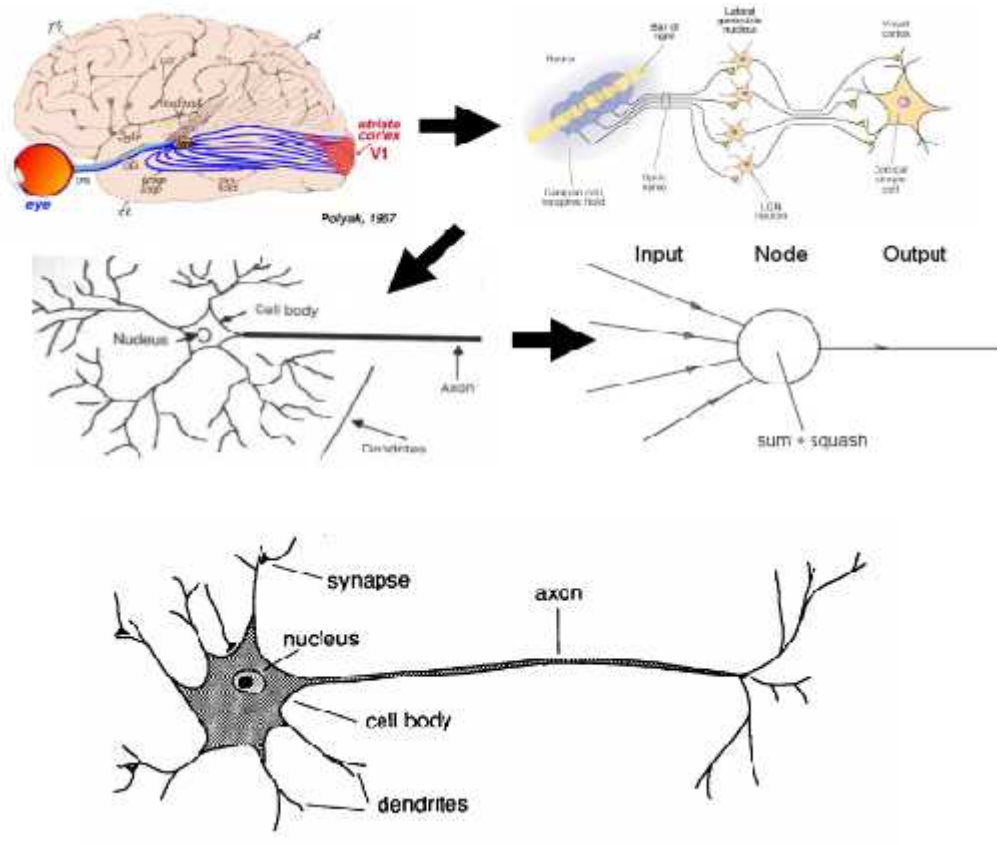
BAB II

LANDASAN TEORI

2.1 NEURAL NETWORK

Pembuatan struktur jaringan saraf tiruan diilhami oleh struktur jaringan biologi, khususnya jaringan otak manusia, dimana neuron adalah satuan unit pemroses terkecil pada otak, bentuk sederhana sebuah neuron tersebut dapat diproyeksikan sebagai berikut:



Gambar 2.1 struktur dasar jaringan syaraf tiruan dan struktur sederhana sebuah neuron[12]

Neural network backpropagation merupakan salah satu alternatif baru dalam penanganan perilaku karakter, hal ini didasari oleh banyaknya keluhan dari para pemain game FPS yang mengeluhkan respon dari AI NPC game FPS yang statis, yang mengakibatkan monotonya gameplay yang berarti kejenuhan dalam permainan.

Sifat *Neural Network* yang dinamis, baik dalam proses *learning* maupun dalam pengambilan keputusan dapat menjadi titik awal baru dalam perkembangan game FPS dengan AI NPC yang bersifat dinamis, atau dengan kata lain dengan kemampuan yanghampir medekati kemampuan *human player*, bahkan dapat melebihi kemampuan dari human player pada umumnya.

Jaringan saraf tiruan (JST) atau neural *network* adalah suatu metode komputasi yang meniru sistem jaringan saraf biologis. Metode ini menggunakan elemen perhitungan non-linier dasar yang disebut *neuron* yang diorganisasikan sebagai jaringan yang saling berhubungan, sehingga mirip dengan jaringan saraf manusia. Jaringan saraf tiruan dibentuk untuk memecahkan suatu masalah tertentu seperti pengenalan pola atau klasifikasi karena proses pembelajaran.

Layaknya *neuron* biologi, JST juga merupakan sistem yang bersifat *fault tolerant* dalam 2 hal. Pertama, dapat mengenali sinyal *input* yang agak berbeda dari yang pernah diterima sebelumnya. Sebagai contoh, manusia sering dapat mengenali seseorang yang wajahnya pernah dilihat dari foto atau dapat mengenali sesorang yang wajahnya agak berbeda karena sudah lama tidak menjumpainya. Kedua, tetap mampu bekerja meskipun beberapa *neuron*-nya tidak mampu bekerja dengan baik. Jika sebuah *neuron* rusak, *neuron* lain dapat dilatih untuk menggantikan fungsi *neuron* yang rusak tersebut. JST menyerupai otak manusia dalam dua hal, yaitu:

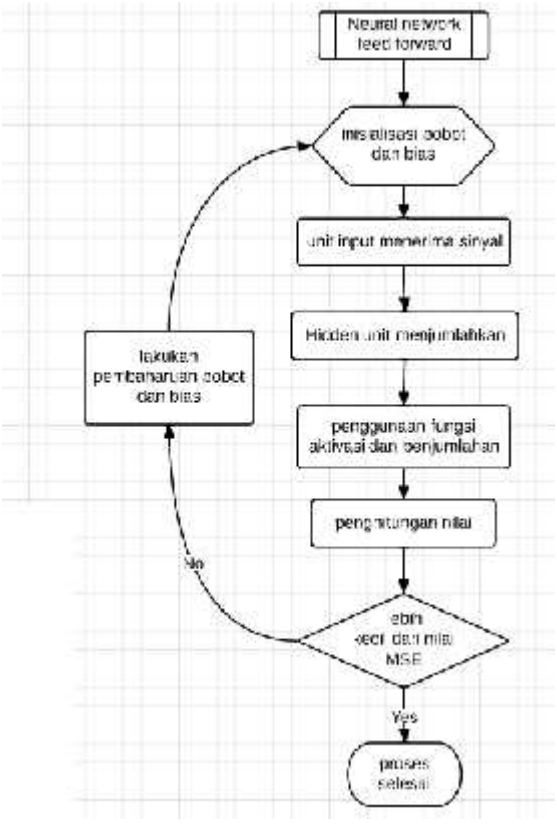
a. Pengetahuan diperoleh jaringan melalui proses belajar
b. Kekuatan hubungan antar sel syaraf (*neuron*) yang dikenal sebagai bobot-bobot sinaptik digunakan untuk menyimpan pengetahuan.

Menurut (Siang, 2004) JST ditentukan oleh 3 hal:

a. Pola Hubungan Antar Neuron
b. Metode untuk menentukan bobot penghubung neuron
c. Fungsi Aktivasi

Salah satu dari jenis Neural network, yaitu *Multi layer Feed forward* atau yang lebih dikenal sebagai MLF neural network merupakan jenis *Neural network* yang menggunakan proses terbimbing dengan tujuan utama meminimalisir nilai *mean square eror*, dimulai dengan pemunculan nilai random sebagai *weight* yang akan diproses melalui tahapan iterasi bertingkat.

Langkah yang dilakukan dalam algoritma MLF tidak jauh berbeda dengan yang digunakan dalam algoritma *backpropagation*, yang membedakan hanya tidak adanya proses propagasi balik yang memungkinkan algoritma memiliki waktu proses yang lebih cepat bila dibandingkan algoritma *backpropagation*, berikut adalah flowchartnya:



Gambar 2.2 flowchart *neural network feed forward*\

Adapun tahapannya adalah:

a. Inisialisasi bobot dan bias, bobot dan bias dapat di set secara sembarang dengan menggunakan angka acak, namun pada kasus ini nilainya dibatasi antara -1 hingga 1.

b. Setiap unit input yang telah ditentukan sebelumnya menerima sinyal input dan menyebarkanya ke seluruh unit yang ada pada *hidden layer*, adapun input yang digunakan adalah inputan yang berasal dari input *data training* yang sudah diskalakan.

c. Setiap *hidden unit* akan menjumlahkan sinyal sinyal input yang sudah memiliki bobot, begitupula dengan biasnya.

$$U_in = U_{0j} + \sum_{i=1}^{n} XiUij \qquad (1)$$

Dimana U_in = sinyal input U

U_{0j} = Bias pada lapisan tersembunyi U

X_i = Unit input

U_{ij} = Bobot pada lapisan tersembunyi U

d. Penggunaan fungsi aktivasi yang telah ditentukan untuk menghitung sinyal output dari *hidden unit* yang bersangkutan lalu mengirimkan sinyal output ke seluruh unit pasa unit output

$$U = 1/1+e^{-U_in} \qquad (2)$$

e. Setiap unit output akan menjumlahkan sinyal sinyal input yang sudah berbobot, termasuk biasnya .

$$U3 = f(U3_in) \qquad (3)$$

Dimana $f(U_n_in)$ adalah nilai aktifasi unit hiden U (dimana n = 1,2,3)

f. Bila proses penghitungan sinyal input sudah selesai secara keseluruhan, maka langkah selanjutnya adalah penghitungan MSE yang memiliki fungsi

$$MSE=0{,}5 \times \{(t_{k1}-y_{k1})^2+ (t_{k2}-y_{k2})^2+\ldots+ (t_{km}-y_{km})^2\} \qquad (4)$$

MSE berfungsi sebagai penentu stopping process dalam proses MLF.

g. Apabila nilai MSE masih belum lebih kecil dari nilai yang ditentukan, maka akan dilakukan pembaharuan bobot dan bias, yang memiliki fungsi

- Unit output

  $w_{jk}(baru)=w_{jk}(lama) + \Delta w_{jk}$ (5)

- hiden unit

  $v_{ij}(baru)=v_{ij}(lama) + \Delta v_{ij}$ (6)

Kemudian proses akan terus berlanjut hingga syarat terpenuhi atau jumlah *eppoch* maksimal default tercapai, yaitu 20.000 *eppoch*.

Selain itu terdapat beberapa kelebihan dan kekurangan dari MLF[3], yaitu:

Kelebihan:

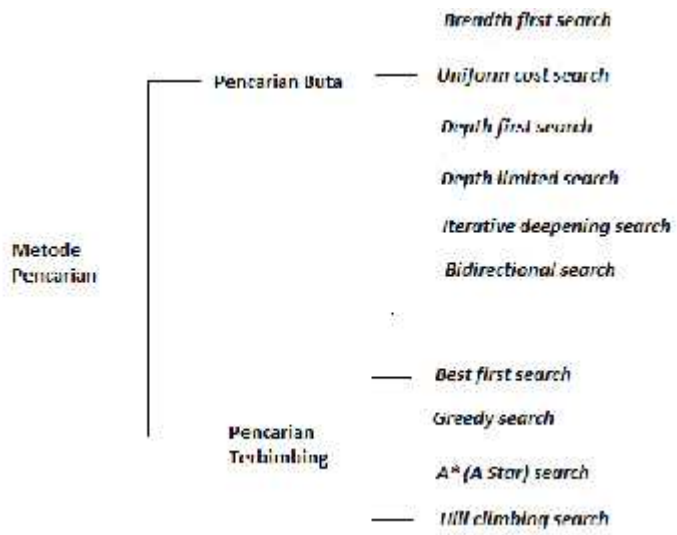
a. Proses *learning* yang memiliki kemampuan untuk beradaptasi secara mandiri walaupun tanpa pengawasan dari user.
b. *Nonlinearity*, yaitu memposisikan neuron sebagai unit non linear, hal ini merupakan sebuah properti yang sangat penting, karen a relasi antara input dan output yang dihasilkan bersifat *non linear*.
c. Input Output mapping, memungkinkan MLF untuk mempelajari data yang ada melalui struktur antara input dan output yang dihasilkan saat mencapai nilai yang stabil.
d. Robustness, memiliki kemampuan pemrosesan yang cepat.

Kekurangan:

a. Perlunya perancangan struktur layer yang sesuai dengan data inputan untuk mencegah nilai bias yang memiliki nilai statis bahkan membesar pada proses iterasi yang dilakukan
b. Hasil yang diperoleh akan kurang optimal jika terjadi kesalahan pemilihan struktur layer yang digunakan.

2.2 Algoritma Pendukung

Selain penggunaan *neural network* dalam pengambilan keputusan, dibutuhkan pula algoritma pencarian rute, 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.



Gambar 2.3 Bagan Metode Pencarian [4]

Pada gambar di atas, metode penulusuran 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*.

2.2.1 Pencarian Buta

Pencarian buta (*Blind Search*) adalah pencarian solusi tanpa adanya informasi yang dapat mengarahkan pencarian untuk mencapai *goal state* dari *current state* [4]. 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 banyanknya *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* [4].

2.2.2 Pencarian Terbimbing

Kata heuristik berasal dari kata kerja bahasa Yunani, heuriskein, yang berarti mencari atau menemukan. Dalam dunia pemograman, 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. Di dalam mempelajari metode – metode pencarian ini, kata heuristik diartikan sebagai suatu fungsi yang memberikan suatu nilai berupa biaya perkiraan (estimasi) dari suatu solusi [4].

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 dimana *heuristic* lebih merupakan perkiraan untuk membantu algoritma dan tidak harus valid setiap waktu. Beberapa contoh algoritma pencarian yang mengguanakan metode *heuristic search* adalah *Best First Search, Greedy Search*, A* (A Star) dan *Hill Climbing Search* [4].

2.2.3 Algoritma A* *(*A Star)

Algoritma A* (A Star) merupakan perbaikan dari metode BFS dengan memodifikasi fungsi heuristicnya. 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 kita mencari 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 yang hampir sama kecuali dengan dua perbedaan yaitu :

1. Simpul-simpul di *list* terbuka diurutkan oleh *cost* keseluruhan dari simpul awal ke simpul tujuan, dari *cost* terkecil sampai *cost* terbesar. *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 kesimpul tujuan adalah penjumlahan jarak dari simpul tersebut ke simpul tujuan. Untuk lebih mudah dimengerti, berikut rumusnya.

$$f(n) = g(n) + h(n)$$

(II.1)

keterangan :

f(n) = fungsi evaluasi

g(n) = biaya yang sudah dikeluarkan dari keadaan awal sampai keadaan h(n) = estimasi biaya untuk sampai pada suatu tujuan mulai dari n

Node dengan nilai terendah merupakan solusi terbaik untuk diperiksa pertama kali pada g(n) + h(n). Dengan fungsi *heuristic* yang memenuhi kondisi tersebut maka pencarian dengan algoritma A* dapat optimal.Keoptimalan dari A* cukup langsung dinilai optimal jika h(n) adalah *admissible heuristic* yaitu nilai h(n) tidak akan memberikan penilaian lebih pada *cost* untuk mencapai tujuan. Salah satu contoh dari *admissible heuristic* adalah jarak dengan menarik garis lurus karena jarak terdekat dari dua titik adalah dengan menarik garis lurus [4].

Beberapa terminologi dasar yang terdapat pada algor itma 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:

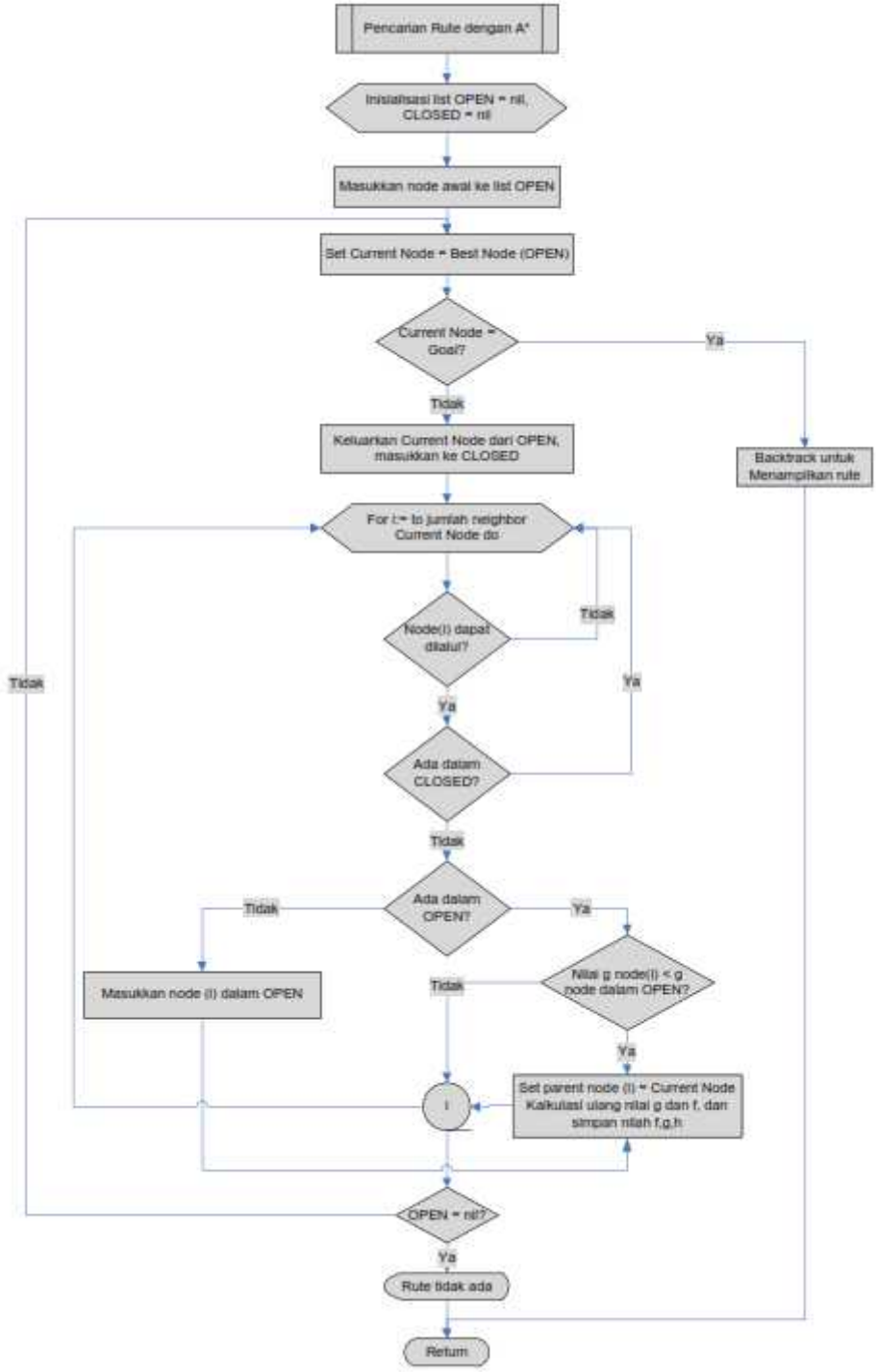
1. Menambahkan *target point* ke dalam *closed list*, dalam hal ini jalan telah ditemukan, atau,
2. Gagal untuk menemukan *target point*, dan *open list* kosong. Dalam kasus ini, tidak ada jalan.

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 2.4 *Flowchart* Algoritma A* [5].

2.3 Pengertian Game

Game adalah kebutuhan dasar setiap manusia untuk menikmati hidup dan sebagai media pembelajaran. Game adalah aktifitas yang melibatkan satu atau lebih pemain.*Game* dapat pula diartikan sebagai tujuan yang ingin dicapai pemain atau sekumpulan aturan yang menandakan apa yang dilakukan pemain dan yan tidak dapat dilakukan. *Game* dimainkan terutama untuk hiburan,kesenangan,tetapi dapat juga berfungsi sebagai sarana latihan,pendidikan dan simulasi

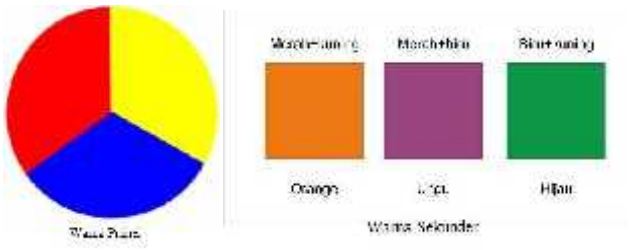
Pengertian game pada umumnya berarti aktifitas yang bisa berupa tindakan nyata ataupun tindakan di dalam suatu sistem/aplikasi yang dapat membawa kesenangan/hiburan bagi penggunanya. Dimana hiburan yang didapat tetap mempunyai aturan dan target.

2.4 Unsur Game

Dalam sebuah *game* terdapat unsur-unsur yang melengkapinya. Berikut beberapa unsur dalam suatu *game* yaitu [6]:

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 2.5 Warna *Primer* dan *Sekunder* [6]

2. Komposisi

Komposisi adalah pengaturan segala elemen didalam 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 pengkuran secara seksama, komposisi yang sesuai dengan anak-anak adalah komposisi yang sederhana dan tidak menggunakan petunjuk terlalu rumit. Kesederhanaan diwujudkan dengan penggunaan visual 2D dan penerapan warna-warna dalam seluruh aspek desain.

3. Bentuk dasar

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



Gambar 2.6 Bentuk Dasar [6]

4. Tipografi

   Tipografi merupakan representasi visual dari sebuah bentuk komunikasi *verbal* dan merupakan *property visual* 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 sudah termasuk didalamnya atau dapat ditambahkan pada komputer.

2.5 Jenis-jenis Game

Berikut ini beberapa jenis *game* berdasarkan cara pembuatannya, cara pemasarannya dan mesin yang menjalankannya [6]. Jenis-jenis *game* tersebut adalah :

1. *Game* PC

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

2. *Game Console*

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

3. *Game Arcade*

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

4. *Game Online*

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

2.6 Genre Game

Berdasarkan *genre*, *game* dapat dibagi menjadi beberapa *genre* yaitu [6] :

1. *Action Game*

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

a. *Action Adventure Game*

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

b. *Stealth Game*

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

c. *Survival Horror Game*

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

d. *Beat'em Up Game*

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

e. *Fighting Game*

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

f. *Maze Game*

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

g. *Platfrom Game*

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

h. *Shooter*

a) *First Person Shooter Game*

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

b) *Massively Multiplayer Online First Person Shooter Game*

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

c) *Third Person Shooter Game*

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

2.7 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 [7]

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*.

2.8 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 [8]. *UML* adalah metodologi untuk mengembangkan sistem OOP dan sekelompok perangkat *tool* untuk mendukung pengembangan sistem tersebut.

UML mulai diperkenalkan oleh *Object Management Group*, sebuah organisasi yang telah mengembangkan model, teknologi, dan standar OOP sejak tahun 1980-an. Sekarang UML sudah mulai banyak digunakan oleh para praktisi OOP. UML merupakan dasar bagi perangkat (*tool*) desain berorientasi objek dari IBM. UML menyediakan 10 macam diagram yang dapat digunakan untuk memodelkan aplikasi berorientasi objek yang dibuat, yaitu :

2.8.1 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 [8].

2.8.2 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 [8].

2.8.3 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 [8].

2.8.4 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 [8].

2.8.5 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 [8].

2.8.6 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 [8].

2.8.7 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 [8].

2.8.8 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 [8].

2.9 C#

C# merupakan sebuah bahasa pemrograman yang berorientasi objek yangdikembangkan oleh Microsoft sebagai bagian dari inisiatif kerangka .NET *Framework* [8]. 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 *clien-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 [9].

2.10 Unity

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* [10].

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.*

2.11 3ds 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 [11].

Sejalan dengan berkembangnya teknologi termasuk juga dalam bidang komputerisasi. 3D *Max* pun mengalami perubahan-perubahan untuk menyesuaikan dengan kemampuan komputer yang semakin tinggi dalam hal grafis. *3D 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 Max* menjadi program animasi komputer 3D dengan penjualan terbesar 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[11].