BAB 2
TINJAUAN PUSTAKA

2.1 *Game*

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

2.1.2 Pengertian *Game*

Game merupakan sebuah bentuk seni dimana pemainnya disebut dengan pemain (*Player*), diharuskan membuat keputusan-keputusan dengan tujuan mengelola sumber daya yang diperoleh dari kesempatan-kesempatan bermain (*token*) miliknya untuk mencapai tujuan tertentu. *Video game* adalah bentuk *game* yang interaksi umumnya melibatkan media *video* dan *audio.*

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

Berdasarkan representasinya, *game* dapat dibedakan menjadi dua jenis yaitu *game* dua dimensi (2D) dan *game* tiga dimensi (3D). *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 perhitungannya sehingga konsep kamera pada *game* 3D benar-benar menyerupai konsep kamera pada kehidupan nyata[5].

2.1.3 Elemen Dasar *Game*

Menurut Teresa Dillon, elemen-elemen dasar sebuah *game* adalah:

1. *Game Rule*

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

2. *Plot*

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

3. *Thema*

Di dalam suatu *game* biasanya ada pesan moral yang akan disampaikan.

4. *Character*

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

5. *Object*

Object merupakan suatu hal yang penting dan biasanya digunakan pemain untuk memecahkan masalah dan adakalanya pemain harus punya keahlian dan pengetahuan untuk bisa memainkannya.

6. *Text, graphic dan Sound*

Game biasanya merupakan gabungan dari media teks, gambar, suara dan animasi, walaupun tidak harus semuanya ada di dalam permainan *game*.

7. *Animation*

Sebuah game tidak akan bisa dipisahkan dengan animasi, animasi biasanya digunakan di dalam sebuah *game* untuk menggerakan karakter yang ada di dalam permainan, seperti properti dari objek.

8. *User Interface*

Merupakan fitur yang menghubungkan *user* dan sistem.

2.1.4 *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.

2.1.5 Jenis-jenis *Game*

Berikut ini beberapa jenis game berdasarkan cara pembuatannya, cara pemasarannnya dan mesin yang menjalankannnya. Jenis-jenis *game* tersebut adalah[5]:

1. *Game PC (Personal Computer)*

Game PC adalah *game* yang dimainkan pada PC (*Personal Computer*) yang memiliki kelebihan tampilan yaitu 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. Kekurangannnya adalah spesifikasi komputer yang sangat bervariasi antar satu komputer dengan komputer yang lainnya menyebabkan beberapa *game* dapat ditampilkan dengan baik pada komputer tetapi tidak berjalan dengan baik pada komputer yang lainnya.

2. *Game Consol*

Game consol 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 dimainkan secara *online* melalu LAN (*Local Area Network*) atau Internet.

2.1.6 *Genre Game*

Berdasarkan *genre* (tipe), *game* dapat dibagi menjadi beberapa *genre* (tipe) yaitu[5] :

1. *Action Game*

Action game dikategorikan sebagai *gameplay* dengan model pertarungan. Beberapa macam *game* yang termasuk dalam *genre action* yaitu :

a. Action Adventure Game

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

b. Stealth Game

Termasuk dalam *genre* baru, biasanya digolongkan dalam mata-mata yang bisa 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 *Dead Island* 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 Warior Series* dan *Final Fight*.

e. Fighting Game

Game pertarungan dua pemain dengan jurus-jurus yang bisa dikeluarkan dengan menekan beberapa tombol pada *keyboard* dengan urutan tertentu. Contoh *game* dari *genre* ini adalah *street fighter* dan *Teken series*.

f. Maze Game

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

g. Platform Game

Genre game dengan gameplay berlari, melompat , mengayun dan sebagainya. Contoh game genre ini adalahDonkey Kong dan Ray Man.

h. Shooter

a. First Person Shooter

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

b. Massively Multiplayer Online First Person Shooter

Genre game yang mengkonmbinasikan gameplay *first person shooter* dengan dunia virtual dimana banyak *player* juga ikut bermain melalui internet. Contoh *game genre* ini adalah *Counter Strike Online.*

c. ifdsfds

Game dengan *genre* yang sama seperti *First Person Shooter* yaitu mengutamakan *shooting* dan *combat* dari perspektif yang bertujuan untuk memberikan pemandangan yang lebih luas dan gerakan yang lebih banyak.

d. Tactical Shooter Game

Genre yang megutamakan perencanaan dan kerjasama 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 Game

Game dengan *genre* ciri khas gambar 2D dan *scrolling palying area*. Contoh *game* dari *genre* ini adalah *Star Fox Series*.

2. *Adventure Game*

game pertualangan yang dikategorikan sebagai *gameplay* yang mengharuskan pemain memecahkan bermacam-macam teka-teki melalui interaksi dengan orang lingkungan game tersebut.

a. *Text Adventure*

Pemain akan menggunakan *keyboard* untuk mengetikkan berupa perintah dan komputer akan menganalisa 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. *Intercative Movie Game*

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

e. *Dialog Game*

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

3. *Role Playing Game*

a. Action Role Playing Game

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

b. Massively Multiplayer Online 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 Final Fantasy Tactics.

4. *Simulation Game*

Genre game ini bertujuan untuk memberikan pengalaman simulasi kepada pemain.

a. Contruction 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 memenangkan game. Contoh game dari genre ini adalah The Sims Series.

d. Goverment Simulation Game

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

5. *Strategy Game*

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

a. Real Time Strategi

Game real time strategi, action dilakukan dalam waktu bersamaa 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. *Vehcile 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 NFS (*Need For Speed*) Series.

c. Space Game

Genre ini bersifat pertarungan diluar angkasa. Contoh *game* dari *genre* ini adalah Star Wars dan HomeWorld.

d. Trian Game

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

2.1.7 Unsur *Game*

Dalam sebuah *game* terdapat unsur-unsur yang melengkapinya. Berikut beberapa unsur dalam game yaitu[5]:

1. Warna

Warna mempunyai kemampuan untuk membuat orang tanggap terhadap semua yang dilihat karena tidak ada seseuat hal bermakna tanpa warna.Warna terlihat sebelum penampakan gambar. Mata manusia tertarik oleh warna pada suatu *level* karena warna dari objek diterima sebelum detail-detail dipisahkan oleh bentuk-bentuk dan garisnya. Warna merah memiliki panjang gelombang yang terpanjang,biru memiliki panjang gelombang yang pendek sedangkan hijau memiliki panjang gelombang menengah. Pada anak-anak cenderung tertarik pada warna-warna yang cerah dan mencolok. Warna-warna yang cerah terutama warna *primer* (merah,kuning,biru) dan warna *sekunder* (orange,ungu,hijau).



Gambar 2.1 Warna Primer dan Sekunder.

2. Komposisi

Komposis adalah pengaturan elemen di dalam sebuah karya desain yang sedemikian rupa dengan tujuan tertentu. Komposisi yang baik adalah komposisi yang mampu memenuhi kebutuhan dan tujuan desain, mudah dipahami dan membentuk kesatuan yang serasi dan harmonis. Kemudian layout yaitu perencanaan, penempatan semua unsur mulai dari tulisan, gambar, ilustrasi, teks, nama dan sebagainya dengan pengukuran secara seksama. Komposisi yang sesuai dengan anak-anak adalah komposisi yang sederhana adan tidak menggunakan petunjuk terlalu rumit. Kesederhanaan diwujudkan dengan penggunaan visual 2D dan penerapan warna-warna dalam seluruh aspek desian.

3. Bentuk Dasar

Bentuk dasar adalah bentuk-bentuk yang mudah ditemui dalam kehidupan sehari-hari seperti kotak, lingkaran, segitiga dan lain sebagainya.



Gambar 2.2 Bentuk Dasar

4. Tripografi

Tipografi merupakan representasi visual dari sebuah bentuk komunikasi verbal dan merupakan property yang pokok dan efektif. Huruf memainkan peranan penting dalam keberhasilan suatu bentuk komunikasi grafis. Dalam media pembelajaran untuk anak-anak, sebuah huruf harus legible yaitu jelas dan memiliki tingkat kemudahan untuk dibaca.

5. Audio

Audio adalah sinyal elektrik yang digunakan untuk membawa suara dalam batas pendengaran manusia. Audio merupakan komponen sistem yang mudah termasuk didalamnya atau dapat ditambahkan pada komputer.

2.2 Kecerdasan Buatan

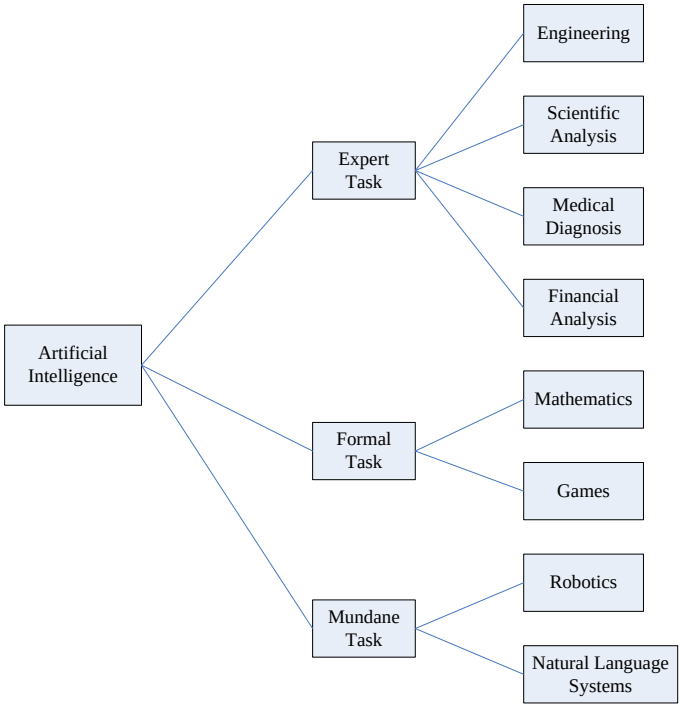
Kecerdasan buatan atau *artificial intelligence* (AI) merupakan cabang dari ilmu komputer yang berhubungan dengan pengautomatisasian tingkah laku cerdas. Kecerdasan buatan didasarkan pada teori suara (*sound theoritical*) dan prinsip-prinsip aplikasi dari bidangnya. Prinsip-prinsip ini meliputi struktur data yang digunakan dalam representasi pengetahuan, algoritma yang diperlukan untuk mengaplikasikan pengetahuan tersebut serta bahasa dan teknik pemrograman yang digunakan untuk mengimplementasikannya[6].

Berdasarkan sudut pandang AI dapat dipandang sebagai berikut :

1. Sudut pandang kecerdasan, AI adalah bagaimana membuat mesin yang cerdas dan dapat melakukan hal-hal yang sebelumnya hanya dapat dilakukan 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 dari studi area.

Kecerdasan buatan memiliki sub disiplin ilmu yang sering digunakan untuk pendekatan yang esensial bagi penyelsaian suatu masalah dan dengan aplikasi bidang AI yang berbeda. Berikut gambar bidang-bidang tugas dari AI yaitu:



Gambar 2.3 Bidang-bidang tugas (*task domain*) dari AI.

Aplikasi penggunaan AI dapat dibagi ke dalam tiga kelompok, yaitu :

1. *Expert Task*

AI dibentuk berdasarkan pengalaman dan pengetahuan yang dimiliki oleh para ahli. Pengetahuan ini dapat membantu para ahli untuk menyampaikan ilmu-ilmu yang dimiliki. Contohnya adalah anaisis finansial, analisis medical, analisis pengetahuan, dan rekayasa (desain, pencarian, kegagalan. Perencanaan, manufaktur).

2. *Formal Task*

AI digunakan untuk melakukan tugas-tugas formal yang selama ini manusia biasa laukan dengan lebih baik. Contohnya adalah *game* dan matematika (geometri, logika, kalkulus, integral).

3. *Mundane Task*

Secara harfiah *mundane* adalah keduniaan. AI digunakan untuk melakukan hal-hal yang sifatnya duniawi atau melakukan kegiatan yang dapat membantu manusia. Contohnya adalah persepsi, bahasa alami, dan *robot* control.

Aplikasi kecerdasan buatan memiliki dua bagian utama yaitu :

1. Basis pengetahuan (*knowledge base*) : berisi fakta-fakta, teori, pemikiran dan hubungan antara satu dengan lainnya.
2. Motor inferensi (*inference engine*) : kemampuan menarik kesimpulan berdasarkan pengalaman.

2.3 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[6] :

1. Papan *game* dan *puzzle* (tic-tac-toc, catur, menara honai).
2. Penjadwalan dan masalah *routing* (*travelling, salesman problem*).
3. *Parsing* bahasa dan interprestasinya (pencarian struktur dan arti).
4. Logika pemrograman (pencarian fakta dan implikasinya).

5. *Computer vision* dan pengenalan pola.
6. Sistem pakar berbasis kaidah (*rule base expert system*).

Pencarian adalah proses mencari solusi dari suatu permasalahan melalui sekumpulan kemungkanan 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, maka keadaan baru tersebut menjadi keadaan sekarang dan proses ini diulangi sampai solusi ditemukan atau ruang keadaan habis dipakai.

2.4 Algoritma Pencarian

Permasalahan pencarian dapat diselsaikan dengan beberapa metode yaitu[6] :

1. Metode pencarian yang pertama adalah metode sederhana yang hanya berusaha mencari kemungkinan penyelsain yang disebut juga pencarian buta.
2. Metode yang lebih kompleks yang akan mencari jarak terpendek. Metode ini adalah *british museum procedure, branch* dan *bound, dymanic programming, best first search,greddy search*, A* (A Star) dan *hill climbing search.* Metode-metode ini digunakan pada saat perjalanan untuk mencari kemungkinan menjadi perhitungan.

Metode pencarian sangat penting untuk menyelsaikan permasalahan karena setiap *state* atau keadaan menggambarkan langkah-langkah untuk menentukan apa yang harus dilakukan dimana setiap *state* menggambarkan menungkinan posisi pada sutau saat. Metode pencarian adalah bagian dari kesimpulan dimana setiap *state* menggambarkan hipotesis dalam sebuah rangkaian dedukatif.

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

Beberapa contoh algoritma pencarian yang menggunakan metode *heuristic Search* adalah : *Best First Search*, *Greedy Search*, A* (*A Star*) *Search*, dan *Hill Climbing Search*.

2.4.1 Pencarian Buta (*Blind Search*)

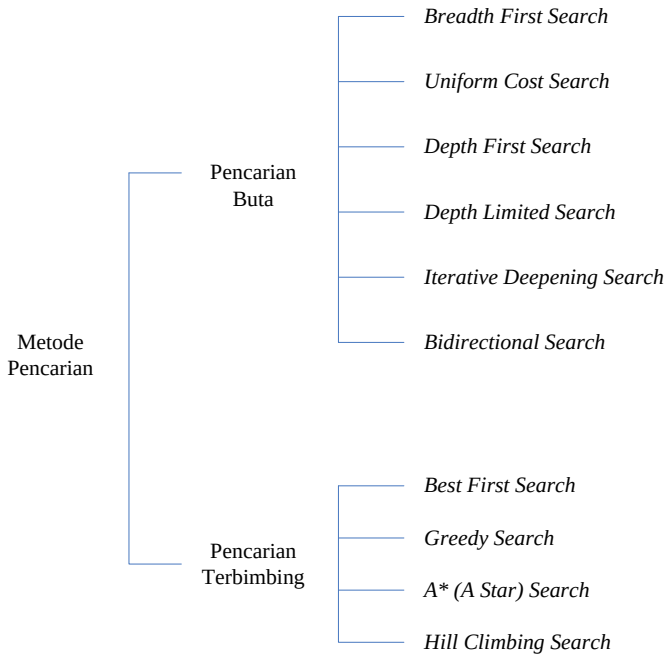
Pencarian buta (*Blind Search*) adalah pencarian solusi tanpa adanya informasi yang dapat mengarahkan pencarian untuk mencapai *goal state* dari *current state*. Informasi yang ada hanyalah definisi *goal state* itu sendiri sehingga algoritma dapat mengenali *goal state*. Apabila tidak ada informasi 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*[6].

2.4.2 Pencarian Terbimbing (*Heuristic Search*)

Pencarian terbimbing (*Heuristic Search*) mempunyai informasi tentang biaya untuk mencapai *goal state* dari *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 di mana *heuristic* lebih merupakan perkiraan untuk membantu 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*[6].

Gambar bagan metode pencarian dibagi mendari dua golongan, yaitu :



Gambar 2.4 Bagan Metode Pencarian.

2.5 Fungsi Heuristik

BFS dan A* sebagai algoritma pencarian yang menggunakan fungsi heuristik untuk "menuntun" pencarian rute, khususnya dalam hal pengembangan dan pemeriksaan *node-node* pada peta. Dalam aplikasi ini, fungsi heuristik yang digunakan untuk pencarian rute mengatasi nilai/notasi h pada algoritma BFS dan A*. Ada beberpa fungsi heuristik umum yang bisa dipakai untuk algoritma BFS dan A* ini. Salah satunya adalah yang dikenal dengan istilah "*Manhhattan Distance*". Fungsi heuristik ini digunakan untuk kasus dimana pergerakan pada peta hanya lurus (*horizontal* atau *vertical*), tidak diperbolehkan pergerakan diagonal[7].

Perhitungan nilai heuristik untuk node ke-n mengguanakn *Manhattan Distance* adalah sebagai berikut :

$$h(n) = \left(abs(n.x - goal.x) + abs(n.y - goal.y)\right) \quad (2.1)$$

Dimana h(n) adalah nilai heuristik untuk *node* n, dan goal adalah *node* tujuan.

Jika pergerakan diagonal pada peta diperbolehkan, maka fungsi heuristik selain *Manhattan Distance*. Untuk mendekati kenyataan, *cost* untuk perpindahan *node* secara *diagonal* dan *orthogonal* dibedakan. *Cost* diagonal adalah 1,4 kali *cost* perpindahan secara *orthogonal*.

Maka fungsi heuristik yang digunakan adalah sebagai berikut :

$$h_{diagonal(n)} = \min\left(abs(n.x - goal.x) + abs(n.y - goal.y)\right) \quad (2.2)$$

$$h_{orthogonal(n)} = (abs(n.x - goal.x) + abs(n.y - goal.y)) \quad (2.3)$$

$$h(n) = h_{diagonal(n)} + (h_{orthogonal(n)} - (2 * h_{diagonal(n)})) \quad (2.4)$$

Dimana h_*diagonal*(n) adalah banyaknya langkah diagonal yang bisa diambil untuk mencapai *goal* dari *node* n. H_orthogoanl adalah banyaknya langkah lurus yang bisa diambil untuk mencapai goal node n.

Nilai heuristik kemudian diperoleh dari *h_diagonal*(n) ditambah dengan selisih *h_orthogonal*(n) dengan dua kali *h_diagonal*(n). Dengan kata lain, jumlah langkah diagonal kali *cost* ditambah jumalh langkah lurus yang masih bisa diambil dikali *cost* pergerakan lurus.

2.6 Algoritma SMA* (Start)

*Simplified memory bounded A** merupakan pengembangan algoritma A* yang mengatasi masalah *storage problem* dengan meletakan *limit* tetap pada ukuran *open list*. Ketika sebuah *node* baru diproses, jika memiliki nilai yang lebih besar dari setiap *node* yang ada di *list* maka akan dibuang. Jika tidak maka akan ditambahkan, dan *node* yang sudah di dalam *list* dengan nilai terbesar dihapus.

Pendekatan ini bisa jauh lebih efeisien dibandingkan dengan pendekatan IDA* (*Iretative Deeping A**), walaupun ini masih bisa menyebabkan mengunjungi ulang *node* yang sama berulang kali selama proses pencarian. Hal ini sangat tergantung dengan nilai heuristik yang digunakan. Nilai heuristik yang tidak tepat bisa menyebabkan node penting tersingkirkan oleh *node* tidak berguna dari *open list*.

SMA* adalah contoh dari sebauh mekanisme pencarian "*lossy*". Dalam rangka untuk mengurangi konsumsi *memory*, hal ini membuang informasi dengan asumsi bahwa informasi yang dibuang itu tidak penting. Bagaimanapun tidak ada jaminan bahwa hal itu tidak penting. Dalam semua kasus dengan SMA*, jalur yang ditemukan tidak memiliki jaminan menjadi jalur yang optimal. Pada awal pencarian, node yang tidak menjanjikan bisa saja dibuang.

Menetapkan *limit* yang besar pada ukuran *open list* dapat membantu meringankan masalah ini, namun fungsi untuk mengurangi penggunaan memori menjadi terbuang. Pada kasus ekstrem yang lain, dengan memberi limit 1 simpul pada *open list*, ini bisa mempercepat sekaligus mengurangi penggunaan memori dalam pencarian, namun jalur yang ditemukan bisa tidak optimal[8].

SMA* memiliki beberapa sifat sebagai berikut :

1. F-*cost* digunakan untuk mendapatkan *node* dengan nilai terkecil diantara semua *node* yang berada di *open list*.
2. SMA* selalu memperluas *node* yang memiliki nilai terbaik pada keturunannya yang blm diperiksa.
3. SMA* dijamin mendapatkan solusi yang optimal, jika disediakan *max* setidaknya sama besar dengan jumlah node pada jalur solusi optimal.
4. SMA* berperilaku indetik dengan A*, bila *max* lebih besar dari jumlah node yang dihasilkan oleh A*.

Adapun deskripsi atau *pseudocode* dari algoritma SMA* bisa dilihat pada gambar 2.5

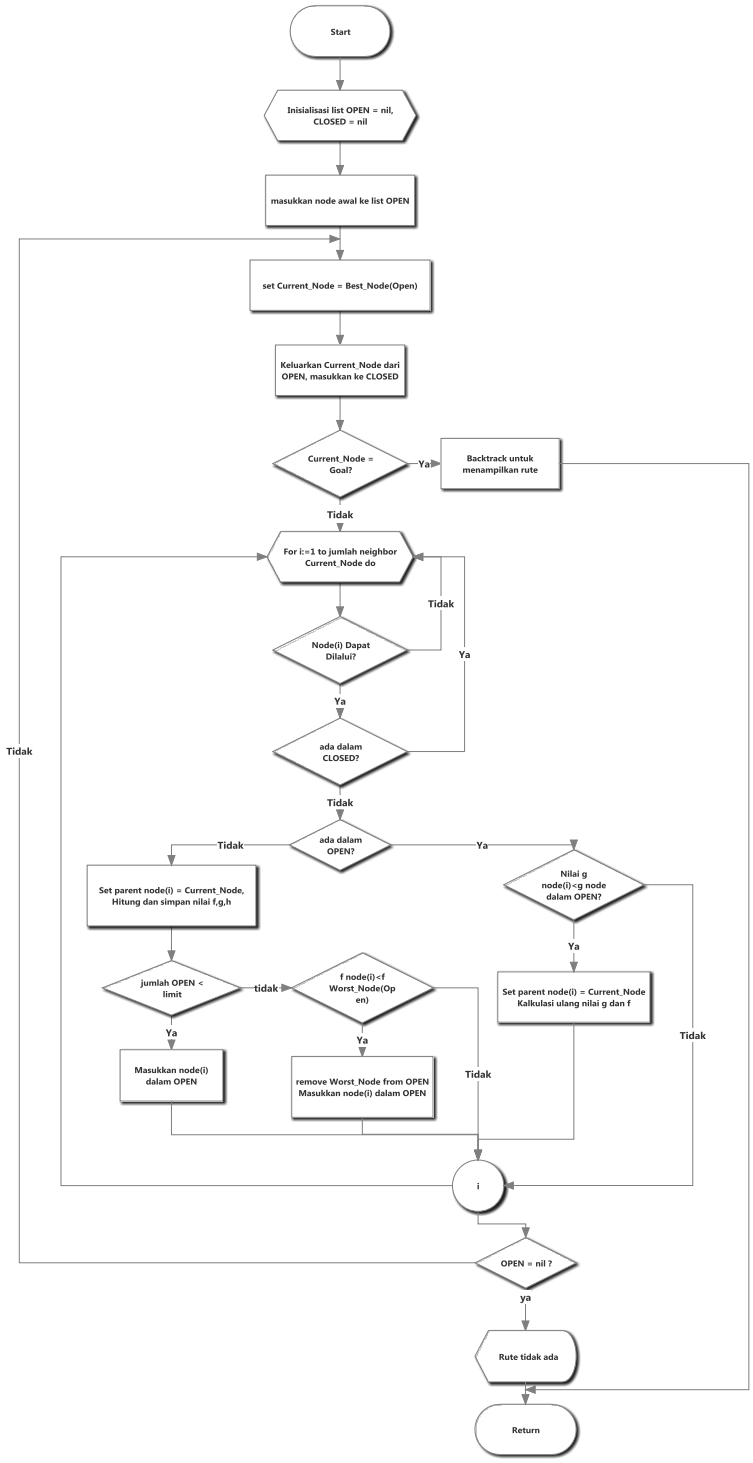
Pseudocode Algoritma SMA* (start)

```
put start on OPEN; USED – 1;
loop
        if empty(OPEN) return with failure;
        best ← deepest least f-cost leaf in OPEN;
        if goal(best) then return with success;
        succ ← next successor (best);
        f(succ) ← max(f(best),g(succ) + h(succ));
        if completed(best), BACKUP(best);
        if S(best) all in memory, remove best from OPEN
        USED ← USED +1;
        if USED > MAX then
                delete shallowest, highest f-cost node in OPEN;
                remove it from its parent's successor list;
                insert its parent on OPEN if necessary;
                USED ← USED -1;
        insert succ on OPEN.

Procedure BACKUP(n)
if n is completed and has parent then
        f(n)=least f cost of all successors;
        if f(n) changed, BACKUP(parent(n));
```

Gambar 2.5 *Pseudocode* Algoritma SMA*

SMA* dimulai dengan memasukan *node* awal ke dalam *open list*. *Variable global* max ditentukan untuk maksimum *node* yang bisa dimasukan ke dalam *open list*, dan variabel *uses* digunakan untuk melihat berapa banyak *node* yang sedang berada di dalam memori. Di dalam setiap *node* disimpan nilai g, h, dan f-*cost*, dan juga disimpan informasi mengenai *parent* dari *node* tersebut.



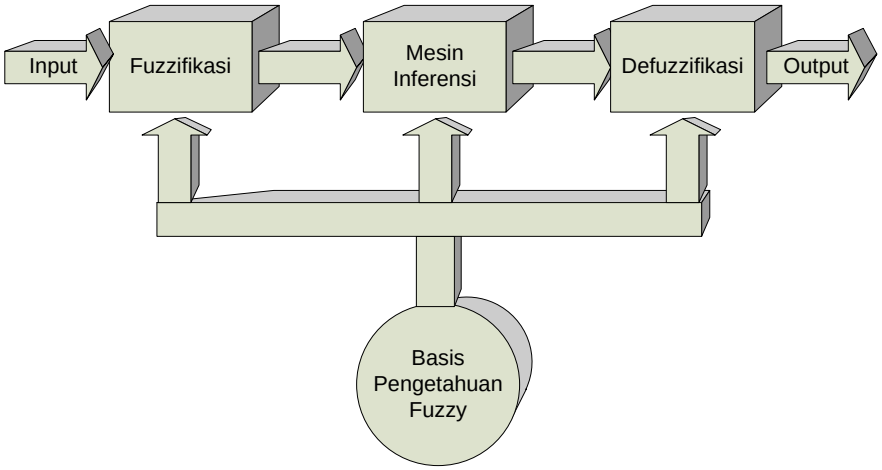
Gambar 2.6 *Flowchart* Algoritma SMA*

2.7 *Fuzzy Logic*

Konsep tentang logika *fuzzy* diperkenalkan oleh Prof. Lotfi Astor Zadeh pada 1962. Logika fuzzy adalah metodologi sistem kontrol pemecahan masalah, yang cocok diimplementasikan pada sistem, mulai dari sistem yang sederahana, sistem kecil, embedded system, jaringan PC, multi-channel atau workstation berbasis akurasi data, dan sistem kontrol. Metodologi ini dapat diterapkan pada perangkat keras, perangkat lunak, atau kombinasi keduanya. Dalam logika klasik dinyatakan bahwa segala sesuatu bersifat biner, yang artinya adalah hanya mempunyai dua kemungkinan, "Ya atau Tidak", "Benar atau Salah", "Baik atau Buruk" dan lain-lain. Oleh karena itu, semua ini dapat mempunyai nilai keanggotaan 0 dan 1. Akan tetapi, dalam logika fuzzy memungkinkan nilai kenaggotaan berada diantara 0 dan 1. Artinya, bisa saja suatu keadaan mempunyai dua nilai "Ya dan Tidak", "Benar dan Salah", "Baik dan Buruk" secara bersamaan, namun besar nilainya tergantung pada bobot keanggotaan yang dimilikinya. Logika *fuzzy* dapat digunakan diberbagai bidang, seperti pada sistem diagnosis penyakit (dalam bidang kedokteran), pemodelan sistem pemasaran, riset operasi (dalam bidang ekonomi), kendali kualitas air, prediksi adanya gempa bumi, klasifikasi dan pencocokan pola (dalam bidang teknik)[9].

Konsep himpunan *fuzzy* memiliki dua atrubut, yaitu :

1. Linguistik, yaitu nama suatu kelompok yang mewakili suatu keadaan tertentu dengan menggunakan bahasa alami, misalnya dingin, sejuk, panas mewakili variable temperatur.
2. Numeris, yaitu suatu nilai yang menunjukan ukuran dari suatu variabel, misalnya 10, 35, 40 dan sebagainya.



Gambar 2.7 Struktur sisten inferensi fuzzy.

Keterangan :

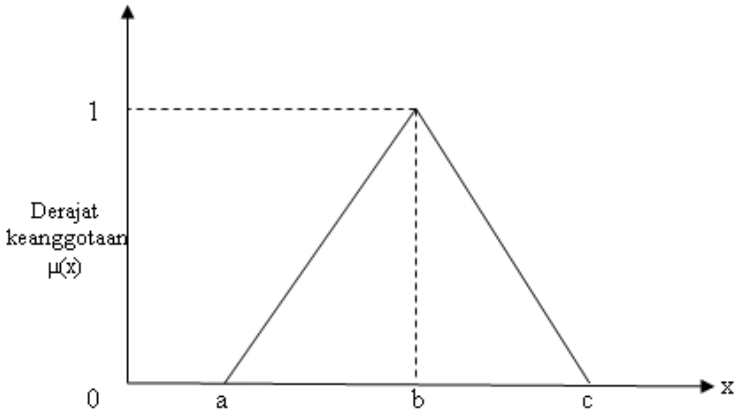
1. Basis Pengetahuan *Fuzzy* merupakan kumpulan *rule-rule fuzzy* dalam bentuk pernyataan IF…THEN.
2. *Fuzzyfikasi* adalah proses untuk mengubah input sistem yang mempunyai nilai tegas menjadi variabel linguistic menggunakan fungsi keanggotaan yang disimpan dalam basis pengetahuan *fuzzy*.
3. Mesin Inferensi merupakan proses untuk mengubah input *fuzzy* dengan cara mengikuti aturan-aturan (*IF-THEN Rules*) yang telah ditetapkan pada basis pengetahuan *fuzzy*.
4. *DeFuzzyfikasi* merupakan proses mengubah output *fuzzy* yang diperoleh dari mesin inferensi menjadi nilai tegas menggunakan fungsi keanggotaan yang sesuai saat dilakukan *fuzzyfikasi*.

2.7.1 Fungsi Keanggotaan

Fuzzy system merupakan fungsi keanggotaan yang memainkan peran yang sangat penting untuk merepresentasikan masalah dan menghasilkan keputusan yang akurat. Terdapat banyak sekali fungsi keanggotaan yang dapat digunakan di antaranya adalah [9] :

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 2.8



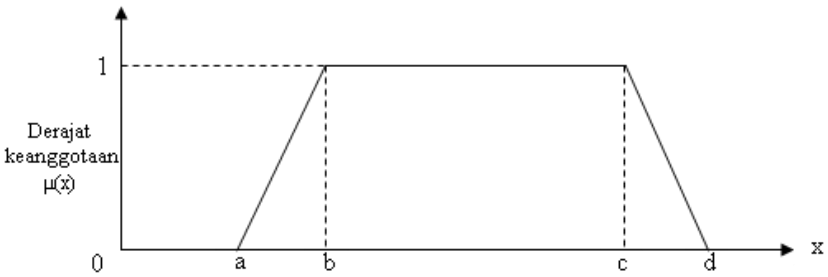
Gambar 2.8 Grafik Fungsi Segitiga

Fungsi Keanggotaan :

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

2. Fungsi Trapesium

   Berbeda dengan fungsi segitiga, pada fungsi trapesium ini terdapat beberapa nilai x yang memiliki derajat keanggotaan sama dengan 1, yaitu ketika . Tetapi derajat keanggotaan untukmemiliki karakteristik yang sama dengan fungsi segitiga. Grafik dan notasi matematika dari fungsi trapesium dapat di lihat pada gambar 2.9.



Gambar 2.9 Grafik Fungsi Trapesium

Fungsi Keanggotaan :

$$\mu(x) = \begin{cases} 0; & x \le a \text{ atau } x \ge d \\ (x-a)/(b-a); & a \le x \le b \\ 1; & b \le x \le c \\ (d-x)/(d-c); & c < x < d \end{cases}$$

2.7.2 Metode Sugeno

Fuzzy metode sugeno merupakan metode inferensi fuzzy untuk aturan yang direpresentasikan dalam bentuk pernyataan IF – THEN, dimana output (konsekuen) sistem tidak berupa himpunan *fuzzy* melainkan berupa konstanta atau persamaan linier. Metode ini diperkenalkan oleh Takagi Sugeno pada tahun 1985.model sugeno menggunakan fungsi kenanggotaan 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 metode Mamdani, hanyasaja output (konsekuen) sistem tidak berupa himpunan *fuzzy*, melainkan berupa konstanta atau persamaan linier. Michio sugeno mengusulkan penggunaan singleton [9].

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 dimana 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 akan digunakan dalam fungsi implikasi adalah sebagai berikut :

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

Dengan x dan y adalah skalar, 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 operator fuzzy seperti,

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

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*).

$$z^* = \frac{\Sigma \mu_i \, . \, z_i}{\Sigma \mu_i}$$

Dengan z* = titik pusat nilai crisp

μ_i = derajat kanggotaan masing-masing variabel

z_i = nilai domain pada variabel lingusitik z.

2.8 *Link* List

Dikembangkan tahun 1955-1956 oleh Allen Newell, Cliff Shaw dan Herbert Simon di RAND *corporation* sebagai struktur data utama untuk bahasa *information processing Language* (IPL). IPL dibuat untuk mengembangkan program *artificial intelligence*, seperti pembuatan Chess Solver. Victor Ynhve di *Massachusetts Institute of Technology* (MIT) jugs menggunakan *linked list* pada *natural language processing* dan *machine transittions* pada bahasa pemrograman COMMIT.

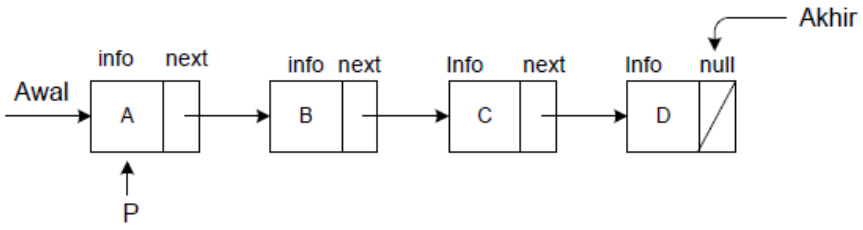
Linked list adalah salah satu bentuk struktur data, berisi kumpulan data (*node*) yang tersusun secara sekuensial, saling sambung menyambung, dinamis dan tidak terbatas. Pada tabel 2.1 dapat disimpulkan *linked list* lebih optimal daripada *array. Linked list* sering disebut juga senarai berantai dan saling terhubung dengan bantuan *variabel pointer*. Masing-masing data dalam *linked list* disebut dengan *node* (simpul) yang menempati alokasi memori secara dinamis dan biasanya berupa *struct* yang terdiri dari beberapa *field*[10].

Table 2.1 Perbandingan array dan link list

Array	*Linked List*
Statis	Dinamis
Penambahan/penghapusan data terbatas	Penambahan/penghapusan data tidak terbatas
Random access	*Random access*
Penghapusan *array* tidak mungkin	Penghapusan *linked list* mudah

Linked list terdiri dari tiga jenis yaitu *single linked list*, *doubel linked list*, dan *circular linked list*. Salah satu yang akan dijelaskan yaitu *single linked listI*.

Single linked list adalah *linked list* dengan simpul berisi satu *link*/*pointer* yang mengacu ke simpul berikutnya, ilustrasi dijelaskan pada gambar 2.8



Gambar 2.10 Ilustrasi *Single link list*.

Setiap *node* pada *linked list* mempunyai *field* yang berisi *pointer* ke *node* berikutnya, dan juga memiliki *field* yang berisi data. Pada akhir *linked list*, *node* terakhir akan menunjuk ke *NULL* yang akan digunakan sebagai kondisi berhenti pada saat pembacaan isi *linked list*.

Penambahan *node* baru memiliki tiga cara penyisipan yaitu sisip didepan, sisip ditengah dan sisip si akhir. Pada pembahasan ini akan lebih dijelaskan tentang sisip didepan, penambahan dara didepan akan dikaitkan di *node* paling awal, namun pada saat pertama kali (data masih kosong), maka penambahan data dilakukan dengan cara "Awal" ditunjukan ke node baru tersebut[10].

2.9 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[11].

2.9.1 Istilah-istilah OOP

Istilah OOP sendiri sudah sering kali didengar di dalam ilmu bahasa pemrograman. Berikut ini istilah-istilah yang sering digunakan didalam 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[11].

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[11].

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. Abstaksi

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.

2.9.2 Konsep-konsep OOP

Konsep dasar dari pemrograman berorientasi objek lebih menekankan pada konsep sebagai berikut[11] :

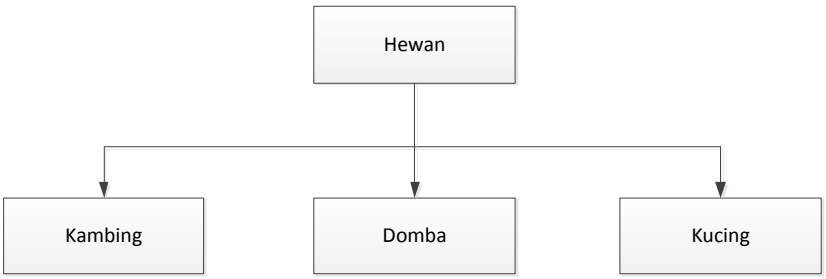
1. Kelas Abstaksi (*Class Abstraction*)
    - Kelas merupakan deskripsi abstrak informasi dan tingkah laku dari sekumpulan data.
    - Kelas dapat di ilustrasikan sebagai sesuatu cetak biru (*blueprint*) atau prototipe yang digunakan untuk menciptakan objek.

- Kelas merupakan tipe data bagi objek yang mengenkapsulasi data dan operasi pada data dalam suatu unit tunggal.
- Kelas mendefinisikan suatu struktur yang terdiri atas data kelas (*data field*), prosedur atau fungsi (*method*), dan sifat kelas (property).

2. Enkapsulasi (*Encapsulation*)
    - Istilah enkapsulasi sebenarnya adalah kombinasi data dan fungsionalitas dalam sebuah unit tunggal sebagai bentuk untuk menyembunyikan detail informasi.
    - Proses enkapsulasi memudahkan kita untuk menggunakan sebuah objek dari suatu kelas karena kita tidak perlu mengetahui segala hal secara rinci.
    - Enkapsulasi menekankan pada antarmuka suatu kelas, atau dengan kata lain bagaimana menggunakan objek kelas tertentu.
    - Contoh : kelas mobil menyediakan antarmuka fungsi untuk menjalankan mobil tersebut, tanpa kita perlu tahu komposisi bahan bakar, udara dan kalir yang diperlukan untuk proses tersebut.
3. Pewarisan (*Inheritance*)
    - Kita dapat mendefinisikan suatu kelas baru dengan mewarisi dari kelas lain yang sudah ada.
    - Penurunan sifat ini bisa dilakukan secara bertingkat-tingkat, sehingga semakin ke bawah kelas tersebut semakin spesifik.
    - Sub kelas memungkinkan kita untuk melakukan spesifikasi detail dan perilaku khusu dari kelas supernya.

    - Dengan konsep pewarisan, seorang *programmer* dapat menggunakan kode yang telah ditulisnya pada kelas *super* berulang kali pada kelas turunananya tanpa harus menulis ulang semua kode.

Contoh pewarisan :



Gambar 2.11 Contoh Pewarisan

Keuntungan dari pewarisan :

a. *Subclass* menyediakan *state/behaviour* yang spesifik yang membedakan *supercalss*, 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* (abstaksi), untuk mendefinisikan *class* dengan tingkah laku dan *state* secara umum.

4. *Polimorfisme*

- Polimorfisme merupakan kemampuan objek-objek yang berbeda kelas namun terkait dalam pewarisan untuk merespon secara berbeda terhadap suatu pesan yang sama.
- Polimorfisme juga dapat dikatakan kemampuan sebuah objek untuk memutuskan method mana yang akan diterapkan padanya, tergantung letak objek tersebut pada jenjang pewarisan.

2.10 UML (*Unified Modeling Language*)

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 mendefinisikan diagram-diagram sebagai berikut[12] :

1. *Use Case Diagram.*
2. *Class Diagram.*
3. *Activity diagram.*
4. *Sequence Diagram.*

2.10.1 *Use Case Diagram*

Use case adalah deskrispsi fungsi dari sebuah system dari persprektif pengguna. Use case bekerja dengan mendeskripsikan tipikal interaksi antara user (pengguna) dengan *system* sendiri melalui sebuah cerita bagaimana sebuah *system* digunakan. Urutan langkah-langkah yang menerangkan antara antara user (pengguna) dan system disebut scenario. Setiap *scenario* mendekskipsikan urutan kejadian, setiap urutan di inisialisasi oleh orang, system yang lain. Perangkat keras atau urutan waktu. Dengan demikian secara singkat bisa dikatakan *use case* adalah serangkaian *scenario* yang digabungkan bersama-sama oleh tujuan umum pengguna[12].

Use case diagram dapat sangat membantu menyusun *requirement* sebuah sistem, mengkomunikasikan rancangan dengan klien, 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-*include* 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 *behaviour*-nya sendiri. Sementara hubungan generalisasi antar *use case* menunjukkan bahwa *use case* yang satu merupakan spesialisasi dari yang lain[12].

2.10.2 *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[12].

2.10.3 *Activity Diagram*

Activity diagram menggambarkan berbagai alur aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alur berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi.

Activity diagram merupakan *state diagram* khusus, dimana 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 jalur-jalur aktivitas dari level atas secara umum[12].

2.10.4 *Sequence Diagram*

Sequence diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pemain, *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[12].

2.11 *Unity3D*

Unity adalah sebuah *tool* yang terintegrasi untuk membuat *game*, arsitektur bangunan dan simulasi. Unity bisa untuk *game*s PC dan *games Online*. Untuk *games Online* kita memerlukan sebuah *plugin*, yaitu *Unity Web Player*, sama halnya dengan *Flash Player* pada *Browser*. Bahasa pemrograman yang digunakan bermacam-macam, mulai dari *Javascript*, C#, dan *Boo*.

Pada unity, kita tidak bisa melakukan desain / *modelling*, dikarenakan unity bukan *tool* untuk mendesain Jadi jika kita ingin mendesain, kita memerlukan 3D editor lain seperti *3dsmax* atau *Blender*. Banyak hal yang bisa di lakukan di unity , ada fitur *audio reverb zone , particle effect , Sky Box* Untuk menambahkan langit , dan masih banyak lagi tentunya .kita juga bisa langsung mengedit *texture* dari *editor* seperti *photoshop* dll , unity bagus untuk pemula maupun *expert*[13].

Features (Scripting)

1. Mendukung 3 bahasa pemrograman, *JavaScript*, *C#*, dan *Boo*.
2. *Flexible* adn *EasyMoving*, *rotating*, dan *Scaling Object* hanya perlu sebarus kode. Begitu juga dengan *Duplicating*, *removing*, dan *changing properties*.
3. *Multiplatform game* bisa di *deploy* di PC, Mac, Wii, iphone, iPad dan browser, android.
4. *Visual properties variables* yang di definisikan dengan script ditampilkan pada Editor. Bisa digeser, di *drag* and *drop*, bisa memilih warna dengan color picker.
5. Berbasis .NET→ penjalanan program dilakukan dengan *Open Source* .NET *platform*, Mono.