BAB 2
TINJAUAN PUSTAKA

2.1 Game

2.1.1 Pengertian *Game*

Game merupakan kata dalam bahasa Inggris yang berarti permainan. Permainan adalah kegiatan yang kompleks yang didalamnya terdapat peraturan, *play* dan budaya. Sebuah permainan adalah sebuah sistem dimana pemain terlibat dalam konflik buatan, yang mana pemain berinteraksi dengan sistem dan konflik dalam permainan tersebut, didalam permainan terdapat peraturan yang bertujuan untuk membatasi perilaku pemain dan menentukan permainan. *Game* sebenarnya bertujuan untuk menghibur, yang biasanya banyak disukai oleh anak-anak hingga orang dewasa sekalipun. *Game* juga sebenarnya penting untuk perkembangan otak, diantaranya meningkatkan konsentrasi dan melatih memecahkan masalah dengan tepat dan cepat karena dalam *game* terdapat berbagai konflik atau masalah yang menuntut untuk diselesaikan dengan cepat dan tepat. Permainan dalam hal ini merujuk pada pengertian kelincahan intelektual (*Intellectual Playability Game*) yang juga bisa diartikan sebagai arena keputusan dan aksi pemainnya.

Video game adalah *game* yang berbasis elektronik dan visual. *Video game* dimainkan dengan memanfaatkan media visual elektronik.[2]

2.1.2 Sejarah *Game* Komputer

Game komputer adalah sebuah permainan interaktif yang dijalankan menggunakan komputer. Program komputer menerima input dari si pemain melalui pengendali dan menampilkan lingkungan buatan melalui TV atau layar monitor. Sejarah teknologi *game* komputer secara langsung berhubungan dengan perkembangan komputer itu sendiri.

Game komputer generasi pertama, tahun 1952, adalah sebuah permainan OXO (tic-tac-toe) dalam versi grafik yang diciptakan oleh seorang mahasiswa Universitas Cambridge bernama A.S. Gouglas. *Game* ini diprogram dengan memakai komputer EDVAC vacuum tube yang memiliki layer berupa *cathode ray tube* (CRT). Pada tahun 1958. William Haginbotham menciptakan *video game* pertamanya, dan *game* komputer pertama yang benar-benar diciptakan menggunakan komputer itu sendiri adalah Spacewar oleh Steve Russel yang terinspirasi dari kisah fiksi ilmiah berjudul Skylark karangan Edward E Smith. Steve Russel bekerja menggunakan sebuah komputer mainframe bernama MIT PDP-1 yang biasa dipakai untuk perhitungan statistic pada tahun 1961. Saat itu di tahun 60-an, komputer adalah barang sangat mewah. Komputer umumnya dipakai untuk kepentingan riset dalam dunia militer.

Game generasi kedua, di tahun 1971, Nolan Bushnell bersama Ted Dabney menciptakan *game* bergenre arcade yang pertama yang diberi judul Komputer Space, yang didasari oleh Spacewar. Tahun 1972, Nolan dan Ted memulai Atari komputer. Mereka mengembangkan *game* berjudul Pong yaitu *game* pertama yang tersedia untuk publik. Karena sebelumnya *game* hanya ada di

dalam komputer mainframe yang digunakan untuk kesenangan sendiri. Asal usul Pong dimulai saat Nolan ingin membuat *game* sederhana dan mudah dimengerti. Menggunakan memory dan micro processor kelas rendah, dengan kemampuan grafis yang sederhana, akhirnya dia membuat versi elektronik dari permainan ping pong yang kemudian menjadi Pong. Pong kemudian berevolusi menjadi sebuah *game* bernama Breakout. *Game* itu diciptakan oleh Steve Jobs untuk Atari, dan bersama Steve Wozniak mulai berpikir untuk menciptakan sebuah PC. Mereka meminjam semua peralatan yang dipakai dalam proyek Breakout untuk membuat sebuah prototype bernama Apple I. dan itu adalah cikal bakal komputer Apple Macintosh yang ada sekarang.

Pada tahun 1980, Atari mengeluarkan *game* berjudul Asteroid dan Lunar Lander. Kedua *game* tersebut adalah *game* pertama yang didaftarkan kepada perusahaan untuk mendapatkan hak cipta yang paten. Asteroid merupakan *game* yang penuh inovasi baru dalam grafisnya, dengan menggunakan metode raster, *game* ini merupakan *grafis vector line* seperti yang ada pada *osciloscope*. Pengenalan Atari Video komputer *System* (Atari 2600) dengan CPU biasa dan slot untuk kasetnya, yang menjadi suatu era baru dalam dunia *game*. Di tahun 1980 itu juga menunjukkan adanya peningkatan penjualan PC yang biasa dipakai untuk *game*.

Kini komputer bersaing dengan mesin *console* seperti PS atau Xbox. Komputer tidak hanya sebagai mesin untuk kepentingan bisnis semata, tapi juga untuk hiburan seperti *game*.[3]

2.1.3 Elemen Dasar *Game*

Dalam suatu *game* terdapat beberapa elemen yang menyusun *game* tersebut, seperti jenis *game*, karakter *game*, *background*, elemen *sound*/suara, dan gerakan-gerakan dasar.

2.1.4 Jenis "*Platform*" / Alat yang digunakan

Ada beberapa *Platform*/alat yang digunakan untuk menjalankan suatu *game*, diantaranya adalah :

1. ***Arcade games***

   Game yang dijalankan pada mesin dengan input dan output audio visual yang telah terintegrasi dan tersedia ditempat-tempat umum, seperti mal, bandara dan sebagainya. Contoh permainan ding-dong.

2. ***PC Games***

   Game yang dimainkan pada PC (*Personal Computer*) yang memiliki kelebihan yaitu memiliki tampilan antarmuka yang baik untuk I/O, output visual kualitas tinggi karena layar computer biasanya memiliki resolusi yang jauh lebih tinggi dibandingkan dengan layar televisi biasa.

3. ***Console games***

   Game yang dijalankan pada suatu mesin spesifik yang biasanya tersedia di rumah pribadi, seperti Playstation, Microsoft Xbox 360, Nintendo Wii, Sega dan sebagainya.

4. ***Handheld games***

   Game yang dimainkan di console khusus *video game* yang dapat dibawa kemana-mana, contoh *GameBoy*, Nintendo DS dan Sony PSP.

5. ***Mobile games***

   Game yang dapat dimainkan khusus untuk mobile phone atau PDA.

2.1.5 Genre dan Jenis-jenis *Game* Komputer

Video Game saat ini mempunyai banyak jenis *platform/console*, yang diantaranya PC termasuk dalam *console* generasi ketiga selama perjalanannya, karena itu jenis *console* ini mempunyai cukup banyak *genre game*. Berikut ini adalah *genre-genre game,* khususnya yang biasa dimainkan dengan *console* PC (berdasarkan pengalaman pemain dalam memainkan *game* tersebut atau yang biasa disebut *gameplay*).

1. ***Action Game***

   Action Game dikategorikan sebagai *gameplay* dengan model pertarungan, *Action Game* dibagi kedalam beberapa jenis *game,* diantaranya :

   a. *Action Adventure Game*
   b. *Stealth Game*
   c. *Survival Horror Game*
   d. Beat 'em Up Game
   e. *Fighting Game*
   f. *Maze Game*
   g. *Platform Game* atau *platformers*
   h. Shooter

Dalam *game Shooter* terbagi kedalam beberapa jenis, diantaranya :

a) *First-Person Shooter Game*

b) *Massively multiplayer online first person shooter Game*

c) *Third-person shooter Game*

d) *Tactical shooter Game*

e) *Light-gun Game*

f) *Shoot 'em up Game*

2. ***Adventure Game***

Gameplay jenis ini adalah keharusan *player* memecahkan bermacammacam *puzzle* melalui interaksi dengan orang lingkungan dalam *game* tersebut. *Adventure Game* dibagi kedalam beberapa jenis *game*, diantaranya :

a. *Text adventure / Interactive fiction Game*

b. *Graphical adventure Game*

c. *Visual novel Game*

d. Interactive movie Game

e. *Dialog* G*ame*

3. ***Role-Playing Game***

Computer role-playing games (CRPGs atau RPGs) mempunyai *gameplay* di mana *player* akan bertualang dengan *skill combat* atau *cast spell* dalam cerita *game*. *Role-Playing Game* dibagi kedalam beberapa jenis *game*, diantaranya :

a. *Action Role-Playing Game*

b. *Massively multiplayer online role-playing Game*

c. *Tactical role-playing Game*

4. ***Strategy Game***

Strategy game berfokus pada *gameplay* dimana dibutuhkan pemikiran yang tepat agar dapat meraih kemenangan. *Strategy Game* dibagi kedalam beberapa jenis *game*, diantaranya :

a. *Real-time strategy* (RTS) *and turn-based strategy* (TBS) *Game.*

b. *Tactical Game*

c. 4X *Game*, yang berarti *eXplore* (penjelajahan), *eXpand* (meluas), *eXploit* (menja-jah), dan *eXterminate* (memusnahkan).

d. *Artillery Game*

5. ***Simulation Game***

Genre ini bertujuan untuk memberi pengalaman melalui simulasi. *Simulation Game* dibagi kedalam beberapa jenis *game*, diantaranya :

a. *Construction and management simulation Game* (CMSs), adalah tipe *game* simulasi di mana *player* harus mendirikan, memperluas, atau mengelola komunitas atau proyek fiksi. Yang terdiri dari :

a) *City-Building Game*

b) *Economic-simulation Game*

c) *God Game*

d) *Government simulation Game*

6. ***Vehicle Simulation Game***

   Vehicle Simulation Game, adalah tipe *game* simulasi yang berusaha memberikan *player* sebuah pengalaman realistik mengendarai kendaraan-kendaraan tertentu. Yang terbagi kedalam beberapa jenis, diantaranya :

   a. *Flight Game*
   b. *Racing Game*
   c. *Space Game*
   d. *Train Game*
   e. *Vehicular (Car) Combat.*[4]

7. ***Time Management Game***

   Time Management Game merupakan sebuah permainan yang menuntut kecepatan dan ketepatan, juga strategi agar bisa memanfaatkan waktu sebaik-baiknya untuk menyelesaikan tugas yang diberikan.

   Manajemen waktu adalah melakukan perencanaan, pengorganisasian, pelaksanaan dan pengendalian waktu secara efektif dan efisien. Bagaimana memahami waktu untuk berkonsentrasi pada tugas yang sedang ditangani, bagaimana melakukan tugas pada saat yang bersamaan.

2.2 AI (*Artificial Intelligence*)

2.2.1 Pengertian AI

Artificial Intelligence (AI) atau kecerdasan buatan merupakan sub-bidang pengetahuan komputer yang khusus ditujukan untuk membuat *software* dan *hardware* yang sepenuhnya bisa untuk menirukan beberapa fungsi otak manusia.

Dengan demikian diharapkan komputer bisa membantu manusia dalam memecahkan berbagai masalah yang lebih rumit. AI juga didefinisikan sebagai kecerdasan yang ditunjukkan oleh suatu entitas buatan. Sistem seperti ini umumnya dianggap komputer. Kecerdasan diciptakan dan dimasukkan kedalam suatu mesin (komputer) agar dapat melakukan pekerjaan seperti yang dapat dilakukan manusia. Diantara yang menggunakan kecerdasan buatan antara lain sistem pakar, *games*, logika fuzzy, jaringan syaraf tiruan dan robotika.

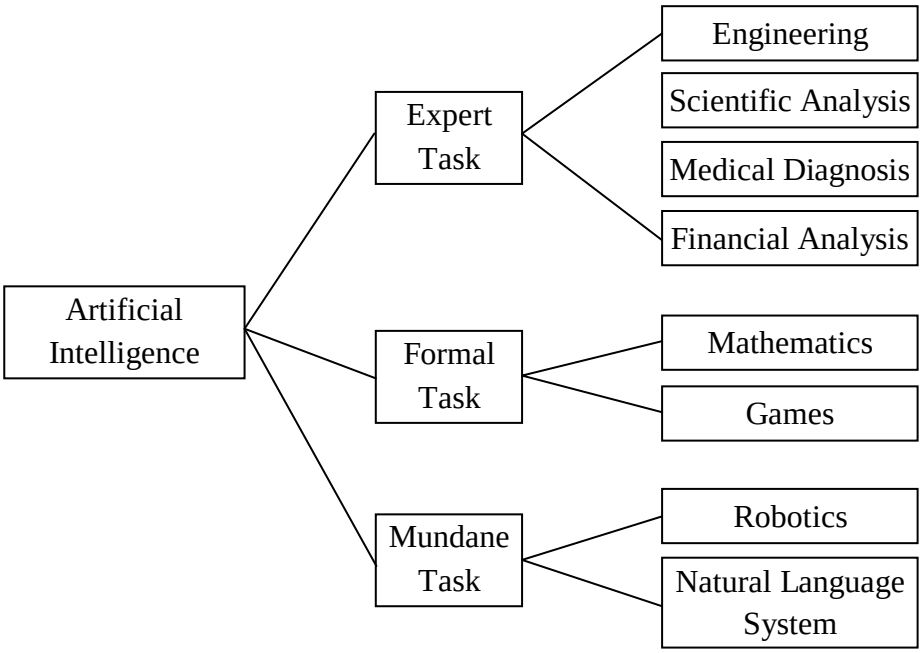
Ada beberapa pengertian *Artificial Intelligence* diantaranya sebagai berikut :

1. Suatu cara yang sederhana untuk membuat komputer dapat "berpikir" secara *intelligent*.
2. Bagian dari ilmu komputer yang mempelajari perancangan sistem komputer yang *intelligent*, yaitu suatu sistem yang memperlihatkan karakteristik yang ada pada tingkah laku manusia.
3. Suatu studi bagaimana membuat komputer dapat mengerjakan sesuatu, yang pada saat ini orang dapat mengerjakan lebih baik.
4. Bidang ilmu komputer yang memungkinkannya untuk memahami, bernalar dan bertindak.

Dari beberapa pengertian diatas, definisi AI dapat disimpulkan kedalam empat kategori, yaitu :

1. Sistem yang dapat berfikir seperti manusia *"Thinking humanly"*.
2. Sistem yang dapat bertingkah laku seperti manusia *"Acting Humanly"*.
3. Sistem yang dapat berfikir secara rasional *"Thinking rationally*".
4. Sistem yang dapat bertingkah laku secara rasional *"Acting rationally"*.

AI seperti bidang ilmu lainnya juga memiliki sejumlah sub-disiplin ilmu yang sering digunakan untuk pendekatan yang esensial bagi penyelesaian suatu masalah dan dengan aplikasi bidang AI yang berbeda. Gambar 2.1 merupakan sejumlah bidang-bidang tugas *(task domains)* dari AI.



Gambar 2.1. Bidang-bidang tugas *(task domains)* dari AI

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

1. *Mundane task*

   Secara harfiah, arti *mundane* adalah keduniaan. Di sini, AI digunakan untuk melakukan hal-hal yang sifatnya duniawi atau melakukan kegiatan yang dapat membantu manusia. Contohnya :

   a) Persepsi (*vision* & *speech*).

   b) Bahasa alami (*understanding, generation* & *translation*).

   c) Pemikiran yang bersifat *commonsense*.

   d) *Robot control*.

2. *Formal task*

   AI digunakan untuk melakukan tugas-tugas formal yang selama ini manusia biasa lakukan dengan lebih baik. Contohnya :

   a) Permainan/*games*.

   b) Matematika (geometri, logika, kalkulus, integral, pembuktian).

3. *Expert task*

   AI dibentuk berdasarkan pengalaman dan pengetahuan yang dimiliki oleh para ahli. Penggunaan ini dapat membantu para ahli untuk menyampaikan ilmu-ilmu yang mereka miliki. Contohnya :

   a) Analisis finansial

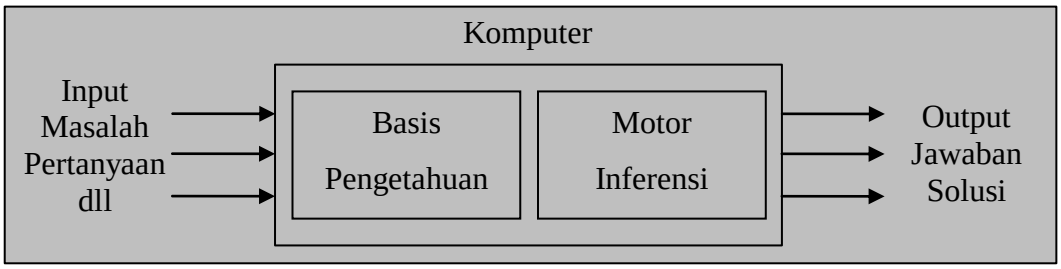
   b) Analisis medikal

   c) Analisis ilmu pengetahuan

   d) Rekayasa (desain, pencarian, kegagalan, perencanaan, manufaktur)

Aplikasi *Artificial Intelegent* 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.



Gambar 2.2. Penerapan konsep kecerdasan buatan di komputer [4]

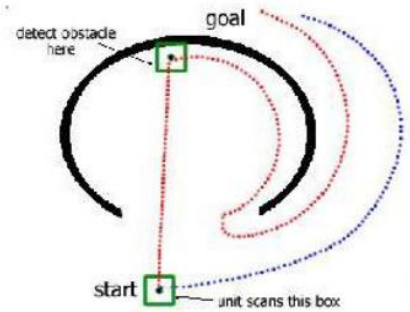
2.2.2 Teknik-Teknik Dasar Pencarian

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. [4]

2.2.2.1 Pathfinding

Pathfinding (pencarian jalan/*rute*) adalah salah satu bidang penerapan yang sering ditangani oleh kecerdasan buatan khususnya dengan menggunakan algoritma pencarian. Penerapan yang dapat dilakukan dengan *pathfinding* antara lain adalah pencarian *rute* dalam suatu *game* dan pada suatu peta. Algoritma pencarian yang dipakai harus dapat mengenali jalan dan elemen peta yang tidak dapat dilewati.

Sebuah algoritma *pathfinding* yang baik dapat bermanfaat untuk mendeteksi halangan/rintangan yang ada pada medan dan menemukan jalan untuk menghindarinya, sehingga jalan yang ditempuh lebih pendek daripada yang seharusnya bila tidak menggunakan algoritma *pathfinding*. Lihat ilustrasi pada Gambar 2.3.



Gambar 2.3. Penentuan *rute* tanpa *pathfinding*

Pada Gambar 2.5, dari *start* menuju *goal*, tanpa algoritma *pathfinding*, unit hanya akan memeriksa lingkungan sekitarnya saja (dilambangkan dengan daerah di dalam kotak hijau). Unit tersebut akan maju terus ke atas untuk mencapai tujuan, baru setelah mendekati adanya halangan, lalu berjalan memutar untuk menghindarinya. Sebaliknya, penentuan *rule* dengan algoritma *pathfinding* pada Gambar 2.4 akan memprediksi ke depan mencari jalan yang lebih pendek menghindari halangan (dilambangkan garis biru) untuk mencapai tujuan, tanpa pernah mengirim unit ke dalam „perangkap" halangan berbentuk U. Karena itu peran algoritma *pathfinding* sangat berguna untuk memecahkan berbagai permasalahan dalam penentuan *rute*.



Gambar 2.4. Penentuan *rute* dengan *pathfinding*

2.2.2.2 Algorima Pencarian (*Search Algorithms*)

Penerapan kecerdasan buatan *(Artificial intelligence)* untuk pemecahan masalah *(problem solving)* dalam bidang ilmu komputer telah mengalami perkembangan yang pesat dari tahun ke tahun seiring perkembangan kecerdasan buatan itu sendiri. Permasalahan yang melibatkan pencarian *(searching)* adalah salah satu contoh penggunaan kecerdasan buatan yang cukup populer untuk memecahkan berbagai macam permasalahan.

Penerapannya bermacam-macam, mulai masalah dunia nyata, seperti penetuan rute pada suatu peta, *travelling salesman problem* (TSP), penentuan urutan perakitan *(assembly sequencing)* oleh robot, sampai penerapan dalam dunia *game*, seperti membuat komputer dapat bermain catur layaknya manusia ataupun penentuan pengambilan jalan karakter dalam sebuah *game*.

Dalam cara kerjanya, sistem menggunakan algoritma tertentu untuk mencapai kondisi yang diinginkan atau menemukan hasil yang dicari dari kondisi atau input yang ada sekarang. Dalam algoritma pencarian, dikenal istilah *'state'* yang berarti kondisi. Kondisi akhir yang hendak dituju dikenal dengan istilah *'goal state'.* Contoh *state* antara lain, dalam *game* catur misalnya, adalah letak tiap buah catur pada papan. *Goal state* dalam kasus ini biasanya kondisi raja ter-skak mati.

Pada umumnya, algoritma pencarian bekerja dengan mengembangkan berbagai kemungkinan *state* yang mungkin dicapai dari *state* sekarang. *State* dalam proses pencarian biasa disebut dengan istilah *node*. Kumpulan *node* akan terus dikembangkan sampai ditemukan *node* yang merupakan *goal state* atau bila

sudah tidak ada lagi *node* yang dikembangkan (berarti tidak ditemukan solusi). Rangkaian kumpulan *node* dari *state* awal sampai *goal state* yang terbaiklah yang akhirnya diambil menjadi solusi. Kriteria terbaik disini tergantung pada kasus yang dihadapi. Pada penetuan rute misalnya, terbaik biasanya adalah solusi yang memberikan jalan terpendek atau tercepat untuk mencapai tujuan.

Berbagai algoritma untuk pencarian *(searching algorithm)* yang ada berbeda satu dengan yang lain dalam hal pengembangan kumpulan *node* untuk mencapai *goal state*. Perbedaan ini terutama dalam hal cara dan urutan pengembangan *node*, dan sangat berpengaruh pada kinerja masing-masing algoritma. Empat kriteria yang menjadi ukuran algoritma pencarian adalah :

- *Completeness*

  apakah algoritma pasti dapat menemukan solusi?

- *Time Comlexity*

  berapa lama waktu yang dibutuhkan untuk menemukan sebuah solusi?

- *Space Complexity*

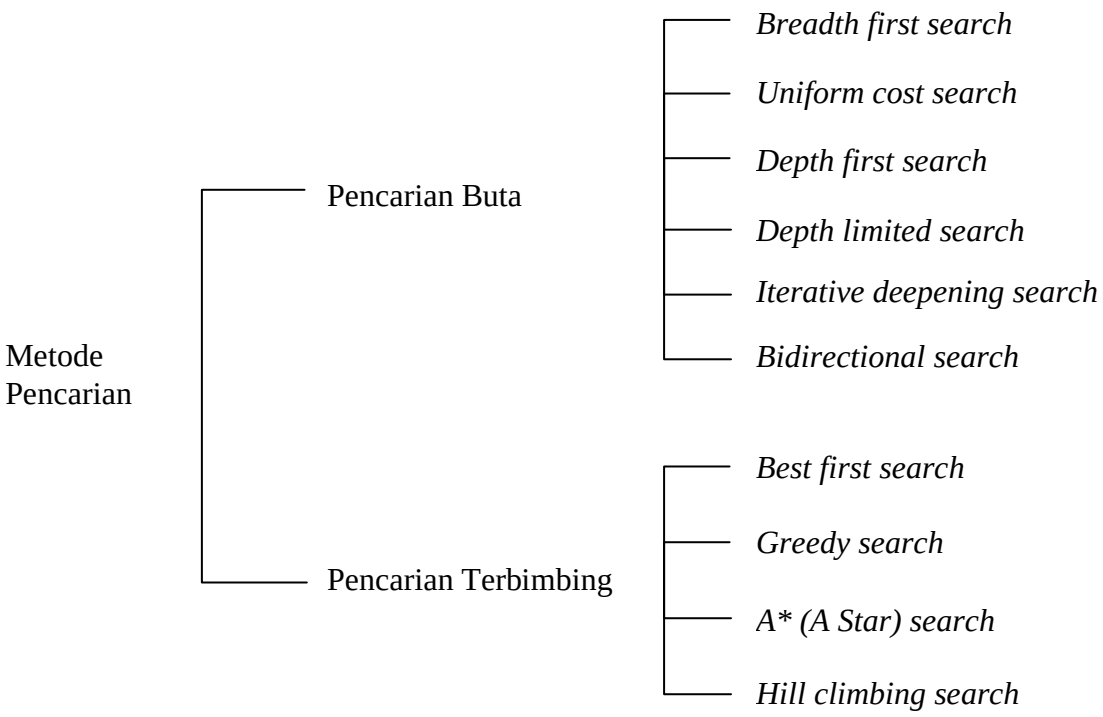
  berapa memori atau *resource* yang diperlukan untuk melakukan pencarian?

- *Optimality*

  apakah algoritma tersebut dapat menemukan solusi yang terbaik jika terdapat beberapa solusi yang berbeda?

Namun perlu diingat, algoritma pencarian yang dikatakan terbaik sekalipun belum tentu sesuai untuk semua jenis kasus/masalah. Harus dipilih algoritma pencarian yang sesuai dengan kebutuhan kasus yang dihadapi. Menurut cara algoritma mengembangkan *node* dalam proses pencarian, terdapat dua

golongan, yakni *Uninformed Search/Blind Search* dan *Informed Search/Heuristic Search.*



Gambar 2.5. Bagan Metode Pencarian (*Searching*)

Dari Gambar 2.3 bagan metode pencarian/penelusuran dibagi menjadi dua golongan, yakni pencarian buta *(blind search)* dan pencarian terbimbing *(heuristic search)*. [4]

A. Pencarian Buta (*Blind Search*)

Blind Search adalah pencarian solusi tanpa adanya informasi yang dapat mengarahkan pencarian untuk mencapai *goal state* dari *current state* (keadaan sekarang). Informasi yang ada hanyalah definisi *goal state* itu sendiri, sehingga algoritma dapat mengenali *goal state* bila menjumpainya.

Dengan ketiadaan informasi, maka *blind search* dalam kerjanya memeriksa/mengembangkan *node-node* secara tidak terarah dan kurang efisien untuk kebanyakan kasus karena banyaknya *node* yang dikembangkan.

Beberapa contoh algoritma yang termasuk *blind seacrh* antara lain adalah *Breadth First Search, Uniform Cost Search, Depth First Search, Depth Limited Search, Iterative Deepening Search,* dan *Bidirectional Search.* [6]

B. Pencarian Terbimbing (*Heuristic Search*)

Berbeda dengan *blind search*, *heuristic search* mempunyai informasi tentang *cost*/biaya untuk mencapai *goal state* dari *current state.* Dengan informasi tersebut, *heuristic search* dapat melakukan pertimbangan untuk mengembangkan atau memeriksa *node-node* yang mengarah ke *goal state.* Misalnya pada pencarian *rute* pada suatu peta, bila berangkat dari kota A ke kota tujuan B yang letaknya di Utara kota A, dengan *heuristic search,* pencarian akan lebih difokuskan ke arah Utara (dengan informasi *cost* ke *goal*), sehingga secara umum, *heuristic search* lebih efisien daripada *blind search.*

Heuristic search untuk menghitung (perkiraan) *cost* ke *goal state,* digunakan fungsi *heuristic.* Fungsi *heuristic* berbeda daripada algoritma,

dimana *heuristic* lebih merupakan perkiraan untuk membantu algoritma, dan tidak harus valid setiap waktu. Meskipun begitu, semakin bagus fungsi *heuristic* yang dipakai, semakin cepat dan akurat pula solusi yang didapat. Menentukan *heuristic* yang tepat untuk kasus dan implementasi yang ada juga sangat berpengaruh terhadap kinerja algoritma pencarian.

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

Tabel 2.1. Kegunaan *Heuristic search*

Algiritma Pencarian	**Kegunaan**	**Implementasi**
Best First Search	Penelusurannya hanya menggunakan estimasi *cost* / jarak ke node tujuan, h(n) akibatnya pencarian tidak menyeluruh.	Sistem pakar, penjadwalan, pencarian *rute* pada peta geografis dan lain-lain.
Greedy Search	Pada *greedy search* ide utamanya adalah mengembangkan node dengan nilai estimasi biaya ke *goal* yang paling kecil (berarti *node* yang paling dekat ke tujuan).	Masalah penukaran uang, minimisasi waktu di dalam sistem (penjadwalan), memilih beberapa jenis investasi (penanaman modal), memilih jurusan di perguruan tinggi, masalah *knapsnak* dan lainnya.
A* *(A Star) Search*	Menjumlahkan jarak sebenarnya dengan estimasi jaraknya dan pencariannya menyeluruh, tetapi akan memakai memori yang cukup besar untuk menyinpan *node* sebelumnya.	Pencarian jalur terpendek pada peta, pencarian jalur angkutan kota, pencarian jalan pada *game*, menggerakkan karakter *game*, pencarian jalan untuk permainan *Lose Your Marble*, penyelesaian permasalahan 8 *Puzzle* dan lain-lain.

Hill Climbing Search	*Hill climbing* sering digunakan ketika fungsi *heuristic* yang bagus tersedia untuk mengevaluasi *state* tapi ketika tidak ada yang perlu dievaluasi maka fungsi ini tidak ada.	Sistem pakar, pencarian lokasi pada peta, dan lain-lain.

2.2.3 *Best-First Search*

Sesuai dengan namanya, *best-first search* merupakan sebuah metode yang membangkitkan simpul dari sebuah simpul sebelumnya (yang sejauh ini terbaik di antara semua simpul yang pernah dibangkitkan). Penentuan simpul terbaik dilakukan dengan menggunakan sebuah fungsi yang disebut fungsi evaluasi f(n) [5]. Pada *best-first search* jika *successor* digunakan, maka dikatakan algoritma mengembangkan simpul tersebut. Setiap sebuah simpul dikembangkan, algoritma akan menyimpan setiap *successor* simpul *n* sekaligus dengan harga (*cost*) dan petunjuk pendahulunya yang disebut "*parent*". Algoritma akan berakhir pada simpul tujuan, dan tidak ada lagi pengembangan simpul. Fungsi evaluasi pada *best-first search* dapat berupa informasi biaya perkiraan dan suatu simpul menuju ke simpul tujuan atau gabungan antara biaya sebenarnya dan biaya perkiraan tersebut. Biaya perkiraan dapat diperoleh dengan menggunakan suatu fungsi yang disebut fungsi *heuristic*. Pada strategi *best-first search*, *cost* sebenarnya yaitu dari simpul awal ke simpul *n*, dinotasikan dengan *g(n)* dan fungsi *heuristic* yang digunakan yaitu perkiraan/*estimasi* nilai dari simpul *n* ke simpul tujuan dinotasikan dengan *h(n)*. Algoritma yang menggunakan metode *best-first search*, yaitu :

- *Greedy Best-First Search*
- Algoritma A*

Berikut akan diberikan beberapa istilah yang sering digunakan pada metode *best-first search* :

- *Start Node* adalah sebuah terminologi untuk posisi awal sebuah pencarian.
- *Current Nod*e adalah simpul yang sedang dijalankan (yang sekarang) dalam algoritma pencarian jalan terpendek.
- Kandidat (*successor*) adalah simpul-simpul yang berbatasan dengan *current node*, dengan kata lain simpul-simpul yang akan diperiksa berikutnya.
- Simpul (*node*) merupakan representasi dari area pencarian
- *Open list* adalah tempat menyimpan data simpul yang mungkin diakses dari starting node maupun simpul yang sedang dijalankan.
- *Closed list* adalah tempat menyimpan data simpul yang juga merupakan bagian dari jalur terpendek yang telah berhasil didapatkan.
- *Goal node* yaitu simpul tujuan.
- *Parent* adalah current node dari successor/kandidat.

2.2.4 *Greedy Best First Search*

Salah satu algoritma yang termasuk kedalam kategori *heuristic search* adalah *gready best first search* yang dikenal juga dengan *greedy search*. Secara harfiah *greedy* artinya rakus atau tamak, sifat yang berkonotasi negatif. Sesuai dengan arti tersebut, prinsip *greedy* adalah mengambil keputusan yang dianggap

terbaik hanya untuk saat itu saja yang diharapkan dapat memberikan solusi terbaik secara keseluruhan. Oleh karena itu, pada setiap langkah harus dibuat keputusan yang terbaik dalam menentukan pilihan. Keputusan yang telah diambil pada suatu langkah tidak dapat diubah lagi pada langkah selanjutnya.

Greedy Best First Search seperti halnya algoritma yang menggunakan strategi *best-first search* lainnya mempunyai sebuah fungsi yang menjadi acuan kelayakan sebuah simpul yaitu fungsi evaluasi *f(n).*pada *Greedy Best First Search* fungsi evaluasi tidak bergantung pada *cost* sebenarnya, tetapi hanya tergantung pada fungsi *heuristic* itu sendiri. Jika pada algoritma *Dijkstra* pencarian yang dilakukan tergantung pada *cost* sebenarnya dari sebuah simpul yaitu *g(n),*pada *Greedy best first search* fungsi evaluasi hanya tergantung pada fungsi *heuristic h(n)* yang mengestimasikan arah yang benar, sehingga pencarian jalur dapat berlangsung dengan sangat cepat. Secara matematis fungsi evaluasi pada *greedy search* di berikan oleh:

$$f(n) = h(n)$$

Dengan :

g(n) = estimasi biaya dari simpul *n* ke simpul tujuan (*goalnode)*

Berikut langkah-langkah pencarian lintasan terpendek yang dilakukan *Greedy Best-First Search*:

- Masukkan simpul awal ke dalam *open list*.
- *Open* berisi simpul awal dan *closed list* masih kosong.
- Masukkan simpul awal ke *closed list* dan suksesornya pada *open list*.

- Ulangi langkah berikut sampai simpul tujuan ditemukan dan tidak ada lagi simpul yang akan dikembangkan.
  - Hitung nilai *f* simpul-simpul yang ada pada *open list*, ambil simpul terbaik (*f* paling kecil)
  - jika simpul tersebut sama dengan simpul tujuan, maka sukses
  - jika tidak, masukan simpul tersebut ke dalam *closed*
  - Bangkitkan semua *successor* dari simpul tersebut
  - untuk setiap *successor* kerjakan :
    - jika suksesor tersebut belum pernah dibangkitkan, evaluasi *successor* tersebut, tambahkan *open*, dan catat *"parent"*-nya.
    - jika *successor* tersebut sudah pernah dibangkitkan, ubah *parent*-nya jika jalur melalui *parent* ini lebih baik dari jalur melalui *parent* yang sebelumnya. Selanjutnya, perbarui biaya untuk *successor* tersebut.

2.2.5 Algoritma A* (*A Star*)

Algoritma A* adalah algoritma yang menggabungkan algoritma *Djikstra* dan algoritma *Greedy Best First Search*. Selain menghitung biaya yang diperlukan untuk berjalan dari simpul satu ke simpul lainnya, algoritma A* juga menggunakan fungsi heuristic untuk memprioritaskan pemeriksaan simpul-simpul pada arah yang benar, sehingga algoritma A* mempunyai efisiensi waktu yang baik dengan tidak mengorbankan perhitungan biaya sebenarnya.

2.2.5.1 Sejarah Algoritma A* (*A Star*)

Penggunaan informasi *heuristic* untuk meningkatkan efisiensi pencarian telah dipelajari dalam berbagai bidang. Penggunaan fungsi evaluasi pada masalah pencarian pada graft dikemukakan oleh Lin pada tahun 1965 yang digunakan pada masalah *Traveling Salesman Problem* (TSP) [6]. Pada 1966 Doran dan Michie merumuskan dan mencoba dengan algoritma *best-first search* yang menggunakan perkiraan jarak dan *current node* ke simpul tujuan [7]. Hart, Nilsson dan Raphael memperkenalkan penggunaan sebuah algoritma dalam masalah optimasi yaitu algoritma A* (*A star*). Versi *bi-directional* algoritma A*, yang secara bersamaan mencari dari simpul awal dan simpul tujuan diperkenalkan oleh Pohl pada tahun 1971 [8], yang selanjutnya diteliti oleh de Champeaux dan Sint pada tahun 1977 [9].

2.2.5.2 Algoritma A* dalam Pencarian Rute Terpendek

Beberapa perbedaan strategi *best-first search* hanya pada bentuk fungsi evaluasi yang digunakan. Pada strategi *best-first search*, simpul yang terpilih untuk dikembangkan adalah simpul dengan nilai fungsi evaluasi terendah dan ketika dua lintasan mengarah pada simpul yang sama, simpul dengan nilai paling besar akan dibuang, secara matematis.

Pencarian menggunakan algoritma A* mempunyai prinsip yang sama dengan algoritma BFS, hanya saja dengan dua faktor tambahan.

1. Setiap sisi mempunyai "*cost*" yang berbeda-beda, seberapa besar *cost* untuk pergi dari satu simpul ke simpul yang lain.

2. *Cost* dari setiap simpul ke simpul tujuan bisa diperkirakan. Ini membantu pencarian, sehingga lebih kecil kemungkinan 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. Sebagai contoh, bila berkendaraan melewati jalan biasa bisa saja merupakan jarak terdekat, tetapi melewati jalan tol biasanya memakan waktu lebih sedikit.

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

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

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

Cost antara simpul adalah jaraknya, dan perkiraan *cost* dari suatu simpul ke simpul tujuan adalah penjumlahan jarak dari simpul tersebut ke simpul tujuan. Atau agar lebih mudahnya bisa ditunjukkan seperti berikut ini.

$$f(n) = g(n) + h(n) \quad \text{.....[10]}$$

dengan :

$f(n)$ = Fungsi evaluasi

$g(n)$ = Biaya *(cost)* dari keadaan awal (*start node*) sampai keadaan *n*

$h(n)$ = Estimasi biaya dari keadaan *n* sampai tujuan (*goal node*)

Perhatikan bahwa algoritma ini hanya bekerja bila *cost* perkiraan tidak lebih besar dari *cost* yang sebenarnya. Bila *cost* perkiraan lebih besar, bisa jadi jalan yang ditemukan bukanlah yang terpendek. [10]

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.

Algoritma A* menggunakan dua buah *list* yaitu *Open List* dan *Closed List*. Seperti halnya *best-first search* yang lain kedua *list* mempunyai fungsi yang sama. Pada awalnya *Open List* hanya berisi satu simpul yaitu simpul awal dan *Closed List* masih kosong. Perlu diperhatikan setiap simpul akan menyimpan petunjuk "*parent*"nya sehingga setelah pencarian berakhir lintasan juga akan didapatkan. Berikut adalah langkah-langkah algoritma A* (*A star*) :

- Masukkan simpul awal ke *Open List*.
- Ulangi langkah berikut sampai pencarian berakhir.

- Cari *node n* dengan nilai *f(n)* paling rendah, dalam *Open List*. *Node* ini akan menjadi *current node*.
- Keluarkan *current node* dari *Open List* dan masukkan ke *Closed List*.
- Untuk setiap *successor* dari *current node* lakukan langkah berikut :
    1. Jika sudah terdapat dalam *Closed List*, abaikan, jika tidak lanjutkan.
    2. Jika belum ada pada *Open List*, masukkan ke *Open List*. Simpan *current node* sebagai *parent* dari *successor-successor* ini. Simpan *cost* masing-masing simpul.
    3. Jika belum ada dalam *Open List*, periksa jika simpul *successor* ini mempunyai nilai lebih kecil dibanding *successor* sebelumnya. Jika lebih kecil, jadikan sebagai *current node* dan ganti *parent node* ini.
- Walaupun telah mencapai simpul tujuan, jika masih ada *successor* yang memiliki nilai yang lebih kecil, maka simpul tersebut akan terus dipilih sampai bobotnya jauh lebih besar atau mencapai simpul akhir dengan bobot yang lebih kecil dibanding dengan simpul sebelumnya yang telah mencapai simpul tujuan.
- Pada setiap pemilihan simpul berikutnya, nilai *f(n)* akan dievakuasi, dan jika terdapat nilai *f(n)* yang sama maka akan dipilih berdasarkan nilai *g(n)* terbesar.

Metode A* mirip dengan algoritma pencarian graph yang berpotensial mencari daerah yang luas pada sebuah peta. Metode A* mempunyai fungsi

heuristic untuk memandu pencarian ke depan sampai tujuan. Metode A* dapat melakukan *backtracking* jika jalur yang ditempuh ternyata salah. Metode A* dapat melakukannya karena menyimpan jejak / *track* yang mungkin sebagai jalur yang optimal. Sebagai contoh, jika kita sedang menuju suatu kota dan sampai pada persimpangan jalan, dan memutuskan untuk belok kiri daripada ke kanan, dan ternyata bila jalan yang dipilih ternyata salah, kita akan kembali ke persimpangan dan mengambil jalan satunya. Itulah yang dilakukan metode A* ini.

Performansi algoritma A* dapat diketahui dengan melihat perbandingan metode dari Tabel 2.2.

Tabel 2.2. Perbedaan ketiga algoritma

No.	Nama algoritma	Metode	Keterangan
1.	*Optimal Search (Djikstra's Algorithm)*	$g(n_i)=g(n)+c(n,n_i)$	$g(n)$ adalah *cost* dari IS ke *node n* $c(n,n_i)$ adalah *cost* dari *node n* ke n_i
2.	*Best First Search*	$h(n)$	$h(n)$ adalah estimated cost dari jalur terpendek dari node n ke GS
3.	*A**	$f(n)=g(n)+h(n)$	$f(n)$ adalah fungsi *heruistic* $g(n)$ adalah *cost* dari IS ke *node n* $h(n)$ adalah *estimated cost* dari jalur terpendek dari node n ke GS

Dari informasi pada Tabel 2.2 di atas dapat dibuktikan bahwa bila :

a. $\boldsymbol{g(n) = 0}$, maka $\boldsymbol{f(n) = h(n)}$ sehingga algoritma A* akan bertingkah laku sebagaimana *Best First Search*.

b. $\boldsymbol{h(n) = 0}$, maka $\boldsymbol{f(n) = g\ (n)}$ sehingga algoritma A* akan bertingkah laku sebagaimana *Optimal Search (Dijkstra's Algorithm)*.

Dengan demikian dapat diambil kesimpulan bahwa algoritma A* mengkombinasikan kelebihan dari algoritma *Optimal Search* dan *Best First*

Search. Dari informasi ini, dapat menganalisa dan membandingkan *cost* ketiga algoritma tersebut.

2.2.6 *Heuristic Best First Seach*

Fungsi heuristic *h(n)* merupakan estimasi *cost* dari *n* ke simpul tujuan. Sangat penting untuk memilih fungsi *heuristic* yang baik. Misalkan *h*(n)* merupakan *cost* sebenarnya dari simpul *n* ke simpul tujuan, maka pada algoritma A* terdapat kemungkinan yang terjadi pada pemilihan fungsi *heuristic* yang digunakan yaitu (Amit Gaming):

- Jika *h(n)* – 0, sehingga hanya *g(n)* yang terlibat maka A* akan bekerja seperti halnya algoritma *Djikstra*.
- Jika $h(n) \leq h^*(n)$, maka A* akan mengembangkan titik dengan nilai paling rendah dan algoritma A* menjamin ditemukannya lintasan terpendek. Nilai *h(n)* terendah akan membuat algoritma mengembangkan lebih banyak simpul. Jika $h(n) \leq h^*(n)$, maka *h(n)* dikatakan *heuristic* yang *admissible*.
- Jika $h(n) = h^*(n)$, maka A* akan mengikuti lintasan terbaik dan tidak akan mengembangkan titik-titik yang lain sehingga akan berjalan cepat. Tetapi hal ini tidak akan terjadi pada semua kasus. Informasi yang baik akan mempercepat kinerja A*.
- Jika $h(n) \geq h^*(n)$, maka A* tidak menjamin pencarian rute terpendek, tetapi berjalan dengan cepat.
- Jika *h(n)* terlalu tinggi relative dengan *g(n)* sehingga hanya *h(n)* yang bekerja maka A* berubah jadi *Greedy Best First Search*.

Berikut beberapa *heuristic* yang biasa digunakan yaitu :

2.2.6.1 Manhattan Distance

Manhattan distance atau sering disebut *Taxicab Geometry*, *city block distance,* diperkenalkan oleh Hermann Minkowski pada abad ke-19. *Manhattan distance* merupakan *heuristic* standar. Fungsi *heuristic* ini digunakan untuk kasus dengan pergerakan pada peta hanya lurus x,y (horisontal atau vertikal), tidak diperbolehkan pergerakan diagonal. *Manhattan distance* antara dua *vektor p,q* pada sebuah dimensi *n* adalah penjumlahan panjang proyeksi garis antara dua objek. Secara formal perhitungan nilai *heuristic* untuk simpul ke-*n* menggunakan *Manhattan distance* adalah sebagai berikut :

$$d(p,q) = |\mathrm{p} - \mathrm{q}| = \sum_{i=1}^{n} |p_i - q_i|$$

Pada kasus dua dimensi dan pada peta geografis *Manhattan Distance* adalah sebagai berikut :

$$h(n) = (abs(n.x - tujuan.x) + abs(n.y - tujuan.y))$$

Dengan,

h(n) = nilai *heuristic* untuk simpul *n*

n.x = nilai koordinat *x* dari simpul *n*

n.y = nilai koordinat *y* dari simpul *n*

x-tujuan = nilai koordinat *x* dari simpul tujuan

y-tujuan = nilai koordinat *y* dari simpul tujuan

2.2.6.2 *Euclidean Distance*

Euclidean distance didefinisikan sebagai panjang dari garis lurus yang menghubungkan posisi dua buah objek. Secara logis diketahui bahwa jarak terpendek antara dua titik adalah garis lurus antara kedua titik tersebut. *Euclidean distance* digunakan jika proses dapat bergerak ke segala arah. *Heuristic* ini akan menghitung jarak berdasarkan panjang garis yang dapat ditarik dari dua buah titik. Perhitungannya dapat ditulis sebagai berikut:

$$h(n) = \sqrt{(n.x - tujuan.x)^2 + (n.y - tujuan.y)^2}$$

Dalam kasus ini, skala relatif nilai *g* mungkin akan tidak sesuai lagi dengan nilai fungsi *heuristic h*. Karena *Euclidean distance* selalu lebih pendek dari *Manhattan distance*, maka dapat dipastikan selalu akan didapatkan jalur terpendek, walaupun secara komputasi lebih berat.

Dalam beberapa literatur juga disebutkan jika nilai *g* adalah 0, maka lebih baik jika ongkos komputasi operasi pengakaran pada *heuristic Euclidean distance* dihilangkan saja, menghasilkan rumus sebagi berikut:

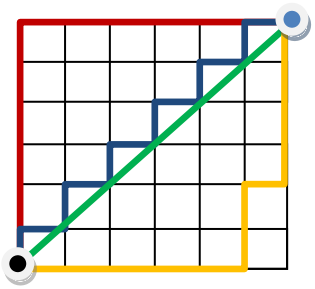
$$h(n) = (n.x\text{-}tujuan.x)^2 + (n.y\text{-}tujuan.y)^2$$

Hal lain yang harus diperhatikan adalah seberapa cepat fungsi *heuristic* dapat dikomputasi. Selalu akan ada *tradeoff* antara akurasi dari fungsi *heuristic* dan waktu yang dibutuhkannya untuk mengomputasinya. Nampaknya bagus jika fungsi *heuristic* yang digunakan sangat akurat, dilihat dari berbagai macam percobaan bahwa jika *heuristic* yang digunakan sempurna maka A* akan selalu melewati jalur yang tepat dan akan selalu memberikan optimum global. Namun, *heuristic* yang sempurna semacam itu tidak ada (dan tidak akan pernah ada), dan

bahkan untuk mendekatinya saja akan memerlukan tambahan komputasi yang tidak ringan. Seringkali dalam aplikasinya *heuristic* yang memberikan hasil yang sangat akurat namun lambat, kurang disenangi dibanding *heuristic* yang tidak begitu optimal namun memberikan hasil dengan cepat. Maka dari itu, pemilihan *heuristic* sangat bergantung pada tujuan penggunaan A*. Jika hasil yang dibutuhkan adalah optimum global, maka fungsi *heuristic* yang digunakannya haruslah "sempurna", sedang jika yang dibutuhkan adalah hasil yang cepat dan tidak harus jalur terpendek, maka lebih bijak menggunakan *heuristic* yang lebih ringan.

Euclidean distance bertujuan untuk memprioritaskan *node-node* yang berada dekat garis lurus antara simpul awal dan simpul tujuan. Pendekatan ini dapat sangat membantu algoritma A* karena nilainya yang tidak pernah akan melebihi nilai sebenarnya. Namun pendekatan ini dapat tidak berpengaruh ataupun malah memperlambat kinerja algoritma A*.

Karena *Euclidean distance* lebih pendek dari *Manhattan distance* atau *diagonal distance* akan didapat lintasan terpendek, namun waktu yang dibutuhkan akan bertambah.



Gambar 2.6 *Euclidean Distance* dan *Manhattan Distance*

Gambar 2.4 memperlihatkan pendekatan *Euclidean distance* dan *Manhattan distance*. Merah, biru dan kuning mempunyai nilai yang sama yaitu 12 merupakan hasil *Manhattan distance* untuk rute yang sama dengan *Eucliden distance*, warna hijau yang mempunyai panjang 6 x $\sqrt{2} \approx 8{,}48$ yang lebih kecil dari *Manhattan distance*.

2.2.7 *Collision Detection*

Collision detection adalah komponen penting pada sebuah *game*. disadari atau tidak, komponen ini sering kali membuat *game* menjadi realistik, ada beberapa teknik collision detection, mulai dari *bounding box*, *reduce bounding box*, *multi bounding box*, *circle*, sampai penggunaan sudut dalam pendeteksian tabrakan.

Bounding box, meski memiliki algoritma yang super cepat, dan metode penyimpanan data yang tidak terlalu susah, namun *bounding box* memiliki kelemahan yang pokok, yakni menganggap area kosong sebagai *solid area*. hal ini memberikan efek kesalahan deteksi *collision*. Kesalahan ini dikurangi dengan menggunakan *reduce bounding box*, namun sayang penggunaan *reduce bounding box* juga masih menyebabkan permasalahan yang hampir sama.

Penggunaan multi *bounding box* dirasa cukup bagus, di mana setiap area tubuh diberikan *bounding box*. seperti tangan, kaki, kepala, dan badan diberikan *bounding box*-nya masing-masing, cara ini pasti lebih lama dari *bounding box*, namun dirasa cukup mengurangi area kosong yang dianggap objek.

Cara lain selain *bounding box* adalah penggunaan lingkaran, penggunaan lingkaran memiliki kelebihan *bound* memiliki jarak yang sama dengan pusat, sehingga dengan memanfaatkan sifat ini dapat dibuat algoritma yang lebih cepat dari deteksi *bounding box*, namun sayang algoritma ini masih kurang bagus dalam mengatasi area kosong.

Penggunaan cara lain adalah dengan pendeteksian garis yang berpotongan, dan penggunaan sudut. untuk cara yang ini, objek akan pertama kali di*list* area *edge* terluar, untuk membatasi area badan, dan dunia luar.

2.3 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 seseorang yang memiliki sebuah departemen

yang memiliki manager, sekretaris, petugas administrasi data dan lainnya. Misal manager tersebut ingin memperoleh data dari bag 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 adminiistrasi. Jadi untuk menyelesaikan suatu masalah dengan kolaborasi antar objek-objek yang ada karena setiap objek memiliki deskripsi tugasnya sendiri.[11]

Pemrograman orientasi-objek menekankan konsep berikut:

- **Kelas (*Class*)** — 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 objek. 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.

- **Objek (*Object*)** - membungkus data dan fungsi bersama menjadi suatu unit dalam sebuah program komputer. Objek merupakan dasar dari modularitas dan struktur dalam sebuah program komputer berorientasi objek.
- **Abstraksi (*Abstract*)** - 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.
- **Enkapsulasi (*Encapsulation*)** - Memastikan pengguna sebuah objek tidak dapat mengganti keadaan dalam dari sebuah objek dengan cara yang tidak layak; hanya metode dalam objek tersebut yang diberi ijin untuk mengakses keadaannya. Setiap objek mengakses *interface* yang menyebutkan bagaimana objek lainnya dapat berinteraksi dengannya. Objek lainnya tidak akan mengetahui dan tergantung kepada representasi dalam objek tersebut.
- **Polimorfisme (*Polimorfism*)** melalui pengiriman pesan. Tidak bergantung kepada pemanggilan subrutin, bahasa orientasi objek dapat mengirim pesan; metode tertentu yang berhubungan dengan sebuah pengiriman pesan tergantung kepada objek tertentu di mana pesa tersebut dikirim. Contohnya, bila sebuah burung menerima pesan "gerak cepat", dia akan

menggerakan sayapnya dan terbang. Bila seekor singa menerima pesan yang sama, dia akan menggerakkan kakinya dan berlari. Keduanya menjawab sebuah pesan yang sama, namun yang sesuai dengan kemampuan hewan tersebut. Ini disebut polimorfisme karena sebuah variabel tunggal dalam program dapat memegang berbagai jenis objek yang berbeda selagi program berjalan, dan teks program yang sama dapat memanggil beberapa metode yang berbeda di saat yang berbeda dalam pemanggilan yang sama. Hal ini berlawanan dengan bahasa fungsional yang mencapai polimorfisme melalui penggunaan fungsi kelas-pertama.

- **Inheritas (*Inheritance*)**- Mengatur polimorfisme dan enkapsulasi dengan mengijinkan objek didefinisikan dan diciptakan dengan jenis khusus dari objek yang sudah ada - objek-objek ini dapat membagi (dan memperluas) perilaku mereka tanpa harus mengimplementasi ulang perilaku tersebut (bahasa berbasis-objek tidak selalu memiliki inheritas).

2.4 *Tools* yang Digunakan

2.4.1 *UML* (*Unified Modeling Language*)

UML (*Unified Modeling Language*) adalah bahasa spesifikasi standar untuk mendokumentasikan, menspesifikasikan, dan membangun system (Flowler, 2006). *Unified Modeling Language* (*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.[12]

UML adalah suatu bahasa yang digunakan untuk menentukan, memvisualisasikan, membangun, dan mendokumentasikan suatu sistem informasi. *UML* dikembangkan sebagai suatu alat untuk analisis dan desain berorientasi objek oleh Grady Booch, Jim Rumbaugh, dan Ivar Jacobson. Namun demikian *UML* dapat digunakan untuk memahami dan mendokumentasikan setiap sistem informasi. Penggunaan *UML* dalam industri terus meningkat. Ini merupakan standar terbuka yang menjadikannya sebagai bahasa pemodelan yang umum dalam industri peranti lunak dan pengembangan system.

UML menyediakan 10 macam diagram untuk memodelkan aplikasi berorientasi objek, yaitu :

1. ***Use Case Diagram*** untuk memodelkan proses bisnis.
2. ***Conceptual Diagram*** untuk memodelkan konsep-konsep yang ada di dalam aplikasi.
3. ***Sequence Diagram*** untuk memodelkan pengiriman pesan (*message*) antar *objects*.
4. ***Collaboration Diagram*** untuk memodelkan interaksi antar *objects*.
5. ***State Diagram*** untuk memodelkan perilaku *objects* di dalam sistem.

6. ***Activity Diagram*** untuk memodelkan perilaku *Use Cases* dan *objects* di dalam *system*.
7. ***Class Diagram*** untuk memodelkan struktur kelas.
8. ***Object Diagram*** untuk memodelkan struktur *object*.
9. ***Component Diagram*** untuk memodelkan komponen *object*.
10. ***Deployment Diagram*** untuk memodelkan distribusi aplikasi.

2.4.2 *Flash*

Flash merupakan *software* yang memiliki kemampuan menggambar sekaligus menganimasikannya, serta mudah dipelajari.[13] *Flash* tidak hanya digunakan dalam pembuatan animasi, tetapi pada zaman sekarang ini *flash* juga banyak digunakan untuk keperluan lainnya seperti dalam pembuatan *game,* presentasi, membangun web, animasi pembelajaran, bahkan juga dalam pembuatan film.

Animasi yang dihasilkan *flash* adalah animasi berupa *file movie*. *Movie* yang dihasilkan dapat berupa grafik atau teks. Grafik yang dimaksud disini adalah grafik yang berbasis vektor, sehingga saat diakses melalui internet, animasi akan ditampilkan lebih cepat dan terlihat halus. Selain itu *flash* juga memiliki kemampuan untuk mengimpor *file* suara, video maupun *file* gambar dari aplikasi lain.

Flash adalah program grafis yang diproduksi oleh *Macromedia corp*, yaitu sebuah *vendor software* yang bergerak dibidang animasi web. *Macromedia Flash* pertama kali diproduksi pada tahun 1996. *Macromedia flash* telah diproduksi

dalam beberapa versi. Versi terakhir dari *Macromedia Flash* adalah *Macromedia flash* 8. Sekarang *Flash* telah berpindah *vendor* menjadi Adobe.

Adobe adalah *vendor software* yang membeli *Flash* dari *vendor* sebelumnya yaitu *Macromedia*. Sejak itu, *Macromedia Flash* berganti nama menjadi *Adobe Flash*. Versi terbaru dari *Adobe Flash* adalah *Adobe Flash* CS5 *Professional*. Dalam pembuatan animasi ini penulis sudah menggunakan *Adobe Flash* CS5 *Professional* sebagai aplikasinya.

2.4.2.1 *Adobe Flash* CS5

Adobe Flash CS5 adalah salah satu aplikasi pembuat animasi yang cukup dikenal saat ini. Berbagai fitur dan kemudahan yang dimiliki menyebabkan *Adobe Flash* CS5 menjadi program animasi favorit dan cukup populer. Tampilan, fungsi dan pilihan palet yang beragam, serta kumpulan *tool* yang sangat lengkap sangat membantu dalam pembuatan karya animasi yang menarik.

Flash seperti *software* gado-gado dimana didalamnya terdapat semua kelengkapan yang dibutuhkan. Mulai dari fitur menggambar, ilustrasi, mewarnai, animasi, dan *programming*. Kita dapat mendesain gambar atau objek yang akan kita animasikan langsung pada *Flash*. Fitur programming pada *Flash* menggunakan bahasa *ActionScript*.

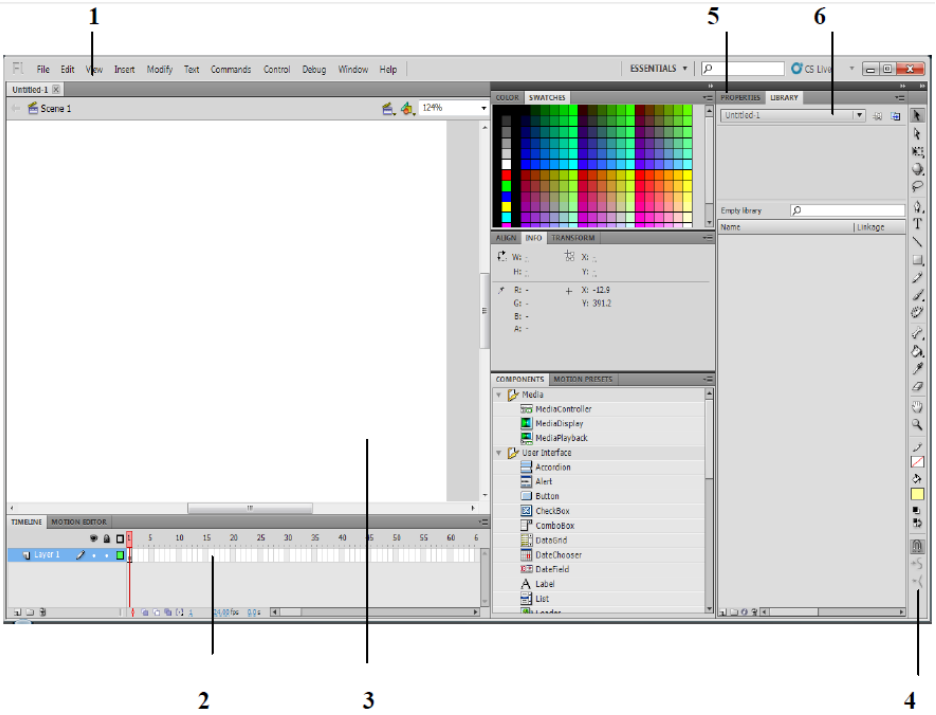
ActionScript dibutuhkan untuk memberi efek gerak dalam animasi. *ActionScript* di *flash* pada awalnya memang sulit dimengerti jika seseorang tidak mempunyai dasar atau mengenal *flash*. Tetapi jika sudah mengenalnya, kita tidak bisa lepas dari *ActionScript* karena sangat menyenangkan dan dapat membuat pekerjaan jauh lebih cepat dan mudah.



Gambar 2.7 Tampilan *Start Page Adobe Flash* CS5

Jendela utama merupakan awal dari pembuatan program, pembuatannya dilakukan dalam kotak *movie* dan *stage* yang didukung oleh *tools* lainnya. Seperti yang pernah dijelaskan dalam sebuah tulisan "Jendela kerja *flash* terdiri dari panggung (*stage*) dan panel-panel. Panggung merupakan tempat objek diletakkan, tempat menggambar dan menganimasikan objek. Sedangkan panel disediakan untuk membuat gambar, mengedit gambar, menganimasi, dan pengeditan lainnya". [14]

Berikut ini adalah bentuk tampilan jendela utama pada *Adobe Flash* CS 5.



Gambar 2.8 Jendela Utama *Adobe Flash* CS5

Keterangan gambar :

1. ***Menu Bar*** adalah kumpulan yang terdiri atas dasar menu-menu yang digolongkan dalam satu kategori. Misalnya menu *file* terdiri atas perintah *New*, *Open*, *Save*, *Import*, *Export*, dan lain-lain.
2. ***Timeline*** adalah sebuah jendela *panel* yang digunakan untuk mengelompokkan dan mengatur isi sebuah *movie*, pengaturan tersebut meliputi penentuan masa tayang objek, pengaturan *layer*, dan lain-lain.
3. ***Stage*** adalah area untuk berkreasi dalam membuat animasi yang digunakan untuk mengkomposisi *frame-frame* secara individual dalam sebuah *movie*.
4. ***Toolbox*** adalah kumpulan *tools* yang sering digunakan untuk melakukan seleksi, menggambar, mewarnai objek, memodifikasi objek, dan mengatur gambar atau objek.

5. ***Properties*** adalah informasi objek-objek yang ada di *stage*. Tampilan panel *properties* secara otomatis dapat berganti-ganti dalam menampilkan informasi atribut-atribut *properties* dari objek yang terpilih.
6. ***Panels*** adalah sebagai pengontrol yang berfungsi untuk mengganti dan memodifikasi berbagai atribut dari objek dari animasi secara cepat dan mudah.

2.4.2.2 *Toolbox*

Fasilitas *Toolbox* seperti telah dijelaskan sekilas diawal adalah sekumpulan *tool* atau alat yang mempunyai fungsi-fungsi tersendiri untuk keperluan desain. Berikut penjelasan setiap *tool* yang terdapat pada *Toolbox*.[13]

1. ***Arrow Tool***

   Arrow Tool atau sering disebut *selection tool* berfungsi untuk memilih atau menyeleksi suatu objek.

2. ***Subselection Tool***

   Subselection Tool berfungsi menyeleksi bagian objek lebih detail dari pada *selection tool*.

3. ***Free Transform Tool***

   Free Transform Tool berfungsi untuk mentransformasi objek yang terseleksi.

4. ***Gradient Transform Tool***

   Gradien Transform Tool berfungsi untuk mentransformasi warna dari *fill* objek yang terseleksi.

5. ***Lasso Tool***

   Lasso Tool digunakan untuk melakukan seleksi dengan menggambar sebuah garis seleksi.

6. ***Pen Tool***

   Pen Tool digunakan untuk menggambar garis dengan bantuan titik-titik bantu seperti dalam pembuatan garis, kurva atau gambar.

7. ***Text Tool***

   Text Tool digunakan untuk membuat objek teks

8. ***Line Tool***

   Line Tool digunakan untuk membuat atau menggambar garis.

9. ***Rectangle Tool***

   Rectangle Tool digunakan untuk menggambar bentuk bentuk persegi panjang atau bujur sangkar.

10. ***Oval Tool***

    Oval Tool digunakan untuk membuat bentuk bulat atau oval.

11. ***Poly Star Tool***

    Poly Star Tool digunakan untuk menggambar bentuk dengan jumlah segi yang diinginkan.

12. ***Pencil Tool***

    Pencil Tool digunakan untuk membuat garis

13. ***Brush Tool***

    Brush Tool digunakan untuk menggambar bentuk garis-garis dan bentuk-bentuk bebas.

14. *Ink bottle*

Ink Bottle digunakan untuk mengubah warna garis, lebar garis, dan *style* garis atau garis luar sebuah bentuk.

15. *Paintbucket Tool*

Paintbucket Tool digunakan untuk mengisi area-area kosong atau digunakan untuk mengubah warna area sebuah objek yang telah diwarnai.

16. Eraser Tool

Eraser Tool digunakan untuk menghapus objek

17. *Hand Tool*

Hand Tool digunakan untuk menggeser tampilan *stage* tanpa mengubah pembesaran.

18. Zoom Tool

Zoom Tool digunakan untuk memperbesar atau memperkecil tampilan *stage.*

19. *Stroke Color*

Stroke Color digunakan untuk memilih atau memberi warna pada suatu garis.

20. *Fill Color*

Fill Color digunakan untuk memilih atau memberi warna pada suatu objek.

21. *Black and white*

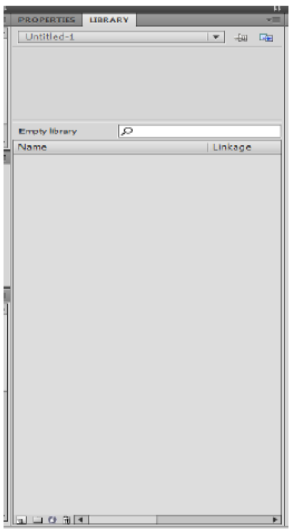
Black and White digunakan untuk memilih warna hitam dan putih saja.

22. *Swap Color*

Swap Color digunakan untuk menukar warna *fill* dan *stroke* atau sebaliknya dari suatu gambar atau objek.

2.4.2.3 *Library*

Fungsi dari *library* adalah sebagai wadah untuk menyimpan program-program terpisah yang sudah jadi, seperti tombol, objek grafis, audio, video, dan lain-lain. Berikut tampilan panel *library*.



Gambar 2.9 *Panel Library*

2.4.2.4 *ActionScript*

Salah satu kelebihan *Adobe Flash* CS5 dibanding perangkat lunak animasi yang lain yaitu adanya *ActionScrip*t. *ActionScript* adalah bahasa pemrograman *Adobe Flash* CS5 yang digunakan untuk membuat animasi atau interaksi.[15] *ActionScript* mengizinkan untuk membuat intruksi berorientasi *action* (lakukan perintah) dan instruksi berorientasi *logic* (analisis masalah sebelum melakukan perintah).

Sama dengan bahasa pemrograman yang lain, *ActionScript* berisi banyak elemen yang berbeda serta strukturnya sendiri. Kita harus merangkainya dengan benar agar *ActionScript* dapat menjalankan dokumen sesuai dengan keinginan. Jika tidak merangkai semuanya dengan benar, maka hasil yang didapat kan akan

berbeda atau *file flash* tidak akan bekerja sama sekali. *ActionScript* juga dapat diterapkan untuk *action* pada *frame*, tombol, *movie clip*, dan lain-lain. *Action frame* adalah *action* yang diterapkan pada *frame* untuk mengontrol navigasi *movie*, *frame*, atau objek lain-lain.[16]

Salah satu fungsi *ActionScript* adalah memberikan sebuah konektivitas terhadap sebuah objek, yaitu dengan menuliskan perintah-perintah didalamnya. Tiga hal yang harus diperhatikan dalam *ActionScrip*t yaitu:

1. *Event*

   Event merupakan peristiwa atau kejadian untuk mendapatkan aksi sebuah objek. *Event* pada Adobe Flash CS 5 ada empat yaitu:

   a. *Mouse event*

      Event yang berkaitan dngan penggunaan *mouse*.

   b. *Keyboard Event*

      Kejadian pada saat menekan tombol *keyboard*.

   c. *Frame Event*

      Event yang diletakkan pada *keyframe*.

   d. *Movie Clip Event*

      Event yang disertakan pada *movie clip*.

2. *Target*

   Target adalah objek yang dikenai aksi atau perintah. Sebelum dikenai aksi atau perintah, sebuah objek harus dikonversi menjadi sebuah simbol dan memiliki nama instan. Penulisan nama target pada skrip harus menggunakan tanda petik ganda (” ”).

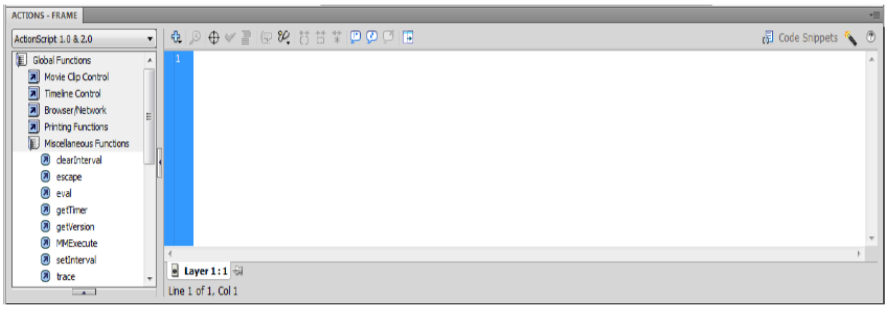
3. *Action*

Pemberian *action* merupakan lagkah terakhir dalam pembuatan interaksi antarobjek. *Action* dibagi menjadi dua antara lain:

a. *Action Frame*: adalah *action* yang diberikan pada *keyframe*. Sebuah *keyframe* akan ditandai dengan huruf a bila pada *keyframe* tersebut terdapat sebuah *action*.

b. *Action* Objek: adalah *action* yang diberikan pada sebuah objek, baik berupa tombol maupun *movie clip*.

ActionScript diketikkan pada panel *actions* yang tersedia pada *software*. *ActionScript* hanya dapat dituliskan pada objek yang bertipe *Movie Clip*, *keyframe*, *Button*, dan objek *components*. *ActionScript* tidak dapat digunakan pada objek tulisan atau gambar lain yang bukan bertipe *Movie Clip*. Jadi bila ingin menggunakan *ActionScript* pada suatu objek, objek tersebut harus diubah menjadi *Movie Clip* terebih dahulu.Untuk membuka *Panel Actions*, klik tulisan *Action* yang ada pada jendela panel. Berikut tampilan dari panel *ActionScript* pada *Adobe Flash* CS5.



Gambar 2.10 *Panel Action*