BAB II
TINJAUAN PUSTAKA

II.1 Multimedia

Multimedia adalah media yang menggabungkan dua unsur atau lebih media yang terdiri dari teks, grafik, gambar, audio, video dan animasi secara terintegrasi [22].

Menurut beberapa ahli multimedia memiliki beberapa definisi yaitu :

1. Rosch

   Multimedia adalah kombinasi dari komputer dan video.

2. McComick

   Multimedia adalah kombinasi dari komputer dari tiga elemen yaitu suara, gambar dan teks.

3. Turban dan kawan-kawan

   Multimedia adalah kombinasi dari paling sedikit dua media *input* dan *output*. Media ini dapat berupa audio, animasi, video, teks, grafik dan gambar.

4. Robin dan Linda

   Multimedia adalah alat yang dapat menciptakan presentasi yang dinamis dan interaktif yang mengkombinasikan teks, grafik, animasi, audio dan video.

5. Hofstetter

   Multimedia adalah pemanfaatan komputer untuk membuat dan menggabungkan teks, grafik, audio, video dengan menggunakan *tool* yang memungkinkan pemakai berinteraksi, berkreasi dan berkomunikasi. Dalam definisi ini terdapat empat komponen multimedia yaitu :

   a. Harus ada peralatan elektronik yang dapat mengkoordinasikan apa yang dapat dilihat dan didengar dan yang dapat berinteraksi dengan pengguna.

b. Harus ada *link* yang dapat menghubungkan pengguna dengan informasi.
c. Harus ada alat navigasi yang dapat memandu pengguna untuk menjelajahi jaringan informasi yang saling terhubung.
d. Harus dapat menyediakan tempat kepada pengguna untuk mengumpulkan, memproses dan mengkomunikasikan informasi dan ide dari pengguna itu sendiri.

Dengan penggunaan multimedia, penyampaian informasi akan menjadi lebih menarik dan mempermudah pengguna dalam mendapatkan informasi tersebut. Seperti yang disebutkan dalam laporan hasil penelitian yang dikeluarkan oleh *Computer Technology Research* (*Hofstetter*) bahwa seseorang hanya akan mendapatkan 20% dari apa yang mereka lihat dan 30% dari yang mereka dengar. Sedangkan melalui multimedia akan mendapatkan 50% dari apa yang mereka lihat dan dengar, sampai 80% dari apa yang mereka lihat, dengar dan berinteraksi pada waktu yang bersamaan.

Keuntungan dari penggunaan multimedia dalam suatu aplikasi adalah:

1. Meningkatkan aliran gagasan dan informasi.
2. Merupakan cara yang kaya untuk mengkomunikasikan sesuatu.
3. Mendorong partisipasi, keterlibatan, dan eksplorasi pengguna.
4. Menstimulasi panca indera.
5. Memberikan kemudahan pemakaian terutama bagi pengguna awam.

II.2 *Game*

Game merupakan sebuah bentuk seni dimana penggunanya 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.

II.2.1 Pengertian *Game*

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 2 jenis yaitu *game* 2 dimensi (2D) dan 3 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.

II.2.2 *Game* 3D

Menurut Andrew Rollings dan Dave Morris, industri *game* selalu berusaha untuk mengikuti perkembangan teknologi yang ada di dunia ini. Ketika perangkat-perangkat komputer seperti *processor, graphic card* versi baru mulai muncul di pasaran, para *developer* game selalu berusaha mengikuti perkembangan tersebut.

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.

II.2.3 Jenis-jenis *Game*

Berikut ini beberapa jenis *game* berdasarkan cara pembuatannya, cara pemasarannya dan mesin yang menjalankannya. 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 dimainkan secara *online* melalui LAN atau internet.

II.2.4 *Third Person Shooter*

Third Person Shooter atau disingkat TPS merupakan sub-*genre* dari *action game* dan dikategorikan sebagai *game play* dengan model pertarungan. *Third person shooter* 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.

Seorang penembak orang ketiga adalah permainan terstruktur sekitar penembakan, dan di mana pemain dapat melihat avatar di layar dalam pandangan orang ketiga. *Genre* ini menarik untuk dimainkan karena *gameplay*nya yang *simple* membuat orang-orang mudah untuk memainkannya. Beberapa contoh *game* dengan *genre third person shooter* adalah *Army Men 3D, Tomb Raider, Resident Evil,* dan *The Punisher.*

II.3 Kecerdasan Buatan

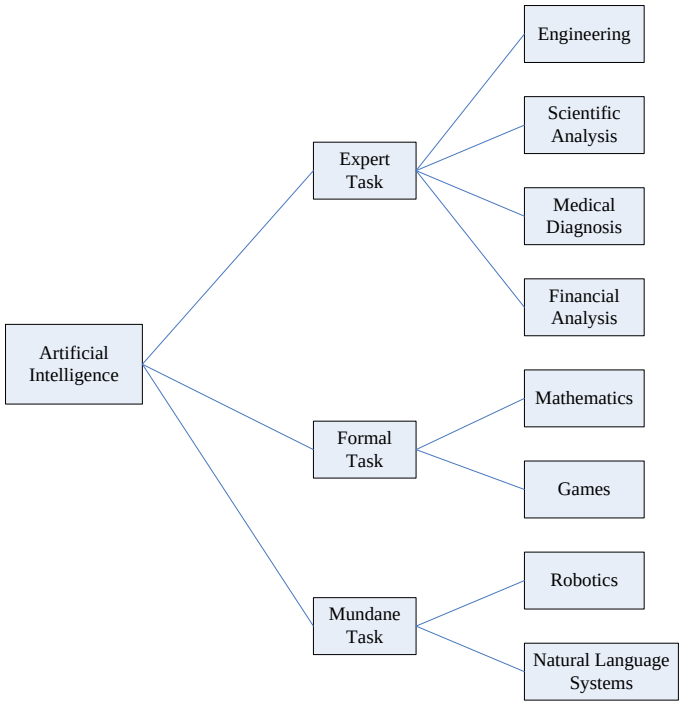
Kecerdasan buatan atau *Artificial Intelligence* (AI) merupakan cabang dari ilmu komputer yang berhubungan dengan pengotomatisan tingkah laku cerdas. Kecerdasan buatan didasarkan pada teori suara (*sound theoretical*) dan prinsip-prinsip aplikasi dari bidangnya. Prinsip-prinsip ini meliputi struktur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

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



Gambar II.1 Bidang-bidang tugas (*task domains*) dari AI [3]

Aplikasi penggunaan AI dapat dibagi ke dalam tiga kelompok yaitu :

1. *Expert Task*

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

2. *Formal Task*

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

3. *Mundane Task*

   Secara harfiah *mundane* adalah keduniaan. AI digunakan untuk melalukan 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 berdasarikan pengalaman.

II.4 Algoritma Pencarian

Kecerdasan buatan diciptakan dengan tujuan untuk memecahkan berbagai masalah yang dihadapi oleh manusia. Masalah di dalam kecerdasan buatan adalah masalah-masalah yang dapat di konversi ke dalam ruang keadaan, mempunyai keadaan awal (*initial state*), dan keadaan tujuan (*goal state*), serta dapat dibuat aturan-aturan untuk mengubah suatu keadaan (*state*) ke keadaan (*state*) lainnya. Algoritma pencarian merupakan teknik pemecahan masalah dan merupakan bagian dari kecerdasan buatan

Metode pencarian sangat penting untuk menyelesaikan permasalahan karena setiap *state* atau keadaan menggambarkan langkah-langkah untuk menyelesaikan permasalahan. Dalam sebuah permainan, metode pencarian akan menentukan apa yang harus dilakukan dimana setiap *state* menggambarkan kemungkinan posisi pada

suatu saat. Metode pencarian adalah bagian dari kesimpulan dimana setiap *state* menggambarkan hipotesis dalam sebuah rangkaian deduktif.

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

II.4.1 Pencarian Buta

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*.

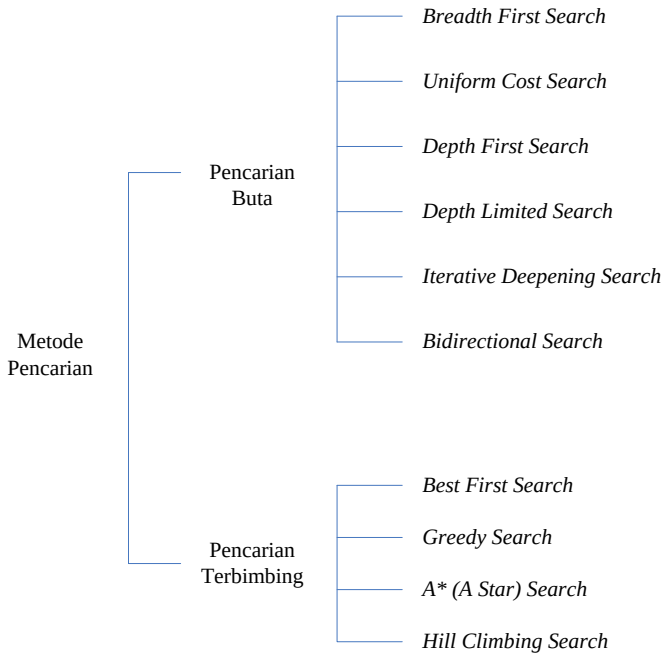
II.4.2 Pencarian Terbimbing

Pencarian terbimbing (*heuristic search)* berasal dari kata kerja bahasa Yunani, *heuriskein*, yang berarti ‘mencari’ atau ‘menemukan’. Dalam dunia pemrograman, sebagian orang menggunakan kata heuristik sebagai lawan kata dari algoritmik, dimana kata heuristik ini diartikan sebagai ‘suatu proses yang mungkin dapat menyelesaikan suatu masalah tetapi tidak ada jaminan bahwa solusi yang dicari selalu dapat ditemukan’[3]. Didalam mempelajari metode-metode pencarian ini, kata heuristik diartikan sebagai suatu fungsi yang memberikan suatu nilai berupa biaya perkiraan (estimasi) dari suatu solusi.

Pencarian terbimbing 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

heuristik. Fungsi heuristik berbeda dari pada algoritma dimana heuristik lebih merupakan perkiraan untuk membantu algoritma dan tidak harus valid setiap waktu. Beberapa contoh algoritma pencarian yang mengguanakan metode pencarian terbimbing atau *heuristic search* adalah *Best First Search, Greedy Search*, A* (A Star) dan *Hill Climbing Search*.

Gambar bagan metode pencarian dibagi menjadi dua golongan, yaitu:



Gambar II.2 Bagan Metode Pencarian [3]

II.5 Algoritma A* (A *Star*)

Algoritma A* adalah algoritma yang dikemukakan oleh *Hart, Nilsson*, dan *Raphael* pada tahun 1968. Algoritma A* merupakan salah satu algoritma *Branch & Bound* atau disebut juga sebagai sebuah algoritma untuk melakukan pencarian solusi dengan menggunakan informasi tambahan (heuristik) dalam menghasikan solusi yang optimal.

Algoritma A* (A Star) merupakan perbaikan dari metode *Best First Search* (BFS) dengan memodifikasi fungsi *heuristic* nya. A* (A Star) akan meminimumkan total biaya lintasan. Pada kondisi yang tepat, A* akan memberikan solusi yang terbaik dalam waktu yang optimal. Pencarian menggunakan algoritma A* mempunyai prinsip yang sama dengan algoritma BFS hanya saja dengan dua faktor tambahan yaitu :

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

Cost untuk setiap simpul tidak harus berupa jarak. *Cost* bisa saja berupa waktu bila ingin mencari jalan dengan waktu tercepat untuk dilalui. Algoritma A* bekerja dengan prinsip-prinsip sebagai berikut:

1. Simpul-simpul di *list* terbuka diurutkan oleh *cost* keseluruhan dari simpul awal ke simpul tujuan, dari *cost* terkecil sampai *cost* terbesar. *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 dimasukkan ke *list* terbuka bila jalan terpendek menuju simpul tersebut ditemukan.

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

$$F(n) = G(n) + H(n) \qquad \text{(II.1)}$$

keterangan :

F(n) adalah jumlah nilai dari fungsi G(n) dan H(n). Dengan nilai terkecil F(n) adalah jalur terpendek menuju tujuan akhir.

G(n) adalah nilai pada pergerakan simpul awal menuju simpul berikutnya.

H(n) adalah perkiraan nilai pergerakan simpul awal menuju tujuan akhir simpul atau fungsi heuristik.

Node dengan nilai terendah merupakan solusi terbaik untuk diperiksa pertama kali pada persamaan II.1 diatas. 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 *admissible heuristic* adalah jarak dengan menarik garis lurus karena jarak terdekat dari dua titik adalah dengan menarik garis lurus.

Beberapa terminologi dasar yang terdapat pada algoritma ini adalah *starting point*, *current node*, simpul, *neighbor node*, *open set*, *closed set*, *came from*, 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*.

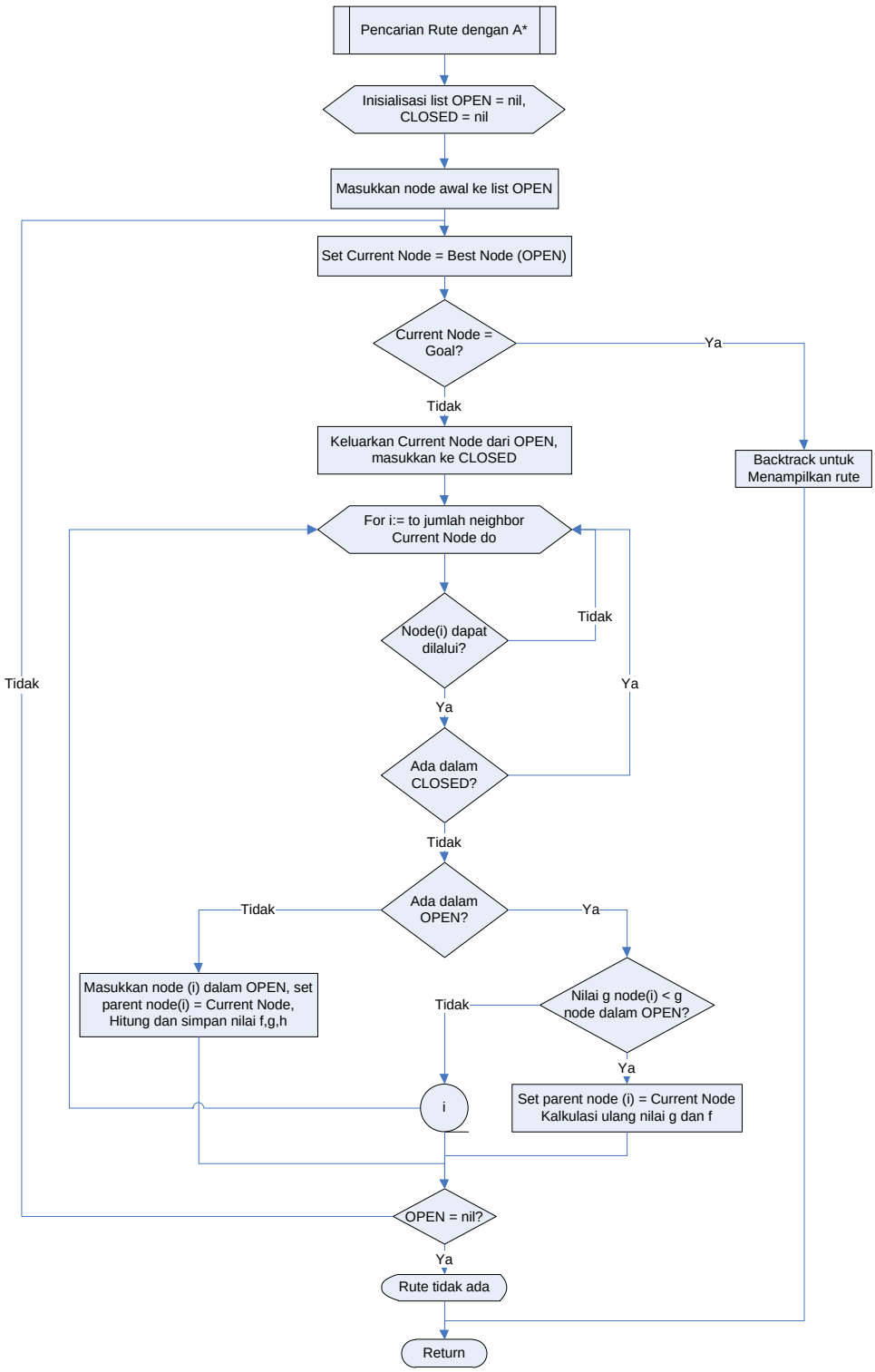
Algoritma A* secara ringkas langkah demi langkahnya adalah sebagai berikut:

1. Tambahkan simpul awal ke dalam *open list*.
2. Ulangi langkah berikut sampai pencarian berakhir:
   a. Carilah simpul n dengan biaya F(n) paling rendah, dalam *open* list. Simpul dengan biaya F terendah kemudian disebut *current node*.
   b. Keluarkan *current node* dari *open list* dan masukkan ke dalam *closed list*.
   c. Untuk setiap 8 simpul (*neighbor node*) dari *current node* lakukan langkah berikut:
      1). Jika sudah terdapat dalam *closed list* atau tidak *walkable*, maka abaikan, jika tidak lanjutkan.
      2). 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.
      3). 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. Berhenti ketika:
    1. Menambahkan *target point* ke dalam *closed list*, dalam hal ini jalan telah ditemukan, atau,
    2. Gagal untuk menemukan *target point*, dan *open list* kosong. Dalam kasus ini, tidak ada jalan.
e. Walaupun telah 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*.
f. 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 jalan. Bekerja mundur dari *target point*, pergi dari masing-masing simpul ke simpul *parent* sampai mencapai *starting point*.

Secara umum dapat digambarkan dengan *flowchart* sebagai berikut:



Gambar II.3 *Flowchart* Algoritma A* [7]

II.6 *Fuzzy Logic*

Konsep tentang *fuzzy logic* atau 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 sederhana, 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.

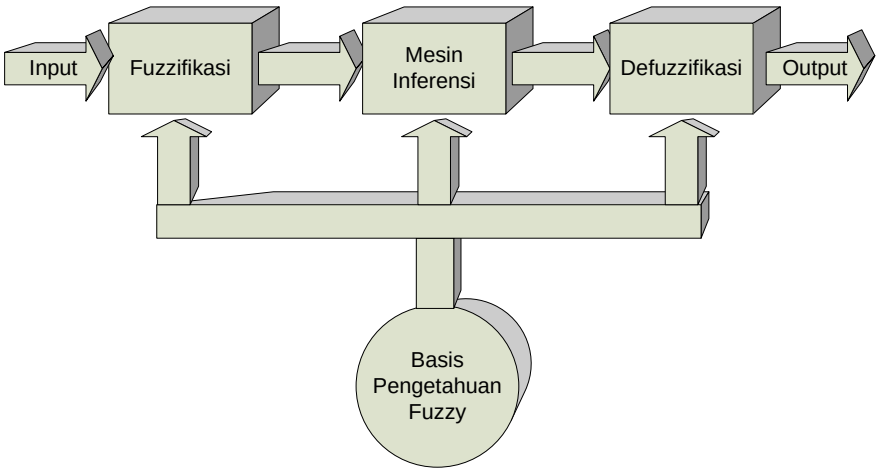
Dalam logika klasik dinyatakan bahwa segala sesuatu bersifat biner, yang artinya adalah hanya mempunyai dua kemungkinan, "Ya atau Tidak", "Benar atau Salah", "Baik atau Buruk" dan lain-lain. Oleh karena itu, semua ini dapat mempunyai nilai keanggotaan 0 dan 1. Akan tetapi, dalam logika *fuzzy* memungkinakan 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).

Konsep himpunan *fuzzy* memiliki 2 atribut, yaitu :

1. Nilai linguistik atau *terma*, 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.

Cara kerja logika fuzzy dapat digambarkan sebagai struktur elemen dasar sistem inferensi *fuzzy* sebagai berikut:



Gambar II.4 Struktur Sistem 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 dengan saat dilakukan *fuzzyfikasi*.

II.6.1 Himpunan *Fuzzy*

Logika fuzzy dikembangkan dari teori himpunan fuzzy. Himpunan klasik yang sudah dipelajari selama ini disebut juga dengan himpunan tegas (*crisp set*). Di dalam himpunan tegas, keanggotaan suatu unsur di dalam himpunan dinyatakan secara tegas, apakah objek tersebut anggota himpunan atau bukan.

Pada himpunan tegas (*crisp*), nilai keanggotaan suatu *item* x dalam suatu himpunan A, yang sering ditulis dengan $\mu_A[x]$, memiliki dua kemungkinan, yaitu :

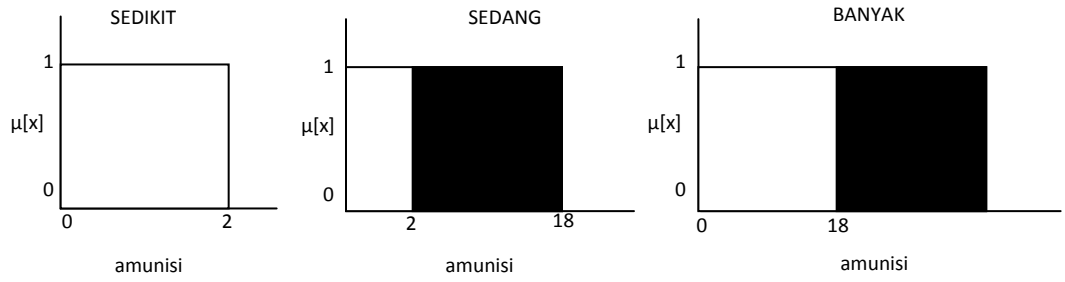
1. Satu (1), yang berarti bahwa suatu *item* menjadi anggota dalam suatu himpunan.
2. Nol (0), yang berarti bahwa suatu *item* tidak menjadi anggota dalam suatu himpunan.

Contoh :

Misalkan variabel amunisi di dalam NPC (*Non Playable Character*) dibagi menjadi tiga kategori, yaitu :

SEDIKIT amunisi < 2
SEDANG $2 \leq$ amunisi ≤ 18
BANYAK amunisi > 18

Nilai keanggotaan secara grafis, himpunan SEDIKIT, SEDANG dan BANYAK ini dapat dilihat pada gambar II.5 berikut.



Gambar II.5 Himpunan SEDIKIT, SEDANG dan BANYAK

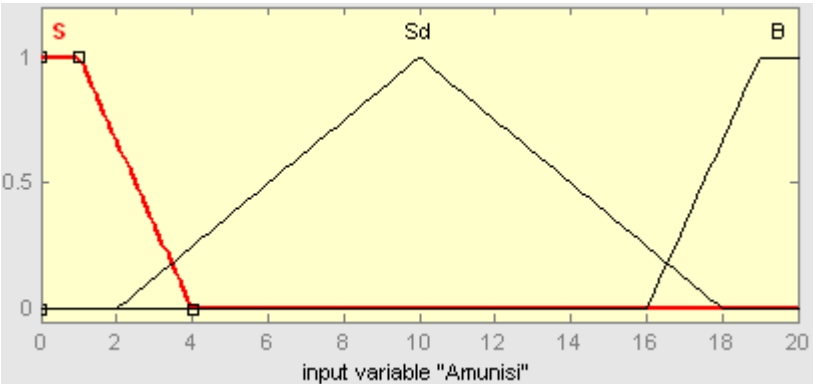
Pada gambar di atas, dapat dijelaskan bahwa :

a. Apabila NPC memiliki jumlah amunisi 1, maka dikatakan SEDIKIT ($\mu_{SEDIKIT}[1]=1$).
b. Apabila NPC memiliki jumlah amunisi 2, maka dikatakan TIDAK SEDIKIT ($\mu_{SEDIKIT}[2]=0$).
c. Apabila NPC memiliki jumlah amunisi 2, maka dikatakan SEDANG ($\mu_{SEDANG}[2]=1$).

d. Apabila NPC memiliki jumlah amunisi 1, maka dikatakan TIDAK SEDANG ($\mu_{SEDANG}[1]=0$).
e. Apabila NPC memiliki jumlah amunisi 18, maka dikatakan SEDANG ($\mu_{SEDANG}[18]=1$).

Dari sini bisa dikatakan bahwa pemakaian himpunan tegas (*crisp*) untuk menyatakan amunisi sangat tidak adil. Adanya perubahan kecil saja pada suatu nilai mengakibatkan perbedaan kategori yang cukup signifikan.

Himpunan *fuzzy* digunakan untuk mengantisipasi hal tersebut. Amunisi pada NPC dapat masuk dalam dua himpunan yang berbeda, SEDIKIT dan SEDANG, SEDANG dan BANYAK dan sebagainya, seberapa besar eksistensinya dalam himpunan tersebut dapat dilihat pada nilai keanggotannya. Gambar II.6 menunjukkan himpunan *fuzzy* untuk variabel amunisi.



Gambar II.6 Himpunan *Fuzzy* Untuk Variabel Amunisi

Pada gambar II.6 di atas dapat dilihat bahwa :

a. Apabila amunisi pada NPC berjumlah 3, termasuk dalam himpunan SEDIKIT ; namun juga termasuk dalam himpunan SEDANG.
b. Apabila amunisi pada NPC berjumlah 17, termasuk dalam himpunan SEDIKIT; namun juga termasuk dalam himpunan BANYAK.

Kalau pada himpunan tegas (*crisp*), nilai keanggotaan hanya ada dua kemungkinan yaitu 0 atau 1, maka pada himpunan *fuzzy* nilai keanggotaan terletak pada rentang 0 sampai 1. Apabila x memiliki nilai keanggotaan *fuzzy* $\mu_A[x]=0$ berarti x tidak menjadi anggota himpunan A, demikian pula apabila x memiliki nilai keanggotaan *fuzzy* $\mu_A[x]=1$ berarti x menjadi anggota penuh pada himpunan A.

Hal yang perlu diketahui dalam memahami sistem *fuzzy* adalah:

a. Variabel *fuzzy*

   Variabel *fuzzy* merupakan variabel yang hendak dibahas dalam suatu sistem *fuzzy*, contoh : umur, temperatur dll.

b. Himpunan *fuzzy*

   Himpunan *fuzzy* merupakan suatu grup yang mewakili suatu kondisi atau keadaan tertentu dalam suatu variabel *fuzzy*.

c. Semesta pembicaraan

   Semesta pembicaraan adalah keseluruhan nilai yang diperbolehkan untuk dioperasikan dalam suatu variabel *fuzzy*. Semesta pembicaraan merupakan himpunan bilangan *real* yang senantiasa naik (bertambah) secara monoton dari kiri ke kanan. Nilai semesta pembicaraan dapat berupa bilangan *positif* maupun *negatif*. Pada suatu kondisi tertentu nilai semesta pembicaraan ini tidak dibatasi batas atasnya.

d. Domain

   Domain himpunan *fuzzy* adalah keseluruhan nilai yang diijinkan dalam semesta pembicaraan dan boleh dioperasikan dalam suatu himpunan *fuzzy*. Seperti halnya semesta pembicaraan, domain merupakan himpunan bilangan *real* yang senantiasa naik (bertambah) secara monoton dari kiri ke kanan. Nilai domain dapat berupa bilangan *positif* maupun *negatif*.

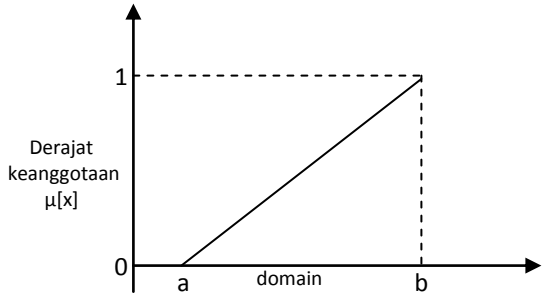
II.6.2 Fungsi Keanggotaan

Fungsi keanggotaan *(membership function)* adalah suatu kurva yang menunjukkan pemetaan titik-titik input data ke dalam nilai keanggotaanya (sering

juga disebut dengan derajat keanggotaan) yang memiliki interval antara 0 sampai 1. Salah satu cara yang dapat digunakan untuk mendapatkan nilai keanggotaan adalah dengan melalui pendekatan fungsi. Ada beberapa fungsi yang bisa digunakan.

1. Representasi Linear

   Pada representasi linear, pemetaan input ke derajat keanggotaannya digambarkan sebagai suatu garis lurus. Bentuk kurva ini paling sederhana dan menjadi pilihan yang baik untuk mendekati suatu konsep yang kurang jelas. Ada 2 keadaan himpunan fuzzy yang linear. Pertama, kenaikan himpunan dimulai pada nilai domain yang memiliki derajat keanggotaan nol [0] bergerak ke kanan menuju ke nilai domain yang memiliki derajat keanggotaan lebih tinggi (gambar II.7).

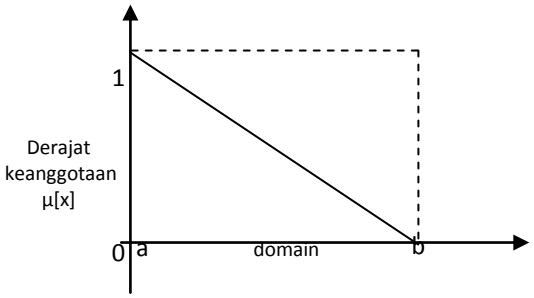


Gambar II.7 Representasi Linear Naik

Fungsi keanggotaan :

$$\mu[x] = \begin{cases} 0; & x < a \\ (x-a)/(b-a); & a \le x \le b \\ 1; & x > b \end{cases} \qquad \text{(II.2)}$$

Kedua, merupakan kebalikan dari yang pertama. Garis lurus dimulai dari nilai domain dengan derajat keanggotaan tertinggi pada sisi kiri, kemudian bergerak menurun ke nilai domain yang memiliki derajat keanggotaan lebih rendah (gambar II.8).



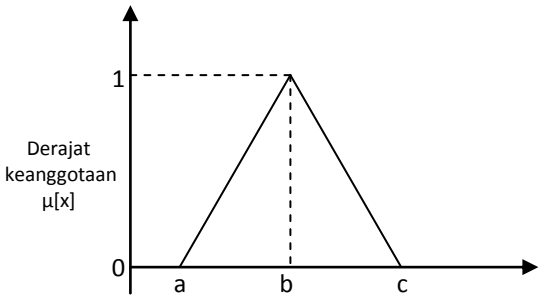
Gambar II.8 Representasi Linear Turun

Fungsi Keanggotaan :

$$\mu[x] = \begin{cases} (b-x)/(b-a); & a \le x < b \\ 0; & x \ge b \end{cases} \quad \text{(II.3)}$$

2. Representasi Kurva Segitiga

Kurva segitiga pada dasarnya merupakan gabungan antara 2 garis (*linear*) seperti terlihat pada gambar



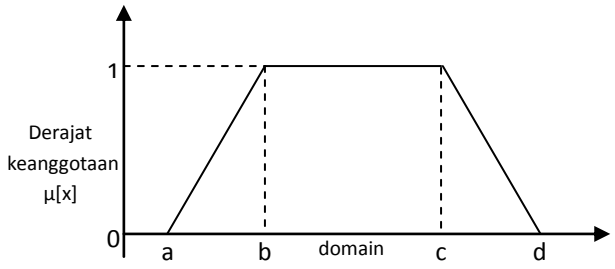
Gambar II.9 Kurva Segitiga

Fungsi keanggotaan :

$$\mu[x] = \begin{cases} 0; & x \le a \text{ atau } x \ge c \\ (x-a)/(b-a); & a < x \le b \\ (c-x)/(c-b); & b < x \le c \end{cases} \quad \text{(II.4)}$$

3. Representasi Kurva Trapesium

Kurva trapesium pada dasarnya seperti bentuk segitiga, hanya saja ada beberapa titik yang memiliki nilai keanggotaan 1.



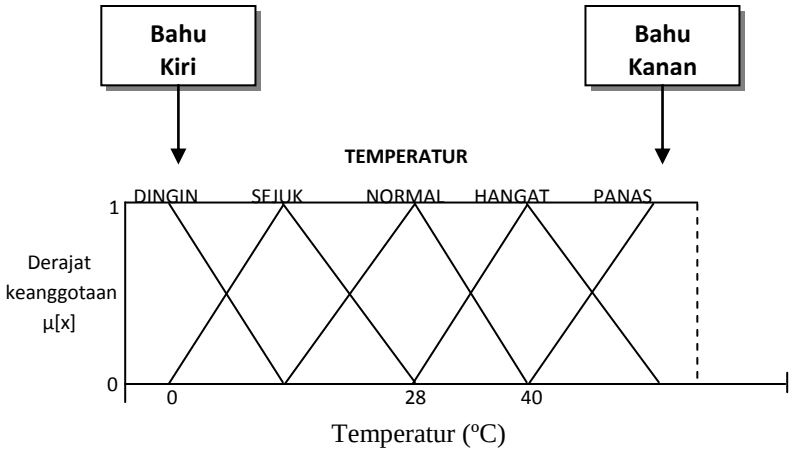
Gambar II.10 Kurva Trapesium

Fungsi Keanggotaan :

$$\mu[x] = \begin{cases} 0; & x \le a \text{ atau } x \ge d \\ (x-a)/(b-a); & a \le x \le b \\ 1; & b \le x \le c \\ (d-x)/(d-c); & c \le x \le d \end{cases} \quad \text{(II.5)}$$

4. Representasi Kurva Bentuk Bahu

   Daerah yang terletak di tengah-tengah suatu variabel yang direpresentasikan dalam bentuk segitiga, pada sisi kanan dan kirinya akan naik dan turun (misalkan: DINGIN bergerak ke SEJUK bergerak ke HANGAT dan bergerak ke PANAS). Tetapi terkadang salah satu sisi dari variabel tersebut tidak mengalami perubahan. Sebagai contoh, apabila telah mencapai kondisi PANAS, kenaikan temperatur akan tetap berada pada kondisi PANAS. Himpunan *fuzzy* 'bahu', bukan segitiga, digunakan untuk mengakhiri variabel suatu daerah *fuzzy*. Bahu kiri bergerak dari benar ke salah, demikian juga bahu kanan bergerak dari salah ke benar. Gambar dibawah menunjukkan variabel TEMPERATUR dengan daerah bahunya.



Gambar II.11 Daerah 'Bahu' Pada Daerah TEMPERATUR

II.6.3 Operator Dasar *Fuzzy*

Seperti halnya himpunan konvensional, ada beberapa operasi yang didefinisikan secara khusus untuk mengkombinasi dan memodifikasi himpunan *fuzzy*. Nilai keanggotaan sebagai hasil dari operasi dua himpunan sering dikenal dengan nama *fire strength* atau α-predikat. Ada tiga operator dasar yang diciptakan oleh Zadeh, yaitu:

1. Operator AND

   Operator ini berhubungan dengan operasi interseksi pada himpunan. α-predikat sebagai hasil operasi dengan operator AND diperoleh dengan mengambil nilai keanggotaan terkecil antar elemen pada himpunan-himpunan yang bersangkutan.

$$\mu_{A\cap B} = \min(\mu_A[x], \mu_B[y]) \tag{II.6}$$

2. Operator OR

   Operator ini berhubungan dengan operasi union pada himpunan. α-predikat sebagai hasil operasi dengan operator OR diperoleh dengan mengambil nilai keanggotaan terbesar antar elemen pada himpunan-himpunan yang bersangkutan.

$$\mu_{A\cup B} = \max(\mu_A[x], \mu_B[y]) \quad \text{(II.7)}$$

3. Operator NOT

   Operator ini berhubungan dengan operasi komplemen pada himpunan. α-predikat sebagai hasil operasi dengan operator NOT diperoleh dengan mengurangkan nilai keanggotaan elemen pada himpunan yang bersangkutan dari 1.

$$\mu_{A'} = 1 - \mu_A[x] \quad \text{(II.8)}$$

II.6.4 Penalaran Monoton

Metode penalaran secara monoton digunakan sebagai dasar untuk teknik implikasi *fuzzy*. Meskipun penalaran ini sudah jarang sekali digunakan namun terkadang masih digunakan untuk penskalaan *fuzzy*. Jika dua daerah *fuzzy* direlasikan dengan implikasi sederhana sebagai berikut :

IF x is A THEN y is B (II.9)

transfer fungsi :

y = *f*((x,A),B) (II.10)

maka sistem *fuzzy* dapat berjalan tanpa harus melalui komposisi dan dekomposisi *fuzzy*. Nilai output dapat diestimasi secara langsung dari nilai keanggotaan yang berhubungan dengan antesendennya.

II.6.5 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 :

IF x is A THEN y is B

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 sebagai konsekuen. Proposisi ini dapat diperluas dengan menggunakan operator *fuzzy*, seperti:

IF (x_1 is A_1) ○ (x_2 is A_2) ○(x_3 is A_3) ○ ..... ○ (x_N is A_N) THEN y is B
dengan ○ adalah operator (misal: OR atau AND). (II.11)

II.6.6 Metode Mamdani

Metode Mamdani paling sering digunakan dalam aplikasi-aplikasi karena strukturnya yang sederhana, yaitu menggunakan operasi MIN-MAX atau MAX-PRODUCT untuk mendapat output atau keluaran dari IF-THEN rule.

Jika terdapat lebih dari satu kaidah *fuzzy* yang dievaluasi, keluaran semua IF-THEN rule dikombinasikan menjadi sebuah *fuzzy* set tunggal. Jika dilakukan fungsi min pada implikasi dan max pada agregasi, maka metode Mamdani disebut juga metode MIN-MAX (*min-max inferencing*).

Misalkan terdapat *n* buah kaidah yang berbentuk:

IF x_1 is A_1^k and x_2 is A_2^k THEN y^k is B^k *k= 1, 2, ..., n* (II.12)

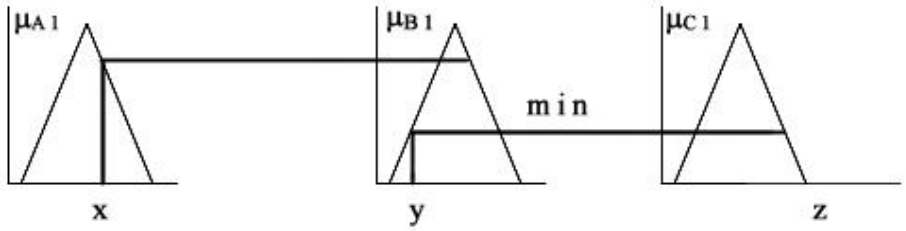
Yang dalam hal ini *n* menyatakan kaidah atau *rules* yang sudah dibangun, dan *k* menyatakan jumlah kaidah dari 1 sampai ke *n.* A_1^k dan A_2^k adalah himpunan *fuzzy* yang merepresentasikan pasangan antesenden ke-*k*, dan B^k adalah himpunan *fuzzy* yang menyatakan konsekuen ke-*k*.

Berdasarkan metode implikasi Mamdani, maka keluaran untuk *n* buah kaidah diberikan oleh:

$$\mu_B(y) = \max_{\substack{k \\ k = 1, 2, \ldots, n}} [\min[\mu_{A_1^k}(input(i)), \mu_{A_2^k}(input(j))]] \quad \text{(II.13)}$$

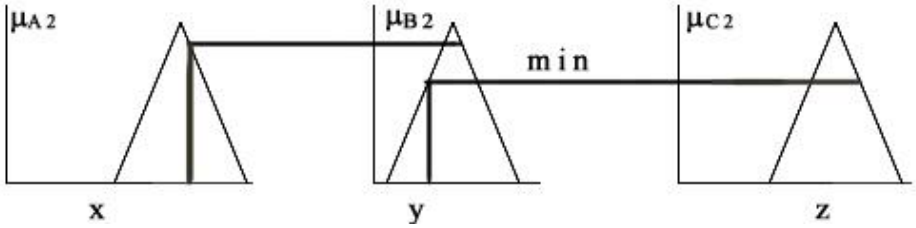
Metode MIN-MAX diilustrasikan dengan contoh sebagai berikut. Misalkan antesenden terdiri dari dua buah proposisi dengan penghubung “AND”.

Kaidah pertama: IF x is A1 AND y is B1 THEN z is C1 (II.14)



Gambar II.12 Komposisi Aturan *Fuzzy* Metode Mamdani: Kaidah Pertama [11]

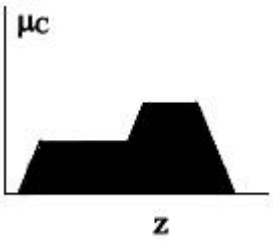
Kaidah kedua: IF x is A2 AND y is B2 THEN z is C2 (II.15)



Gambar II.13 Komposisi Aturan *Fuzzy* Metode Mamdani: Kaidah Kedua [11]

Pada gambar II.12 dan II.13 diatas dijelaskan dua buah proposisi x dan y dengan penghubung "AND" dilakukan fungsi "MIN" pada implikasi dan menghasilkan z. Selanjutnya pada agregasi atau hasil akan dilakukan fungsi "MAX" dan digambarkan sebagai berikut.

Hasil:



Gambar II.14 Komposisi Aturan *Fuzzy* Metode Mamdani: Hasil [11]

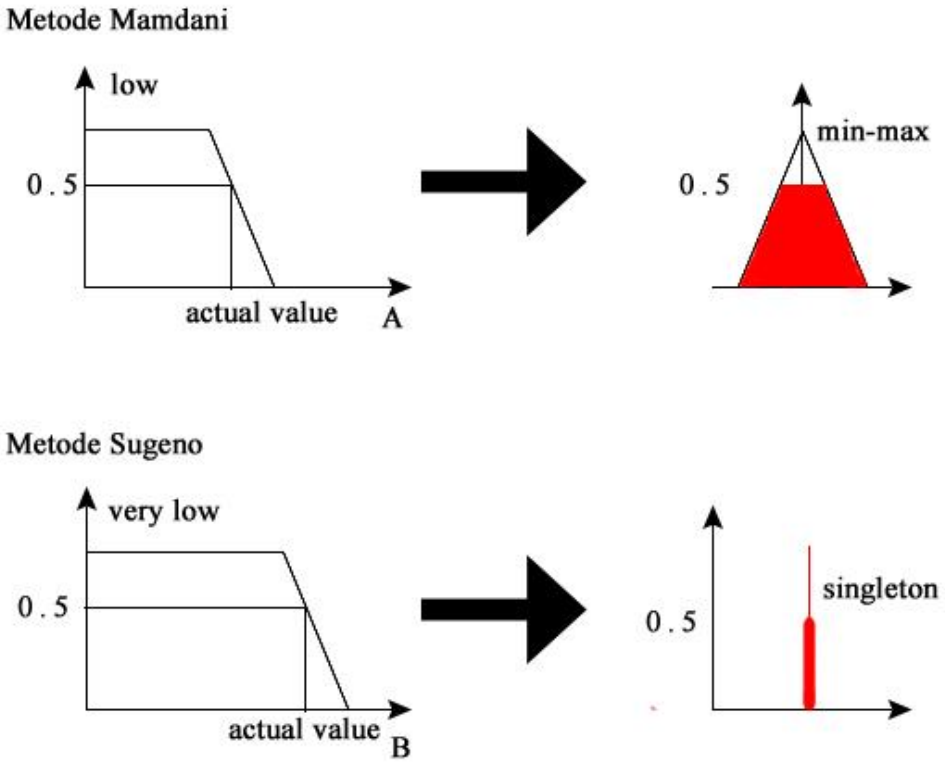
II.6.7 Metode Sugeno

Bila output dari penalaran dengan metode Mamdani berupa himpunan *fuzzy*, tidak demikian dengan metode Sugeno, dalam metode Sugeno, output sistem berupa

konstanta atau persamaan linear. Metode ini diperkenalkan oleh Takagi-Sugeno Kang pada 1985.

Pada metode Sugeno, proses *fuzzyfikasi*, operasi *fuzzy*, dan implikasi sama seperti metode Mamdani. Perbedaannya hanya pada agregasi dan *defuzzifikasi*. Jika pada metode Mamdani agregasi berupa daerah dibawah kurva, maka pada metode Sugeno agregasi berupa *singleton-singleton.*

Atau digambarkan dengan contoh sebagai berikut.



Gambar II.15 Perbedaan Metode Mamdani dan Metode Sugeno [11]

Nilai *singleton* menggunakan fungsi matematik dari variabel inputnya, dengan format sebagai berikut.

IF x is A AND y is B THEN z is f(x,y) (II.16)

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

$$z^* = \frac{\Sigma \mu_i \, . \, z_i}{\Sigma \mu_i} \quad . \qquad \text{(II.17)}$$

dengan z^* = titik pusat nilai *crisp*

μ_i = derajat keanggotaan masing-masing variabel

z_i = nilai domain pada variabel linguistik z

II.7 OOP (*Object Oriented Programming*)

Objek adalah kesatuan entitas yang memiliki sifat dan tingkah laku. Dalam kehidupan sehari-hari, objek adalah benda, baik benda berwujud nyata seperti manusia, hewan, mobil, komputer, handphone, pena, ataupun benda yang tidak nyata atau konsep, seperti halnya tabungan bank, sistem antrian, sistem *internet banking*, dan sebagainya. Jadi pengertian OOP adalah konsep yang membagi program menjadi objek-objek yang saling berinteraksi satu sama lain. Objek adalah benda, baik benda yang berwujud nyata maupun benda yang tidak nyata (konsep). Jika kita menggunakan OOP maka akan ada enam keuntungan yang dapat diperoleh, yaitu :

1. Alami (*Natural*).
2. Dapat diandalkan (*Reliable*).
3. Dapat digunakan kembali (*Reusable*).
4. Mudah untuk dalam perawatan (*Maintainable*).
5. Dapat diperluas (Extendable).
6. Efisiensi waktu.

Berikut ini beberapa bahasa pemrograman yang sudah menggunakan konsep

OOP, adalah :

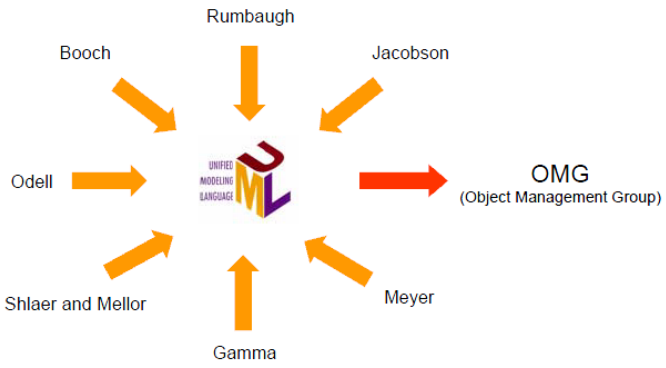
1. *C++*.
2. *Visual C++*.
3. *Visual Basic*.
4. *Java*.

II.8 UML (*Unified Modeling Language*)

UML (*Unified Modelling Language*) 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.

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 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).*



Gambar II.16 UML

Abstraksi konsep dasar UML yang terdiri dari *structural classification*, *dynamic behavior*, dan *model management*, bisa dipahami dengan mudah apabila dilihat gambar diatas dari *diagrams*. *Main concepts* bisa dipandang sebagai *term* yang akan muncul pada saat dibuat diagram. dan *view* adalah kategori dari diagram tersebut.

UML mendefinisikan diagram-diagram sebagai berikut:

1). *use case diagram*
2). *class diagram*
3). *statechart diagram*
4). *activity diagram*
5). *sequence diagram*
6). *collaboration diagram*
7). *component diagram*
8). *deployment diagram*

II.8.1 *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 atau sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu.

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.

II.8.2 *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.

II.8.3 *Statechart Diagram*

Statechart diagram menggambarkan transisi dan perubahan keadaan (dari satu *state* ke *state* lainnya) suatu objek pada sistem sebagai akibat dari *stimuli* yang

diterima. Pada umumnya *statechart diagram* menggambarkan *class* tertentu (satu *class* dapat memiliki lebih dari satu *statechart diagram*).

Dalam UML, *state* digambarkan berbentuk segiempat dengan sudut membulat dan memiliki nama sesuai kondisinya saat itu. Transisi antar *state* umumnya memiliki kondisi *guard* yang merupakan syarat terjadinya transisi yang bersangkutan, dituliskan dalam kurung siku. *Action* yang dilakukan sebagai akibat dari *event* tertentu dituliskan dengan diawali garis miring.

II.8.4 *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.

II.8.5 *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.

II.8.6 *Collaboration Diagram*

Collaboration diagram juga menggambarkan interaksi antar objek seperti *sequence diagram*, tetapi lebih menekankan pada peran masing-masing objek dan bukan pada waktu penyampaian *message*. Setiap *message* memiliki *sequence number*, di mana *message* dari level tertinggi memiliki nomor 1. *Messages* dari level yang sama memiliki *prefiks* yang sama.

II.8.7 *Component Diagram*

Component diagram menggambarkan struktur dan hubungan antar komponen piranti lunak, termasuk ketergantungan (*dependency*) di antaranya. Komponen piranti lunak adalah modul berisi *code*, baik berisi *source code* maupun *binary code*, baik *library* maupun *executable*, baik yang muncul pada *compile time, link time*, maupun *run time*.

Umumnya komponen terbentuk dari beberapa *class* dan/atau *package*, tapi dapat juga dari komponen-komponen yang lebih kecil. Komponen dapat juga berupa *interface*, yaitu kumpulan layanan yang disediakan sebuah komponen untuk komponen lain.

II.8.8 *Deployment Diagram*

Deployment/physical diagram menggambarkan detail bagaimana komponen di-*deploy* dalam infrastruktur sistem, di mana komponen akan terletak (pada mesin, server atau piranti keras apa), bagaimana kemampuan jaringan pada lokasi tersebut, spesifikasi server, dan hal-hal lain yang bersifat fisikal

Sebuah *node* adalah server, *workstation*, atau piranti keras lain yang digunakan untuk men-*deploy* komponen dalam lingkungan sebenarnya. Hubungan antar *node* (misalnya TCP/IP) dan *requirement* dapat juga didefinisikan dalam diagram ini.

II.9 C#

C# merupakan sebuah bahasa pemrograman yang berorientasi objek yang dikembangkan oleh Microsoft sebagai bagian dari inisiatif kerangka *NET Framework*. C# adalah *Java* versi *Microsoft*, sebuah bahasa *multi flatform* yang didesain untuk bisa berjalan di berbagai mesin. C# adalah pemrograman berorientasi *object* (OOP). C# memiliki kekuatan bahasa C++ dan portabilitas seperti Java. Fitur-fitur yang diambilnya dari bahasa C++ dan Java adalah desain berorientasi objek, seperti *garbage collection*, *reflection*, akar kelas (*root class*), dan juga penyederhanaan terhadap pewarisan jamak (*multiple inheritance*).

Bahasa pemrograman C# dibuat sebagai bahasa pemrograman yang bersifat *general-purpose* (untuk tujuan jamak), berorientasi objek, modern, dan sederhana. C# ditujukan agar cocok digunakan untuk menulis program aplikasi baik dalam sistem *client-server* (*hosted system*) maupun sistem *embedded* (*embedded system*), mulai dari program aplikasi yang sangat besar yang menggunakan sistem operasi yang canggih hingga kepada program aplikasi yang sangat kecil.

Meskipun aplikasi C# ditujukan agar bersifat 'ekonomis' dalam hal kebutuhan pemrosesan dan memori komputer, bahasa C# tidak ditujukan untuk bersaing secara langsung dengan kinerja dan ukuran program aplikasi yang dibuat dengan menggunakan bahasa pemrograman C.

II.10 *Unity Game Engine*

Unity Game Engine adalah *software* atau *game engine* yang digunakan untuk membuat *video game* berbasis dua atau tiga dimensi dan dapat digunakan secara gratis. Selain untuk membuat *game*, *unity* juga dapat digunakan untuk membuat konten yang interaktif lainnya seperti, visual arsitektur dan *real-time* 3D animasi.

Unity adalah sebuah sebuah *tool* yang terintegrasi untuk membuat *game*, arsitektur bangunan dan simulasi. *Unity* bisa digunakan untuk *game*s PC dan *games online*. Untuk *games online* diperlukan sebuah plugin, yaitu *Unity Web Player*, yang

sama halnya dengan *flash player* pada *browser*. Bahasa pemrograman yang digunakan bermacam-macam, mulai dari *javascript*, C#, dan *boo*.

Unity tidak bisa melakukan desain atau *modelling*, dikarenakan *unity* bukan merupakan *tools* untuk mendesain. Banyak hal yang bisa di lakukan di unity, ada fitur *audio reverb zone , particle effect , sky box* untuk menambahkan langit, dan masih banyak lagi, dan juga bisa langsung *edit texture* dari *editor* seperti *photoshop* dll.

Features (Scripting) di dalam *unity* adalah sebagai berikut:

1. Mendukung 3 bahasa pemrograman, *JavaScript*, C#, dan *Boo*.
2. *Flexible and EasyMoving, rotating,* dan *scaling objects* hanya perlu sebaris kode. Begitu juga dengan *duplicating, removing,* dan *changing properties.*
3. *Multi Platform Game* bisa di deploy di PC, Mac, Wii, iPhone, iPad dan *browser, android.*
4. *Visual Properties Variables* yang di definisikan dengan *scripts* 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.*

II.11 3D *Studio Max*

3ds Max atau *3D Studio Max* adalah salah *satu software* atau perangkat lunak yang sering digunakan oleh perancang produk untuk membuat animasi atau pemodelan dalam bentuk 3 dimensi. Aplikasi canggih ini dirilis oleh salah satu perusahaan *Autodesk Media & Entertainment* yang pada mulanya dikenal sebagai *Discreet and Kinetix*. *3D Max* merupakan salah satu dari sekian banyak aplikasi *modeling* untuk membuat model 3D dan paling banyak digunakan oleh perancang yang tersebar di seluruh dunia.

Sejalan dengan berkembangnya teknologi termasuk juga dalam bidang komputerisasi. 3D *Studio Max* pun mengalami perubahan-perubahan untuk

menyesuaikan dengan kemampuan komputer yang semakin tinggi dalam hal grafis. *3D Studio Max* dikembangkan dari aplikasi sebelumnya yang bernama *3D Studio for Dos*, tetapi aplikasi ini hanya diperuntukan untuk *platform Win32*.

Dengan semakin canggihnya kemampuan *software* ini, maka tidak aneh *3D Studio Max* menjadi program animasi komputer 3D dengan penjualan terbesar di dunia. *Software* ini memiliki kemampuan *modeling* yang kuat dan merupakan *plugin architecture* yang fleksibel dan bekerja dengan *platform Microsoft Windows*. *3D Studio Max* banyak digunakan oleh para pembuat dan perancang *video game, visual architecture, design product* dan juga studio TV untuk pembuatan animasi.