### *3. Polimorfisme*

*Polimorfisme* adalah kemampuan suatu obyek untuk mempunyai lebih dari satu bentuk. Polimorfisme tidak bergantung kepada pemanggilan subrutin. Metode tertentu yang berhubungan dengan sebuah pengiriman pesan tergantung kepada objek tertentu di mana pesan tersebut dikirim. Contohnya, bila sebuah burung menerima pesan "gerak cepat", maka akan menggerakan sayapnya dan terbang. Bila seekor singa menerima pesan yang sama, maka akan menggerakkan kakinya dan berlari. Keduanya menjawab sebuah pesan yang sama, namun yang sesuai dengan kemampuan hewan tersebut. Ini disebut polimorfisme karena sebuah variabel tunggal dalam program dapat memegang berbagai jenis objek yang berbeda selagi program berjalan dan teks program yang sama dapat memanggil beberapa metode yang berbeda di saat yang berbeda dalam pemanggilan yang sama, hal ini berlawanan dengan bahasa fungsional yang mencapai polimorfisme melalui penggunaan fungsi kelas-pertama.

## 2.4. UML (*Unified Modeling Language*)

*Unified Modelling Language* (UML) adalah sebuah “bahasa” yg telah menjadi standar dalam industri untuk visualisasi, merancang dan mendokumentasikan sistem piranti lunak. UML menawarkan sebuah standar untuk merancang model sebuah sistem. Dengan menggunakan UML dapat membuat model untuk semua jenis aplikasi piranti lunak, dimana aplikasi tersebut dapat berjalan pada piranti keras, sistem operasi dan jaringan apapun, serta ditulis dalam bahasa pemrograman apapun. Tetapi karena UML juga menggunakan *class* dan *operation* dalam konsep dasarnya, maka ia lebih cocok untuk penulisan piranti lunak dalam bahasa bahasa berorientasi objek seperti C++, Java, C# atau VB.NET. Walaupun demikian, UML tetap dapat digunakan untuk modeling aplikasi prosedural dalam VB atau C. Seperti bahasa-bahasa lainnya, UML mendefinisikan notasi dan syntax /semantik. Notasi UML merupakan sekumpulan bentuk khusus untuk menggambarkan berbagai diagram piranti lunak. Setiap bentuk memiliki makna tertentu, dan UML *syntax* mendefinisikan bagaimana bentuk-bentuk tersebut dapat dikombinasikan. Notasi UML terutama diturunkan dari 3 notasi yang telah ada sebelumnya: Grady Booch OOD (*Object-Oriented*

*Design*), Jim Rumbaugh OMT (*Object Modeling Technique*) dan Ivar Jacobson OOSE *(Object-Oriented Software Engineering*).

Sejarah UML sendiri cukup panjang. Sampai era tahun 1990 puluhan metodologi pemodelan berorientasi objek telah bermunculan di dunia. Diantaranya adalah: metodologi booch, metodologi coad, metodologi OOSE, metodologi OMT, metodologi shlaer-mellor, metodologi wirfs-brock, dsb. Masa itu terkenal dengan masa perang metodologi (*method war*) dalam pendesainan berorientasi objek. Masing-masing metodologi membawa notasi sendiri-sendiri, yang mengakibatkan timbul masalah baru apabila dalam bekerjasama dengan group/perusahaan lain yang menggunakan metodologi yang berlainan.

### 2.4.1. Konseps Dasar UML

Dari berbagai penjelasan rumit yang terdapat di dokumen dan buku-buku UML. Sebenarnya konsepsi dasar UML bisa dirangkumkan dalam tabel 2.1.

**Tabel 2.1 Konsep Dasar UML**

| ***Major Area*** | ***View*** | ***Diagrams*** | ***Main Concepts*** |
|---|---|---|---|
| *structural* | *static view* | *class diagram* | *class, association, generalization, dependency, realization, interface* |
| | *use case view* | *use case diagram* | *use case, actor, ssociation, extend, include, use casegeneralization* |
| | *implementation view* | *component diagram* | *component, interface, dependency, realization* |
| *dynamic* | *state machine view* | *statechart diagram* | *state, event, transition, action* |
| | *actifity view* | *activity diagram* | *state, activity, completion transition, fork, join* |

| | | | |
|---|---|---|---|
| | | *sequence diagram* | *interaction, object, message, activation* |
| | *interaction view* | *colaborating diagram* | *collaborating, interaction, collaboration rule, message* |
| *model management* | *model management view* | *class diagram* | *package, subsystem, model* |
| *extensibility* | *all* | *all* | *constraint, stereotype, tagged values* |

Abstraksi konsep dasar UML yang terdiri dari *structural classification*, *dynamic behavior* dan *model management*, bisa dipahami dengan mudah apabila ketika melihat gambar diatas dari Diagram. *Main concepts* bisa dipandang sebagai *term* yang akan muncul pada saat dalam membuat diagram [17]. V*iew* adalah kategori dari diagaram tersebut. Dua hal yang harus di perhatikan:

1. Menguasai pembuatan diagram UML.
2. Menguasai langkah-langkah dalam analisa dan pengembangan dengan UML.

Seperti juga tercantum pada gambar diatas UML mendefinisikan diagram-diagram sebagai berikut:

**a. *Use Case Diagram***

*Use case diagram* menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah "apa" yang diperbuat sistem dan bukan "bagaimana". Sebuah *use case* merepresentasikan sebuah interaksi antara aktor dengan sistem. *Use case* merupakan sebuah pekerjaan tertentu, misalnya *login* ke sistem, meng- *create* sebuah daftar belanja, dan sebagainya. Seorang/sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu. *Use case* diagram dapat sangat membantu dalam menyusun *requirement* sebuah sistem, mengkomunikasikan

rancanga dan merancang *test case* untuk semua *feature* yang ada pada sistem. Sebuah *use case* dapat meng- *include* fungsionalitas *use case* lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa *use case* yang di-*includ*e akan dipanggil setiap kali *use case* yang meng- *include* dieksekusi secara normal. Sebuah *use case* dapat di-*include* oleh lebih dari satu *use case* lain, sehingga duplikasi fungsionalitas dapat dihindari dengan cara menarik keluar fungsionalitas yang *common* . Sebuah *use case* juga dapat meng-*extend use case* lain dengan *behaviou*r-nya sendiri. Sementara hubungan generalisasi antar *use case* menunjukkan bahwa *use case* yang satu merupakan spesialisasi dari yang lain.

**b. *Class Diagram***

*Class* adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek. Class menggambarkan keadaan (atribut/properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda/fungsi). *Class diagram* menggambarkan struktur dan deskripsi *class*, *package* dan objek beserta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi, dan lain-lain.

Class memiliki tiga area pokok :

1. Nama (dan *stereotype*)
2. Atribut
3. Metoda

Atribut dan metoda dapat memiliki salah satu sifat berikut :

a. *Private* , tidak dapat dipanggil dari luar *class* yang bersangkutan.
b. *Protected*, hanya dapat dipanggil oleh *class* yang bersangkutan dan anak-anak yang mewarisinya.
c. *Public*, dapat dipanggil oleh siapa saja.

*Class* dapat merupakan implementasi dari *sebuah interface*, yaitu *class* abstrak yang hanya memiliki metoda. *Interface* tidak dapat langsung diinstansiasikan, tetapi harus diimplementasikan dahulu menjadi sebuah *class*. Dengan demikian *interface* mendukung resolusi metoda pada saat *run-time*. Sesuai dengan perkembangan *class model*, *class* dapat dikelompokkan menjadi *package*. Bisa juga dalam membuat diagram yang terdiri atas *package*. Hubungan antar c*lass* terbagi menjadi :

1. Asosiasi, yaitu hubungan statis antar *class.* Umumnya menggambarkan *class* yang memiliki atribut berupa *class* lain, atau *class* yang harus mengetahui eksistensi class lain. Panah *navigability* menunjukkan arah *query* antar *class*.
2. Agregasi, yaitu hubungan yang menyatakan bagian ("terdiri atas..").
3. Pewarisan, yaitu hubungan hirarkis antar *class. Class* dapat diturunkan dari *class* lain dan mewarisi semua atribut dan metoda *class* asalnya dan menambahkan fungsionalitas baru, sehingga ia disebut anak dari *class* yang diwarisinya. Kebalikan dari pewarisan adalah generalisasi.
4. Hubungan dinamis, yaitu rangkaian pesan ( *message* ) yang di- *passing* dari satu *class* kepada *class* lain. Hubungan dinamis dapat digambarkan dengan menggunakan *sequence diagram.*

**c. *Activity Diagram***

*Activity diagrams* menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing aliran berawal, *decision* yang mungkin terjadi dan bagaimana itu bisa berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi. *Activity diagram* merupakan *state* diagram khusus, di mana sebagian besar *state* adalah *action* dan sebagian besar transisi di-*trigger* oleh selesainya *state* sebelumnya (*internal processing*). Oleh karena itu *activity* diagram tidak menggambarkan *behaviour internal* sebuah sistem (dan interaksi antar subsistem) secara eksak, tetapi lebih menggambarkan proses-proses dan jalurjalur aktivitas dari *level* atas secara umum. Sebuah aktivitas dapat direalisasikan oleh satu *use case* atau lebih. Aktivitas menggambarkan proses yang berjalan, sementara *use*

*case* menggambarkan bagaimana aktor menggunakan sistem untuk melakukan aktivitas. Sama seperti *state* , standar UML menggunakan segiempat dengan sudut membulat untuk menggambarkan aktivitas. *Decision* digunakan untuk menggambarkan *behaviour* pada kondisi tertentu. Untuk mengilustrasikan proses-proses paralel (*fork* dan *join*) digunakan titik sinkronisasi yang dapat berupa titik, garis horizontal atau vertikal. *Activity diagram* dapat dibagi menjadi beberapa object *swimlane* untuk menggambarkan objek mana yang bertanggung jawab untuk aktivitas tertentu.

**d. *Sequence Diagram***

*Sequence diagram* menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, *display* dan sebagainya) berupa *message* yang digambarkan terhadap waktu. *Sequence* diagram terdiri atar dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait). *Sequence diagram* biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai *respons* dari sebuah *event* untuk menghasilkan output tertentu. Diawali dari apa yang men-*trigger* aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara internal dan output apa yang dihasilkan. Masing-masing objek, termasuk aktor, memiliki *lifeline vertikal*. *Message* digambarkan sebagai garis berpanah dari satu objek ke objek lainnya. Pada *fase* desain berikutnya, *message* akan dipetakan menjadi operasi/metoda dari *class*. *Activation bar* menunjukkan lamanya eksekusi sebuah proses, biasanya diawali dengan diterimanya sebuah *message*. Untuk objek-objek yang memiliki sifat khusus, standar UML mendefinisikan *icon* khusus untuk objek *boundary*, *controller* dan *persistent entity* .

## 2.5. *Tools* yang Digunakan

Dalam membangun aplikasi *game Third Person Shooter 3D* Selamatkan Orangutan dibutuhkan *tools* yang dapat digunakan dalam pembangunan *game* tersebut.

### 2.5.1. Unity3D 4.0.

Unity3D adalah sebuah *Game engine* yang berbasis *cross-platform*. Unity dapat digunakan untuk membuat sebuah *game* yang bisa digunakan pada perangkat komputer, ponsel pintar android, iPhone, PS3, dan bahkan X-BOX. Unity adalah sebuah sebuah tool yang terintegrasi untuk membuat *game*, arsitektur bangunan dan simulasi. Unity bisa *untuk games* PC dan *games Online*. Untuk *games online* diperlukan sebuah *plugin*, yaitu Unity *Web Player*, sama halnya dengan Flash Player pada *Browser*. Unity tidak dirancang untuk proses desain atau modelling, dikarenakan unity bukan *tools* untuk mendesain. Jika ingin mendesain, pergunakan 3D editor lain seperti 3dsmax atau *Blender*. Banyak hal yang bisa dilakukan dengan unity, ada fitur *audio reverb zone, particle effect, dan Sky Box* untuk menambahkan langit.




**Gambar 2.*7* Logo Unity3D**

Fitur *scripting* yang disediakan, mendukung 3 bahasa pemrograman, JavaScript, C# dan Boo. *Flexible and Easy Moving*, *rotating* dan *scaling objects* hanya perlu sebaris kode. Begitu juga dengan *Duplicating*, *removing*, dan *changing properties*. *Visual Properties Variables* yang di definisikan dengan *scripts* ditampilkan pada *Editor*. Bisa digeser, di *drag* and *drop*, bisa memilih warna dengan *color picker*. Berbasis .NET, artinya penjalanan program dilakukan dengan *Open Source* NET *platform*, Mono.

Unity menggunakan sebuah konsep yang disebut *Parenting*, ini digunakan untuk membuat sebuah *game object* menjadi anak dari g*ame object* yang lain. Tarik sebuah *game object* dan pindahkan tepat di atas tulisan *game objec*t yang akan dijadikan parent dalam hierarchy. *Game object* yang terdapat dalam sebuah *game object* lainnya akan mengikuti perpindahan dan perputaran ketika *game object parent* mengalami perubahan posisi.

### 2.5.2. *Adobe Photoshop* CS6

*Adobe Photoshop* adalah sebuah program yang digunakan untuk melakukan perubahan-perubahan (manipulasi) pada gambar, yang dapat menghasilkan gambar digital dengan kualitas tinggi. Adobe Photoshop juga dapat melakukan perubahan-perubahan dari gambar atau foto hasil *scanner*. Misalnya dengan memperbesar atau memperkecil gambar yang sudah ada, melakukan rotasi (perputaran), merubah warna, memberi efek-efek khusus seperti yang terdapat pada menu *filter*. Hal yang utama dalam Adobe Photoshop adalah penggunaan istilah :

1. ***Image***

   Gambar yang tampak di monitor komputer sebagai salah satu *file* dokumen Adobe Photoshop setelah proses *scanning*. Komputer dengan *software* windows terbaru umumnya bekerja dengan jenis *image* .

   a. *Grafis pixel*

      Ideal untuk memproduksi foto dan *image* lain yang berisi sejumlah besar detail yang tidak terstruktur. *Pixel* akan menyimpan informasi suatu bentuk (lingkaran, kotak, gambar) dalam titik-titik kecil dengan kerapatan tertentu. Biasanya image *pixel* kalau diperbesar akan menimbulkan efek buruk pada gambar (gambar pecah). Karena titik-titik kecil pembentuknya akan terlihat membesar dan ruang kosong antara titik-titik tersebut akan terlihat jelas. Warna akan terurai (pecah).

*b. Grafis Vektor*

Merupakan deskripsi matematika dari bentuk yang dapat dibuat dari garis lurus dan kurva. *Grafis vektor* sangat kompak dan *portable* karena dapat diperbesar dan diperkecil tanpa menimbulkan efek buruk pada gambar (gambar pecah). Contohnya apabila menggunakan *software* pengolah vektor atau ilustrasi seperti *Ilustrator, Freehand* dan *Corel Draw*. Program pengolah *vektor* ini sering disebut *resolution independent*.

**2. *Pixel* (*picture element*)**

Titik terkecil yang menyusun image. Kerapatan dari *pixel* dikenal dengan istilah Resolusi *Image*. Satuannya ada 2 macam yaitu pixel/inch atau *pixel*/cm. Contoh gambar dengan 72 *pixel/inch* artinya dalam 1 inch terdapat 72 *pixel*.




**Gambar 2.8 Logo *Adobe Photoshop* CS6**

### *2.5.3. Blender*

*Blender* adalah program 3D dan animasi yang bersifat *open source*, bebas untuk dikembangkan oleh penggunanya dan dapat didistribusikan kembali dan bersifat Legal. *Blender* memiliki *video compositor* dan *intergrated game engine*. Karya yang dihasilkan tidak ada sifat *royalt* kepada *developer* dan dapat dipublikasikan baik *free* maupun untuk dikomersilkan. Blender merupakan salah satu program *Modeling* 3D dan *Animation*, tapi *Blender* mempunyai kelebihan sendiri dibandingkan program *modeling* 3D lainnya. Kelebihan yang dimiliki *Blender* adalah dapat membuat *game* tanpa menggunakan program tambahan lainnya, karena *Blender* sudah memiliki

*Game Engine* sendiri dan menggunakan Python sebagai bahasa pemograman yang lebih mudah ketimbang menggunakan C++,C, dll.

Blender menggunakan *Open*GL sebagai render grafiknya yang dapat digunakan pada berbagai macam sistem operasi seperti *Windows*, *Linux* dan Mac OS X. Saat ini Blender sudah mengeluarkan versi yang terbarunya, yaitu Versi 2.49 yang lebih ditujukan untuk pembuat *game*. Karena Versi ini memiliki fitur-fitur baru yang dirancang untuk membuat tampilan *game* yang lebih realistis dari pada versi sebelumnya.




**Gambar 2.9 Logo *Blender***

### *2.5.4. Autodesk* 3ds *Max*

3D *Studio Max* (kadangkala disebut 3ds Max atau hanya MAX) adalah sebuah perangkat lunak grafik vektor 3-dimensi dan animasi, ditulis oleh *Autodesk Media & Entertainment* (dulunya dikenal sebagai *Discreet and Kinetix*. Perangkat lunak ini dikembangkan dari pendahulunya 3D *Studio* fo DOS, tetapi untuk *platform* Win32. *Kinetix* kemudian bergabung dengan akuisisi terakhir *Autodesk*, *Discreet Logic*. Versi terbaru 3Ds Max pada Juli 2005 adalah 7. 3Ds *Max Autodesk* 8 diperkirakan akan tersedia pada akhir tahun. Hal ini telah diumumkan oleh *Discreet di Siggraph* 2005.



**Gambar 2.10 Logo *Autodesk* 3ds Max**

## 2.6. Pengolahan Data

Pengolahan data secara sederhana diartikan sebagai proses mengartikan data-data lapangan sesuai dengan tujuan, rancangan, dan sifat penelitian. Misalnya dalam rancangan penelitian kuantitatif, maka angka-angka yang diperoleh melalui alat pengumpul data tersebut harus diolah secara kuantitatif, baik melalui pengolahan statistik inferensial maupun statistik deskriptif. Lain halnya dalam rancangan penelitian kualitatif, maka pengolahan data menggunakan teknik non statitistik, mengingat data-data lapangan diperoleh dalam bentuk narasi atau kata-kata, bukan angka-angka. Mengingat data lapangan disajikan dalam bentuk narasi kata-kata, maka pengolahan datanya tidak bisa dikuantifikasikan. Perbedaan ini harus dipahami oleh peneliti atau siapapun yang melakukan penelitian, sehingga penyajian data dan analisis kesimpulan penelitian relevan dengan sifat atau jenis data dan prosedur pengolahan data yang akan digunakan. Di atas dikatakan bahwa pengolahan data diartikan sebagai proses mengartikan data lapangan, yang berarti supaya data lapangan yang diperoleh melalui alat pengumpul data dapat dimaknai, baik secara kuantitatif maupun kualitatif, sehingga proses penarikan kesimpulan penelitian dapat dilaksanakan. Dengan demikian, pengolahan data tersebut dalam kaitannya dengan praktek pendidikan adalah sebagai upaya untuk memaknai data atau fakta menjadi makna.

### 2.6.1. Merancang Kuesioner

Kuesioner adalah pertanyaan terstruktur yang diisi sendiri oleh responden atau diisi oleh pewawancara yang membacakan pertanyaan dan kemudian mencatat jawaban yang diberikan. Pertanyaan yang diberikan pada kuesioner ini adalah pertanyaan menyangkut fakta dan pendapat responden, sedangkan kuesioner yang digunakan pada penelitian ini adalah kuesioner tertutup, dimana responden diminta menjawab pertanyaan dan menjawab dengan memilih dari sejumlah alternatif [18].

#### 2.6.1.1. Jenis pertanyaan dalam kuesioner

Perbedaaan pertanyaan dalam wawancara dengan pertanyaan dalam kuesioner adalah dalam wawancara memungkinkan adanya interaksi antara pertanyaan dan artinya. Dalam wawancara analis memiliki peluang untuk

menyaring suatu pertanyaan, menetapkan istilah-istilah yang belum jelas, mengubah arus pertanyaan, memberi respons terhadap pandanmgan yang rumit dan umumnya bisa mengontrol agar sesuai dengan konteksnya. Beberapa diantara peluang-peluang diatas juga dimungkinkan dalam kuesioner. Jadi bagi penganalisis pertanyaan-pertanyaan harus benar-benar jelas, arus pertanyaan masuk akal, pertanyaan-pertanyaan dari responden diantisipasi dan susunan pertanyaan direncanakan secara mendetail.

1. Pertanyaan terbuka

   pertanyaan-pertanyaan yang memberi pilihan-pilihan respons terbuka kepada responden. Pada pertanyaan terbuka antisipasilah jenis respons yang muncul. Respons yang diterima harus tetap bisa diterjemahkan dengan benar.

2. Pertanyaan tertutup

   pertanyaan-pertanyaan yang membatasi atau menutup pilihan-pilihan respons yang tersedia bagi responden.

### 2.6.2. Skala dalam Kuesioner

Penskalaan adalah proses menetapkan nomor-nomor atau simbol-simbol terhadap suatu atribut atau karakteristik yang bertujuan untuk mengukur atribut atau karakteristik tersebut. Alasan penganalisis sistem mendesain *skala* adalah sebagai berikut.

1. Untuk mengukur sikap atau karakteristik orang-orang yang menjawab *kuesioner*.
2. Agar *respoden* memilih subjek *kuesioner*.

   Empat bentuk *skala* pengukuran yaitu :

1) *Nominal* : *Skala nominal* digunakan untuk mengklasifikasikan sesuatu. Skala nominal merupakan bentuk pengukuran yang paling lemah, umumnya semua analis bisa menggunakannya untuk memperoleh jumlah total untuk setiap klasifikasi. Contoh : Apa jenis perangkat lunak yang paling sering anda gunakan ? 1 = Pengolah kata, 2 = *Spreadsheet*, 3 = Basis Data, 4 = Program *e-mail*.

2) *Ordinal* : *Skala ordinal* sama dengan *skala nominal*, juga memungkinkan dilakukannya kalsifikasi. Perbedaannya adalah dalam ordinal juga menggunakan susunan posisi. *Skala ordinal* sangat berguna karena satu kelas lebih besar atau kurang dari kelas lainnya.
3) *Nterval* : *Skala interval* memiliki karakteristik dimana interval di antara masing-masing nomor adalah sama. Berkaitan dengan karakteristik ini, operasi matematisnya bisa ditampilkan dalam data-data kuesioner, sehingga bisa dilakukan analisis yang lebih lengkap.
4) *Rasio* : *Skala rasio* hampis sama dengan *skala interval* dalam arti *interval-interval* di antara nomor diasumsikan sama. *Skala rasio* memiliki nilai *absolut nol*. *Skala rasio* paling jarang digunakan.

### 2.6.3. Skala *Likert*

Pemakaian *skala likert* digunakan untuk menganalisa *variabel perceived quality, brand loyalty*, dan keputusan pembelian konsumen. Nama skala ini diambil dari nama Rensis Likert, yang menerbitkan suatu laporan yang menjelaskan penggunaannya. Sewaktu menanggapi pertanyaan dalam skala *Likert*, responden menentukan tingkat persetujuan mereka terhadap suatu pernyataan dengan memilih salah satu dari pilihan yang tersedia. Pilihan jawaban pertanyaan kuesioner lebih dari 2 dan minimal 3. Tingkatan pada skala likert terdiri dari sangat setuju, setuju, biasa saja, tidak setuju, sangat tidak setuju, kelima penilaian tersebut diberikan bobot sebagai berikut :

**Tabel 2.2 *Skala likert***

| Skala penilaian | Bobot nilai |
|---|---|
| Sangat setuju | 4 |
| Setuju | 3 |
| Biasa saja | 2 |
| Tidak setuju | 1 |
| Sangat tidak setuju | 0 |

Skala *Likert* dalam penelitian mempunyai gradasi nilai dari sangat positif sampai sangat negatif. Lalu hasil data kuesioner tersebut diolah dengan rumus sebagai berikut :

$$Y = \frac{P}{Q} x100\% \quad (2.7)$$

Keterangan :

1. Y = Nilai presentase yang dicari
2. P = Bobot setiap pilihan jawaban
3. Q = Total bobot maksimal

## 2.7. Metode Pengujian

Pengujian perangkat lunak adalah elemen kritis dari jaminan kualitas suatu perangkat lunak dan merepresentasikan spesifikasi, desain dan pengkodeannya. Pada dasarnya, Pengujian dilakukan terhadap aplikasi untuk memastikan bahwa aplikasi dapat berjalan dengan benar sesuai dengan kebutuhan dan tujuan yang diharapkan. Oleh karena itu, program harus diuji coba terlebih dahulu untuk menemukan kesalahan-kesalahan yang terjadi[19].

*1. Black box testing*

Sebuah Pengujian dilakukan terhadap aplikasi untuk memastikan bahwa aplikasi dapat berjalan dengan benar sesuai dengan kebutuhan dan tujuan yang diharapkan. Pengujian *black bok* berfokus kepada persyaratan fungsional perangkat lunak, pengujian ini berusaha menemukan kesalahan antara lain.

1) Fungsi-fungsi yang salah atau hilang
2) Kesalahan *interface*
3) Kesalahan dalam struktur data
4) Kesalahan kinerja
5) Kesalahan *inisialisasi*

*2. White box testing*

*White Box* merupakan metode desain uji kasus yang menggunakan struktur kontrol dari desain prosedural untuk menghasilkan kasus-kasus uji. Dengan menggunakan metode uji coba *white box*, para pengembang *software* dapat menghasilkan kasus-kasus uji yang:

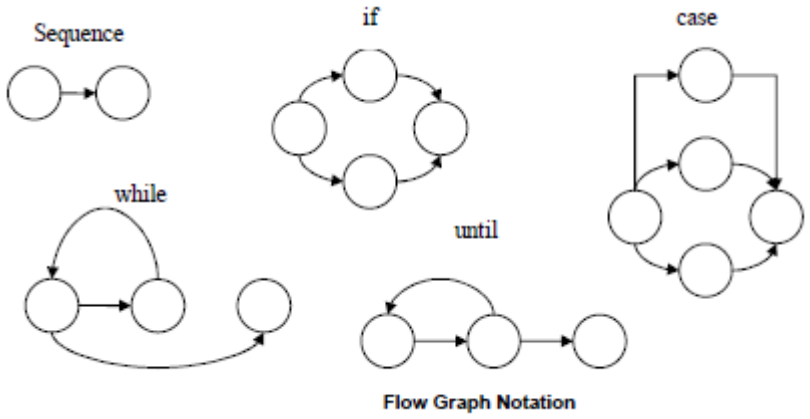
1) Menjamin bahwa seluruh *independent paths* dalam modul telah dilakukan sedikitnya satu kali.
2) Melakukan seluruh keputusan logikal baik dari sisi benar maupun salah
3) Melakukan seluruh perulangan sesuai batasannya dan dalam batasan operasionalnya
4) Menguji struktur data internal untuk memastikan validitasnya

## 2.8. Uji coba berbasis alur (basis *path tasting*)

Uji coba berbasis alur merupakan teknik uji coba *whitebox* pertama yang diusulkan oleh Tom McCabe. Metode berbasis alur memungkinkan perancang kasus uji untuk menghasilkan ukuran kompleksitas logikal dari desain prosedural dan menggunakan ukuran ini untuk mendefinisikan himpunan basis dari alur eksekusi. Kasus uji dihasilkan untuk melakukan sekumpulan basis yang dijamin untuk mengeksekusi setiap perintah dalam program, sedikitnya satu kali selama uji coba.

### 2.8.1 *Notasi graft* alur (*Path Graph Notation*)

Notasi sederhana untuk merepresentasikan alur kontrol disebut *graf* alur (*flow graph*), seperti gambar dibawah ini:




**Gambar 2.11 *Notasi Graph***

Urutan dari simbol proses dan simbol keputusan dapat digambarkan menjadi sebuah *node*, sedangkan anak panah disebut *edges*, menggambarkan aliran dari kontrol sesuai dengan diagram alir.

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*.