BAB 2
LANDASAN TEORI

2.1 Cerita Fabel

Cerita fabel adalah cerita yang disukai oleh semua kalangan di Indonesia karena efektif dan berfungsi sebagai bahan pelajaran moral yang baik untuk anak-anak serta rambu-rambu pengingat untuk kalangan diatasnya (remaja dan dewasa). Karena itulah fabel menjadi jenis cerita paling tepat untuk anak-anak. Selain menghibur untuk usia anak, anak-anak juga mendapat manfaat ekstra dari fabel, yakni memahami nilai-nilai moral. Saat mendengar atau membaca fabel, sejatinya anak belajar mengenai nilai-nilai kehidupan secara menyenangkan, mendidik sekaligus menghibur, ini merupakan kelebihan dari fabel.

2.1.1 Cerita Si Kancil dan Buaya

Sudah menjadi rahasia umum di hutan bahwa kancil merupakan hewan paling cerdik. Akalnya seribu untuk mengatasi berbagai macam masalah. Banyak hewan di dalam hutan meminta pertolongan padanya ketika para hewan terlibat sejumlah masalah. Walaupun, dinilai sebagai hewan paling cerdik, namun kancil tidaklah sombong sehingga ia memiliki banyak teman.

Suatu hari Si Kancil sedang berjalan-jalan di pinggir hutan. Dia hanya ingin mencari udara segar, melihat matahari yang cerah bersinar. Di dalam hutan terlalu gelap, karena pohon-pohon sangat lebat dan tajuknya menutupi lantai hutan. Dia ingin berjemur di bawah terik matahari. Di situ ada sungai besar yang airnya dalam sekali. Setelah sekian lama berjemur, Si Kancil merasa bahwa ada yang berbunyi di perutnya, rupanya perutnya sudah lapar. Dia membayangkan betapa enaknya kalau ada makanan kesukaannya, ketimun. Namun kebun ketimun ada di seberang sungai, bagaimana cara menyeberanginya, dia berfikir sejenak. Tiba-tiba dia meloncat kegirangan, dan berteriak “Buaya, buaya, ayo keluar Aku punya makanan untukmu”. Begitu Kancil berteriak kepada buaya-buaya yang banyak tinggal di sugai yang dalam itu.

Sekali lagi Kancil berteriak, “Buaya, buaya ayo keluar mau daging segar tidak ?”, tak lama kemudian, seekor buaya muncul dari dalam air, “siapa yang

teriak-teriak siang-siang begini mengganggu tidurku saja.” “Hei Kancil, diam kau kalau tidak aku makan nanti kamu.” Kata buaya kedua yang juga muncul. “Wah bagus kalian mau keluar, mana yang lain?” kata Kancil kemudian. “Kalau cuma dua ekor masih sisa banyak nanti makanan ini, ayo keluar semuaaa” Kancil berteriak lagi. “Ada apa Kancil sebenarnya, ayo cepat katakan,” kata buaya. “Begini, maaf kalau aku mengganggu tidurmu, tapi aku akan bagi-bagi daging segar buat buaya-buaya di sungai ini,” oleh karena itu buaya yang ada disungai ini harus keluar semua.

Mendengar bahwa buaya akan dibagikan daging segar, buaya-buaya itu segera memanggil teman-temannya untuk keluar. “Hei, teman-teman semua, mau makan gratis tidak ? Ayo kita keluar” buaya pemimpin berteriak memberikan komando. Tak berapa lama, bermunculanlah buaya-buaya dari dalam air. “Nah, sekarang aku harus menghitung dulu ada berapa buaya yang datang, ayo kalian para buaya baris berjajar hingga ke tepi sungai di sebelah sana,” “Nanti aku akan menghitung kalian satu persatu.”

Tanpa berpikir panjang, buaya-buaya itu segera mengambil posisi, berbaris berjajar dari tepi sungai satu ke tepi sungai lainnya, sehingga membentuk seperti jembatan. “Oke, sekarang aku akan mulai menghitung,” kata Kancil yang segera melompat ke punggung buaya pertama, sambil berteriak, “Satu, dua, tiga” begitu seterusnya sambil terus meloncat dari punggung buaya satu ke buaya lainnya. Hingga akhirnya dia sampai di seberang sungai. Hatinya tertawa, “Mudah sekali ternyata.”

Begitu sampai di seberang sungai, Kancil berkata pada buaya, “Hai buaya bodoh, sebetulnya tidak ada daging segar yang akan aku bagikan. Tidakkah kau lihat bahwa aku tidak membawa sepotong daging pun?” “Sebenarnya aku hanya ingin menyeberang sungai ini, dan aku butuh jembatan untuk lewat. Kalau begitu saya ucapkan terima kasih pada kalian, dan mohon maaf kalau aku mengerjai kalian,” kata Kancil.

“Ha, sialan Kancil nakal, ternyata kita cuma dibohongi. Awas kamu ya, kalau ketemu lagi saya makan kamu,” kata buaya-buaya itu geram.

Si Kancil segera berlari menghilang di balik pohon, menuju kebun Pak Tani untuk mencari ketimun **[6]**.



Gambar 2.1 Cerita Si Kancil dan Buaya

2.2 Daya Serap Anak Terhadap Sastra Berdasarkan Usia

Seorang anak mempunyai respon atau tanggapan yang berbeda-beda terhadap sastra. Dalam menanggapi sebuah bacaan sastra yang didengar atau dibacanya, masing-masing anak mempunyai cara tersendiri dalam mengungkapkan kesenangan, pikiran, dan perasaannya. Setiap tanggapan terhadap sastra memang bersifat personal dan khas untuk masing-masing anak, namun demikian setiap tanggapan itu dapat merefleksikan umur dan pengalamannya.

Anak umur 5 tahun seringkali melibatkan diri secara total dalam sebuah sastra. Anak umur 7 – 8 tahun dapat menunjukkan kemampuannya untuk berbagi temuan terhadap cerita yang didengar/dibacanya. Anak umur 9 –10 tahun sudah memiliki kesenangan tertentu terhadap cerita, misalnya dalam memilih tokoh yang disenangi ataupun tidak disenangi ataupun dalam memilih buku yang akan dibacanya. Sedangkan anak umur 11 – 12 tahun sudah berhasil menggeneralisasi tema yang diambil dari sebuah cerita dan dapat mendiskusikan tujuan pengarang. Hal ini menunjukkan bahwa anak yang lebih dewasa dapat menangkap ide-ide cerita yang abstrak. Uraian tentang tanggapan anak-anak terhadap sastra di atas dapat memberi petunjuk kepada guru dan pustakawan dalam memilih dan menyediakan buku-buku bacaan bagi anak.

Menurut Piaget, periode Sensorimotor merupakan periode awal perkembangan kognisi yang ditandai oleh bayi belajar untuk berjalan sekitar umur 2 tahun. Anak belajar selama periode ini melalui pengkoordinasian persepsi sensori dan kegiatan motorik. Pada usia 1,5 – 2 tahun anak senang dengan berbagai macam tindakan atau rima permainan. Anak sedikit sekali memperhatikan kata-kata. Anak pada periode praoperasional (2 – 7 tahun) belajar menyatakan dunianya secara simbolik melalui bahasa, permainan, dan gambar. Berpikirnya masih egosentris dan didasarkan pada persepsi dan pengalaman langsung. Pada usia ini anak sudah mampu mengembangkan rangkaian cerita. Anak sudah mampu memahami struktur cerita rakyat berdasarkan hubungan tiga peristiwa dengan tanjakan laku (rising action). Anak sudah mampu mengantisipasi klimaks cerita.

Pada periode operasi kongkret (7 – 11 tahun), tanggapan anak terhadap sastra berubah. Karakteristiknya ditandai oleh pikiran yang fleksibel. Anak-anak sudah mampu melihat struktur sebuah buku, misalnya kisah dalam kisah, alur sorot balik, dan mampu mengidentifikasi berbagai sudut pandang cerita. Periode terakhir adalah operasi formal (11 tahun ke atas), yakni anak sudah mampu berpikir abstrak, bernalar dari hipotesis ke simpulan logis. Anak dapat menangkap rangkaian alur atau subalur dalam rangkaian pikirannya **[19]**.

2.3 Multimedia

Multimedia adalah penggunaan komputer untuk menyajikan dan menggabungkan teks, suara, gambar, animasi dan *video* dengan alat bantu (*tool*) dan koneksi (*link*) sehingga pengguna dapat bernavigasi, berinteraksi, berkarya dan berkomunikasi **[9]**. Multimedia sering digunakan dalam dunia hiburan. Selain dari dunia hiburan, multimedia juga diadopsi oleh dunia *game*.

Multimedia juga dapat diartikan sebagai penggunaan beberapa media yang berbeda dalam menyampaikan informasi berbentuk teks, suara, grafik, animasi dan *video*.

2.3.1 Definisi Multimedia menurut beberapa ahli [11] :

1. Rosch : Multimedia adalah kombinasi dari komputer dan video.

2. McComick : Multimedia adalah kombinasi dari tiga elemen: suara, gambar, dan teks.
3. Turban dan kawan-kawan : Multimedia adalah Kombinasi dari paling sedikit dua media *input* atau *output*. Media ini dapat berupa *audio* (suara, musik), animasi, *video*, teks, grafik dan gambar
4. Robin dan Linda : Multimedia adalah Alat yang dapat menciptakan presentasi yang dinamis dan interaktif yang mengkombinasikan teks, grafik, animasi, *audio* dan *video*
5. Steinmetz : Multimedia adalah gabungan dari seminimalnya sebuah media diskrit dan sebuah media kontinu. Media diskrit adalah sebuah media dimana validitas datanya tidak tergantung dari kondisi waktu, termasuk didalamnya teks dan grafik. Sedangkan yang dimaksud dengan media kontinu adalah sebuah media dimana validitas datanya tergantung dari kondisi waktu, termasuk di dalamnya suara dan *video*.
6. Vaughan : Multimedia adalah beberapa kombinasi dari teks, gambar, suara, animasi dan *video* dikirim ke anda melalui komputer atau alat elektronik lainnya atau dengan manipulasi digital.

2.3.2 Kategori Multimedia

Multimedia dapat di definisikan menjadi 2 kategori, yaitu *Multimedia Content Production* dan *Multimedia Communication*.

1. *Multimedia Content Production* adalah penggunaan beberapa media (*text, audio, graphics, animation, video dan interactivity*) yang berbeda dalam menyampaikan suatu informasi atau menghasilkan produk multimedia seperti *video, audio, musik, film, game, entertaintment*, dll. Bisa juga dikatakan sebagai penggunaan beberapa teknologi yang berbeda yang memungkinkan untuk menggabungkan media *(text, audio, graphics, animation, video, dan interactivity*) dengan cara yang baru untuk tujuan komunikasi. Dalam kategori ini media yang digunakan adalah :
   a. Media teks/tulisan
   b. Media *audio*/suara

c. Media *video*
d. Media animasi
e. Media gambar
f. Media Interaktif
g. Media *special effect*

2. *Multimedia Communication* adalah penggunaan media (massa), seperti televisi, radio, media cetak dan internet untuk mempublikasikan atau menyiarkan dan mengkomunikasikan material periklanan, publikasi, entertaintment, berita, pendidikan, dan lian - lain. Dalam kategori ini media yang digunakan adalah :
   a. TV
   b. Radio
   c. Film
   d. Media Cetak
   e. Musik
   f. *Game*
   g. *Entertainment*
   h. Tutorial
   i. Internet

2.3.3 Elemen Multimedia

Menurut Hofstetter **[9]** komponen multimedia terbagi atas lima jenis yaitu:

1. Teks

Teks merupakan elemen multimedia yang menjadi dasar untuk menyampaikan informasi, karena teks adalah jenis data yang paling sederhana dan membutuhkan tempat penyimpanan yang paling kecil. Teks merupakan cara yang paling efektif dalam mengemukakan ide-ide kepada pengguna, sehingga penyampaian informasi akan lebih mudah dimengerti oleh masyarakat. Jenis-jenis teks seperti *Printed Text*, yaitu teks yang dihasilkan oleh *word processor* atau *word editor* dengan cara diketik yang nantinya dapat dicetak. *Scanned Text* yaitu

teks yang dihasilkan melalui proses *scanning* tanpa pengetikan dan *Hypertext* yaitu jenis teks yang memberikan *link* ke suatu tempat/meloncat ke topik tertentu.

2. Grafik (*image*)

Sangat bermanfaat untuk mengilustrasi informasi yang akan disampaikan terutama informasi yang tidak dapat dijelaskan dengan kata-kata. Jenis-jenis grafik seperti bitmap yaitu gambar yang disimpan dalam bentuk kumpulan *pixel*, yang berkaitan dengan titik-titik pada layar monitor. *Digitized picture* adalah gambar hasil rekaman *video* atau kamera yang dipindahkan ke komputer dan diubah ke dalam bentuk bitmaps. *Hyperpictures*, sama seperti *hypertext* hanya saja dalam bentuk gambar.

3. *Audio*

Multimedia tidak akan lengkap jika tanpa *audio* (suara). *Audio* bisa berupa percakapan, musik atau efek suara.

Format dasar *audio* terdiri dari beberapa jenis :

a. WAVE

Merupakan format file digital *audio* yang disimpan dalam bentuk digital dengan eksistensi WAV.

b. MIDI (*Musical Instrument Digital Interface*)

MIDI memberikan cara yang lebih efisien dalam merekam musik dibandingkan wave, kapasitas data yang dihasilkan juga jauh lebih kecil. MIDI disimpan dalam bentuk MID.

4. *Video*

Video menyediakan sumber yang kaya dan hidup untuk aplikasi multimedia. Dengan video dapat menerangkan hal-hal yang sulit digambarkan lewat kata-kata atau gambar diam dan dapat menggambarkan emosi dan psikologi manusia secara lebih jelas.

5. Animasi

Animasi adalah simulasi gerakan yang dihasilkan dengan menayangkan rentetan *frame* ke *layer*. *Frame* adalah satu gambar tunggal pada rentetan gambar yang membentuk animasi.

2.4 *Game*

Game merupakan kata dalam bahasa inggris yang berarti permainan. Permainan adalah sesuatu yang dapat dimainkan dengan aturan tertentu sehingga ada yang menang dan ada yang kalah, biasanya dalam konteks tidak serius atau dengan tujuan refreshing. Suatu cara belajar yang digunakan dalam menganalisa interaksi antara sejumlah pemain maupun perorangan yang menunjukkan strategi-strategi yang rasional. Teori permainan pertama kali ditemukan oleh sekelompok ahli Matematika pada tahun 1944. Teori itu dikemukakan oleh John von Neumann and Oskar Morgenstern yang berisi :

"Permainan terdiri atas sekumpulan peraturan yang membangun situasi bersaing dari dua sampai beberapa orang atau kelompok dengan memilih strategi yang dibangun untuk memaksimalkan kemenangan sendiri atau pun untuk meminimalkan kemenangan lawan. Peraturan-peraturan menentukan kemungkinan tindakan untuk setiap pemain, sejumlah keterangan diterima setiap pemain sebagai kemajuan bermain, dan sejumlah kemenangan atau kekalahan dalam berbagai situasi"**[3]**.

2.4.1 Elemen Dasar *Game*

Menurut Teresa Dillon **[6]** elemen-elemen dasar sebuah *game* adalah :

1. *Game Rule*

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

2. Plot

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

3. Thema

Di dalam biasanya ada pesan moral yang akan disampaikan.

4. *Character*

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

5. *Object*

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

6. *Text, Grafik dan Sound*

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

7. Animasi

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

8. *User Interface*

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

2.4.2 *Adventure Game*

Game adventure pada akhir tahun 1970 dan awal tahun 1980, adalah *game* pertama yang telah dibuat sebagai hiburan di komputer dan menjadi populer. *Adventure game* pertama disebut *text-based* adalah awal dari jenis *game* yang namanya didapat karena tidak memiliki grafik melainkan hanya teks yang mendeskripsikan ruangan dari imajenasi yang tertulis dalam text. Pemain akan menuliskan perintah dipapan ketik, dan *game* akan merespon dari perintah tersebut sesuai dengan apa yang diketikkan oleh pemain, atau menginformasikan kepada pemain bahwa inputan tidak dapat dikenali.

Akhirnya, orang-orang mulai melampirkan gambar ke bidang *puzzle*, termasuk *game* seperti *The King's Quest Series*, *Lucas Arts'* Seminal *Day of the Tentacle* and *Monkey Island Games*. Lucas Arts juga tidak jauh dari *full-text parsers*, bukan mengandalkan pada penyederhanaan kata kunci dan interface yang *iconic*.

Hari ini, *classic adventure game* telah hampir menghilang, tak seorang pun tahu sebabnya. *Game Myst* mungkin telah memberikan penjualan *game* bergenre *adventure*, tapi bukan merupakan penjualan terbesar, mereka mungkin telah menjadi alasan untuk kelangkaan saat ini. Orang-orang mulai mengasosiasikan *game adventure* dengan lambat, game casual itu hanyalah koleksi teka-teki dan dalam bentuk tulisan **[6]**.

2.5 Artificial Intelligence (Kecerdasan Buatan)

Kecerdasan buatan atau *artificial intelligence* merupakan salah satu bagian ilmu komputer yang membuat agar mesin (komputer) dapat melakukan pekerjaan seperti dan sebaik yang dilakukan oleh manusia. Pada awal diciptakannya, komputer hanya difungsikan sebagai alat hitung saja. Namun seiring dengan perkembangan jaman, maka peran komputer semakin mendominasi kehidupan umat manusia. Komputer tidak lagi hanya digunakan sebagai alat hitung, lebih dari itu, komputer diharapkan untuk dapat diberdayakan untuk mengerjakan segala sesuatu yang bisa dikerjakan oleh manusia.

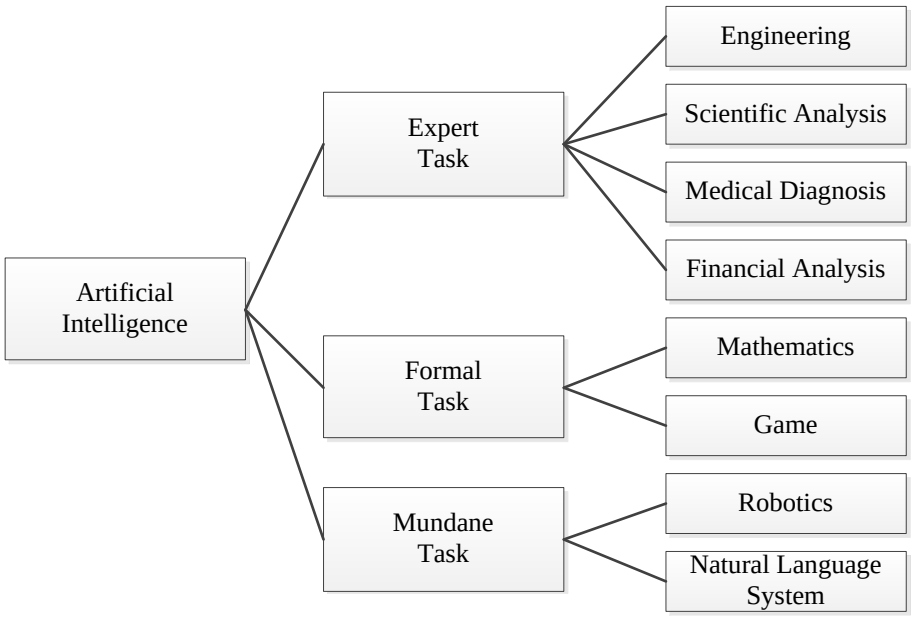
Pengertian kecerdasan buatan dapat dipandang dari berbagai sudut pandang, antara lain:

1. Sudut pandang kecerdasan.

Kecerdasan Buatan akan membuat mesin menjadi “cerdas” (mampu berbuat seperti apa yang dilakukan oleh manusia).

2. Sudut pandang penelitian.

Kecerdasan Buatan adalah suatu studi bagaimana membuat agar komputer dapat melakukan sesuatu sebaik yang dikerjakan oleh manusia. Gambar 2.1 merupakan sejumlah bidang-bidang tugas (*task domains*) dari AI.



Gambar 2.2 Bidang-Bidang Tugas

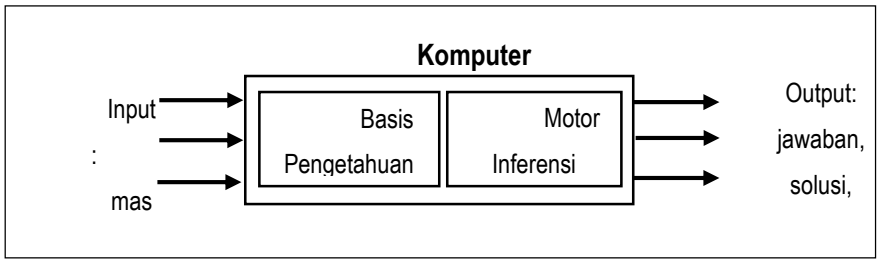
1. *Mundane task*
   a. Persepsi (*vision* & *speech*).
   b. Bahasa alami (*understanding*, *generation* & *translation*).
   c. Pemikiran yang bersifat *commonsense*.
   d. *Robot control*.
2. *Formal task*
   a. Permainan/*games*.
   b. Matematika (geometri, logika, kalkulus integral, pembuktian).
3. *Expert task*
   a. Analisis finansial.
   b. Analisis medikal.
   c. Analisis ilmu pengetahuan.
   d. Rekayasa (desain, pencarian kegagalan, perencanaan manufaktur).
4. Sudut pandang bisnis.

Kecerdasan buatan adalah kumpulan peralatan yang sangat *powerfull* dan metodologis dalam menyelesaikan masalah-masalah bisnis.

5. Sudut pandang pemrograman.

Kecerdasan buatan meliputi studi tentang pemrograman simbolik, penyelesaian masalah (*problem solving*) dan pencarian (*searching*). Untuk melakukan aplikasi kecerdasan buatan ada 2 bagian utama yang sangat dibutuhkan (Gambar 2.2), yaitu:

a. Basis Pengetahuan (*Knowledge Base*), berisi fakta-fakta, teori, pemikiran dan hubungan antara satu dengan lainnya.
b. Motor Inferensi (*Inference Engine*), yaitu kemampuan menarik kesimpulan berdasarkan pengalaman.



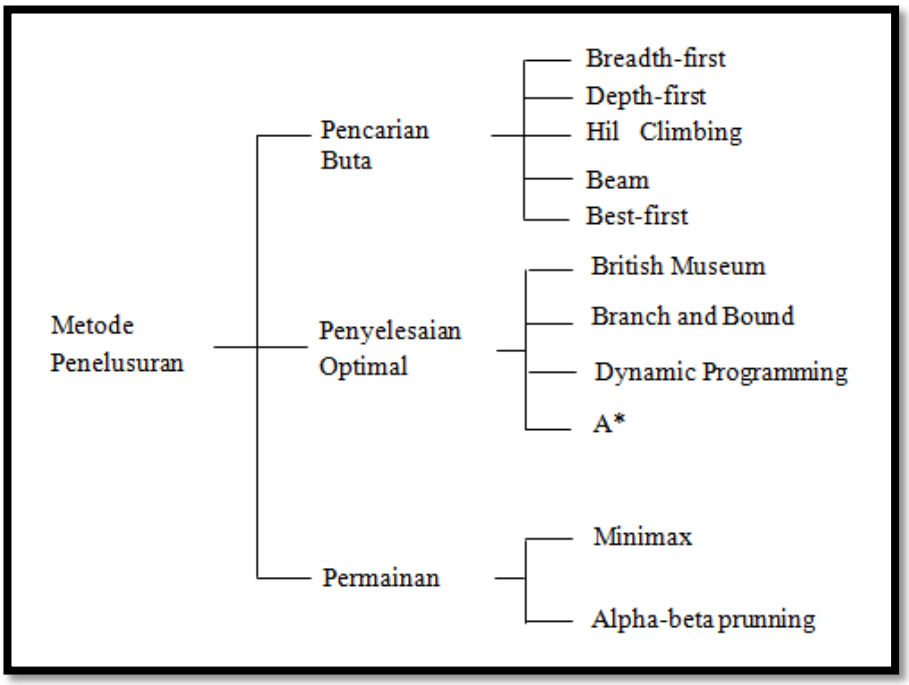
Gambar 2.3 Penerapan Konsep Kecerdasan Buatan

2.5.1 Algoritma Pencarian

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

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

Yang ketiga, adalah beberapa prosedur/metode yang terapkan saat berhadapan dengan musuh. Prosedur ini adalah *minimax search, alpha-beta prunning*. Metode ini banyak digunakan pada program-program permainan seperti catur dsb. Dalam Gambar 0.3 terdapat bagan untuk Metode *Searching*.



Gambar 2.4 Metode Pencarian [9].

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

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

2.5.2 Algoritma A* (A Star)

Algoritma A* 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 **[8]**.

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

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 kita mencari ke arah yang salah.

Cost untuk setiap simpul tidak harus berupa jarak. Cost bisa saja berupa waktu bila kita ingin mencari jalan dengan waktu tercepat untuk dilalui. Sebagai contoh, bila kita berkendaraan melewati jalan biasa bisa saja merupakan jarak terdekat, tetapi melewati jalan tol biasanya memakan waktu lebih sedikit.

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.

$$f(n) = g(n) + h(n) \qquad (2.1)$$

Dengan:

f(n) = fungsi evaluasi

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

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 **[8]**.

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

1. Simpul awal adalah sebuah terminologi untuk posisi awal sebuah benda.
2. Current node adalah simpul yang sedang dijalankan dalam algoritma pencarian jalan terpendek.
3. Simpul adalah petak-petak kecil sebagai representasi dari area pathfinding. Bentuknya dapat berupa persegi, lingkaran, maupun segitiga.
4. Neighbour 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 neighbour 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.

2.5.3 Fungsi Heuristik

BFS dan A* sebagai algoritma pencarian yang menggunakan fungsi heuristik untuk "menuntun" pencarian rute, khususnya dalam hal pengembangan dan pemeriksaan node - node pada peta. Dalam aplikasi ini, fungsi heuristik yang dipakai untuk pencarian rute mengisi nilai/notasi h pada algoritma BFS dan A*.

Ada beberapa fungsi heuristik umum yang bisa dipakai untuk algoritma BFS dan A* ini. Salah satunya adalah yang dikenal dengan istilah "Manhattan Distance". Fungsi heuristik ini digunakan untuk kasus di mana pergerakan pada peta hanya lurus (horizontal atau vertikal), tidak diperbolehkan pergerakan diagonal.:

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

$$\mathrm{h(n)} = \left(\mathrm{abs(n.x} - goal.\mathrm{x}) + \mathrm{abs(n.y} - goal.\mathrm{y})\right) \quad (2.2)$$

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

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

Maka fungsi heuristic yang digunakan adalah sebagai berikut:

$$\mathrm{h}_{diagonal(\mathrm{n})} = \min\left(\mathrm{abs(n.x} - goal.\mathrm{x}) + \mathrm{abs(n.y} - goal.\mathrm{y})\right) \quad (2.3)$$

$$h_{orthogonal(n)} = (\text{abs}(n.x - goal.x) + \text{abs}(n.y - goal.y)) \quad (2.4)$$

$$h(n) = h_{diagonal(n)} + (h_{orthogonal(n)} - (2 * h_{diagonal(n)})) \quad (2.5)$$

Dimana h_diagonal(n) adalah banyaknya langkah diagonal yang bisa diambil untuk mencapai goal dari node n. h_orthogonal adalah banyaknya langkah lurus yang bisa diambil untuk mencapai goal dari node n.

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

2.5.4 Algortima Simplified Memory-Bounded A*

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

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

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

Menetapkan limit yang besar pada ukuran open list dapat membantu meringankan masalah ini, namun fungsi untuk mengurangi penggunaan

memori menjadi terbuang. Pada kasus ekstrem yang lain, dengan memberi limit 1 simpul pada open list, ini bisa mempercepat sekaligus mengurangi penggunaan memori dalam pencarian, namun jalur yang ditemukan bisa tidak optimal **[8]**.

SMA* memiliki beberapa sifat sebagai berikut :

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

2.6 *Tools* yang Digunakan

Dalam membangun aplikasi *game* cerita rakyat fable dibutuhkan tools yang dapat digunakan dalam pembangunan *game* tersebut.

2.6.1 OOP (Object Oriented Programming)

OOP (*Object Oriented Programming*) atau yang dikenal dengan Pemrograman Berorientasi Objek merupakan paradigma pemrograman yang berorientasikan kepada objek. Semua data dan fungsi di dalam paradigma ini dibungkus ke dalam kelas-kelas atau objek-objek.

Model data berorientasi objek dikatakan dapat memberi fleksibilitas yang lebih, kemudahan mengubah program dan digunakan luas dalam teknik piranti lunak skala besar. Lebih jauh lagi, pendukung OOP mengklaim bahwa OOP lebih mudah dipelajari bagi pemula dibanding dengan pendekatan sebelumnya, dan pendekatan OOP lebih mudah dikembangkan dan dirawat.

Dengan menggunakan OOP maka dalam melakukan pemecahan suatu masalah tidak harus melihat bagaimana cara menyelesaikan suatu masalah tersebut (terstruktur) tetapi objek-objek apa yang dapat melakukan pemecahan

masalah tersebut. Sebagai contoh suatu departemen yang memiliki *manager*, sekretaris, petugas administrasi data dan lainnya. Misal manager tersebut ingin memperoleh data dari bagian administrasi maka *manager* tersebut tidak harus mengambilnya langsung tetapi dapat menyuruh petugas bagian administrasi untuk mengambilnya. Pada kasus tersebut seorang *manager* tidak harus mengetahui bagaimana cara mengambil data tersebut tetapi manager bisa mendapatkan data tersebut melalui objek petugas administrasi. Jadi untuk menyelesaikan suatu masalah dengan kolaborasi antar objek-objek yang ada karena setiap objek memiliki deskripsi tugasnya sendiri **[16]**.

2.6.1.1 Istilah-Istilah OOP

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

1. ***Object***

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

2. **Class**

Definisi *class* yaitu *template* untuk membuat objek. *Class* merupakan prototipe atau *blue prints* yang mendefinisikan variabel-variabel dan *method-method* secara umum. Objek merupakan hasil instansiasi dari suatu *class*. Proses

pembentukan objek dari suatu kelas disebut sebagai *instantiation*. Objek disebut juga sebagai *instances*. Dalam bahasa teoritis OOP, *class* merupakan kumpulan atas definisi data dan fungsi-fungsi dalam suatu unit untuk suatu tujuan tertentu. Sebagai contoh *'class of dog'* adalah suatu unit yang terdiri atas definisi-definisi data dan fungsi-fungsi yang menunjuk pada berbagai macam perilaku/turunan dari anjing. Sebuah *class* adalah dasar dari modularitas dan struktur dalam pemrograman berorientasi object. Sebuah *class* secara tipikal sebaiknya dapat dikenali oleh seorang *non-programmer* sekalipun terkait dengan domain permasalahan yang ada dan kode yang terdapat dalam sebuah *class* sebaiknya (relatif) bersifat mandiri dan independen (sebagaimana kode tersebut digunakan jika tidak menggunakan OOP). Dengan modularitas, struktur dari sebuah program akan terkait dengan aspek-aspek dalam masalah yang akan diselesaikan melalui program tersebut. Cara seperti ini akan menyederhanakan pemetaan dari masalah ke sebuah program ataupun sebaliknya.

3. Attributes

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

Status → exploring, moving, returning home

Speed → in miles per hour

Temperature → in Fahrenheit degrees

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

a. Behavior

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

Contoh behavior untuk objek Volcano Robot:

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

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

b. Abstraksi

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

2.6.1.2 Konsep-Konsep OOP

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

1. Enkapsulasi (*Encapsulation*)

Definisi enkapsulasi adalah Pembungkusan variabel dan *method* dalam sebuah objek yang terlindungi serta menyediakan *interface* untuk mengakses variabel tersebut. Variabel dan method yang dimiliki oleh suatu objek, bisa ditentukan hak aksesnya. Dalam OOP, konsep enkapsulasi sebenarnya merupakan

perluasan dari struktur dalam bahasa C. Contoh: jam tangan. Dalam hal ini, penting sekali untuk mengetahui waktu, sedangkan cara jam mencatat waktu dengan baik antara jam bertenaga baterai atau bertenaga gerak tidaklah penting untuk diketahui. Dengan kata lain enkapsulasi berfungsi untuk memastikan pengguna sebuah objek tidak dapat mengganti keadaan dalam/dari sebuah objek dengan cara yang tidak layak, hanya metode dalam objek tersebut yang diberi izin untuk mengakses keadaannya. Setiap objek mengakses *interface* yang menyebutkan bagaimana objek lainnya dapat berinteraksi dengannya. Objek lainnya tidak akan mengetahui dan tergantung kepada representasi dalam objek tersebut.

2. Pewarisan (Inheritance)

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

Contoh Pewarisan :



Gambar 2.5 Contoh Pewarisan

Keuntungan pewarisan :

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

3. Polimorfisme

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

2.6.2 UML (Unified Modeling Language)

Unified Modelling Language (UML) adalah sebuah “bahasa” yg telah menjadi standar dalam industri untuk visualisasi, merancang dan mendokumentasikan sistem piranti lunak. UML menawarkan sebuah standar untuk merancang model sebuah sistem. Dengan menggunakan UML dapat membuat model untuk semua jenis aplikasi piranti lunak, dimana aplikasi tersebut dapat berjalan pada piranti keras, sistem operasi dan jaringan apapun, serta ditulis dalam bahasa pemrograman apapun. Tetapi karena UML juga menggunakan *class* dan *operation* dalam konsep dasarnya, maka ia lebih cocok untuk penulisan

piranti lunak dalam bahasa bahasa berorientasi objek seperti C++, Java, C# atau VB.NET. Walaupun demikian, UML tetap dapat digunakan untuk modeling aplikasi prosedural dalam VB atau C. Seperti bahasa-bahasa lainnya, UML mendefinisikan notasi dan syntax /semantik. Notasi UML merupakan sekumpulan bentuk khusus untuk menggambarkan berbagai diagram piranti lunak. Setiap bentuk memiliki makna tertentu, dan UML *syntax* mendefinisikan bagaimana bentuk-bentuk tersebut dapat dikombinasikan. Notasi UML terutama diturunkan dari 3 notasi yang telah ada sebelumnya: Grady Booch OOD (*Object-Oriented Design*), Jim Rumbaugh OMT (*Object Modeling Technique*) dan Ivar Jacobson OOSE *(Object-Oriented Software Engineering*).

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

2.6.2.1 Konsep Dasar UML

UML adalah notasi yang dihasilkan dari penyatuan OMT (*Object Modeling Technique*), dan OOSE (*Object Oriented Software Enginering*). UML juga telah terpengaruh dengan penyatuan berorientasi objek yang lain, seperti yang diperkenalkan oleh Mellor dan Shlaer pada tahun 1998, Coad dan Yourdon pada tahun 1995, Wirf-Brock pada tahun 1990, dan Martin dan Odel pada tahun 1992.

Tujuan dari UML adalah untuk menyediakan sebuah satuan standar yang dapat digunakan oleh semua metode berorientasi objek dan untuk memilih dan mengintegrasikan prosedur penyatuan elemen terbaik. Sebagai contoh, didalam UML terdapat *use case* diagram yang diperkenalkan oleh OOSE dan menggunakan banyak fitur dari OMT *class diagrams*. UML juga termasuk konsep baru yang tidak terdapat pada metode utama pada saat itu, seperti mekanisme

tambahan dan sebuah bahasa pembatas. UML telah didesain untuk berbagai aplikasi. Oleh karena itu, UML menyediakan konsep untuk berbagai sistem dan aktifitas seperti pendistribusian sistem, analisis, disain sistem, *deployment*. Sistem *development* berfokus kepada tiga model yang berbeda dari sistem, yaitu **[16]**:

1. *The Functional Model*, terwakili dalam UML dengan *use case diagram*, menjelaskan fungsionalitas sebuah sistem dari sudut pandang pengguna.
2. *The Object Model*, terwakili dalam UML dengan *class diagram*, menjelaskan struktur dari sistem dalam hal objek, atribut, assosiasi, dan operasi. Selama persyaratan dan analisis, model objek dimulai sebagai objek model analisis dan menjelaskan konsep aplikasi yang sesuai terhadap sistem. Selama desain sistem, model objek disempurnakan kedalam *system design object model* dan termasuk deskripsi dari subsistem antar muka. Selama desain objek, model objek disempurnakan kedalam *Object Design Model* dan termasuk deskripsi rinci dari objek solusi.
3. *The Dynamic Model*, terwakili oleh UML dengan *interaction diagram*, *state machine diagram, and activity diagram*, menjelaskan prilaku internal dari sistem. *Interaction diagram* menjelaskan prilaku sebagai sebuah urutan pesan yang dipertukarkan antara sebuah *set of object*, sedangkan *state* diagram mesin menjelaskan prilaku dari kondisi suatu *individual object* dan memungkinkan transisi antara *state*. *Activity diagram* menjelaskan prilaku didalam hal kontrol dan alur data.

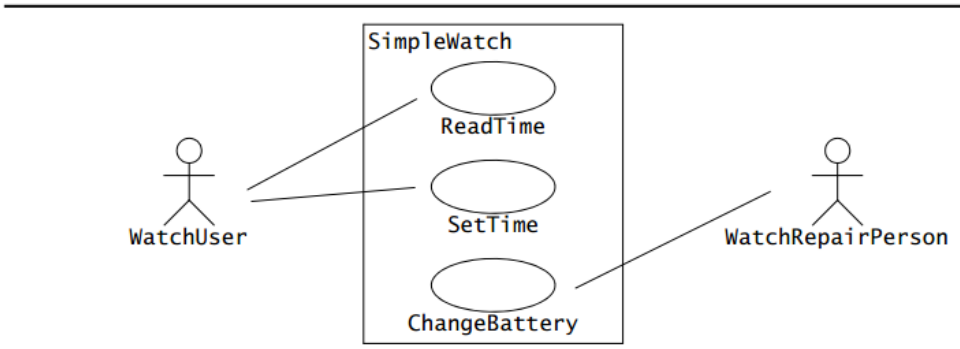
2.6.2.2 Gambaran Umum UML

UML secara singkat akan dijelaskan kepada 5 bagian, yaitu **[16]**:

a. Use Case Diagram

Use case digunakan selama permintaan elisitasi dan analisis untuk mewakili fungsi dari system. *Use case* berfokus pada prilaku dari sudut pandang eksternal. Sebuah *use case* menjelaskan fungsi yang disediakan oleh system yang menghasilkan *visible result* untuk aktor (pengguna sistem). Seorang aktor menjelaskan setiap entitas yang berinteraksi dengan system (misalnya seorang pengguna, sistem lain yang terkait, dan lingkungan fisik system). Identifikasi dari

aktor dan hasil *use case* dalam definisi batas system, yaitu, dalam membedakan tugas yang dilakukan oleh sistem dan tugas yang dilakukan oleh *environtment*. Aktor berada diluar batas sistem, sedangkan *use case* berada didalam batas sistem.



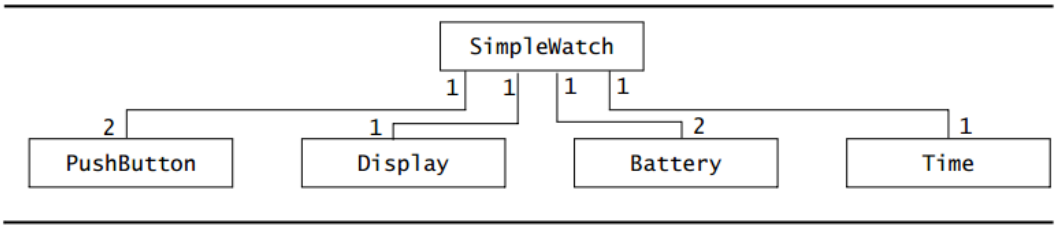
Gambar 2.6 *Use Case Diagram SimpleWatch*

Pada gambar 2.6 menggambarkan *use case diagram* untuk jam. *WatchUser* sebagai pengguna jam dapat memeriksa jam dengan menggunakan *ReadTime use case* atau pengguna jam dapat mengatur waktu jam dengan *SetTime use case.* Sedangkan untuk *ChangeBattery* hanya dapat dilakukan oleh *WatchRepariPerson use case*.

b. Class Diagram

Class diagram digunakan untuk menjelaskan struktur dari sistem. *Class* adalah abstraksi yang menentukan struktur umum dan prilaku satu set objek. Objek adalah contoh dari *class* yang dibuat, dimodifikasi, dan dihancurkan pada saat sistem dijalankan. Sebuah objek memiliki keadaan yang mencakup nilai-nilai atribut dan hubungan dengan *object* lain.

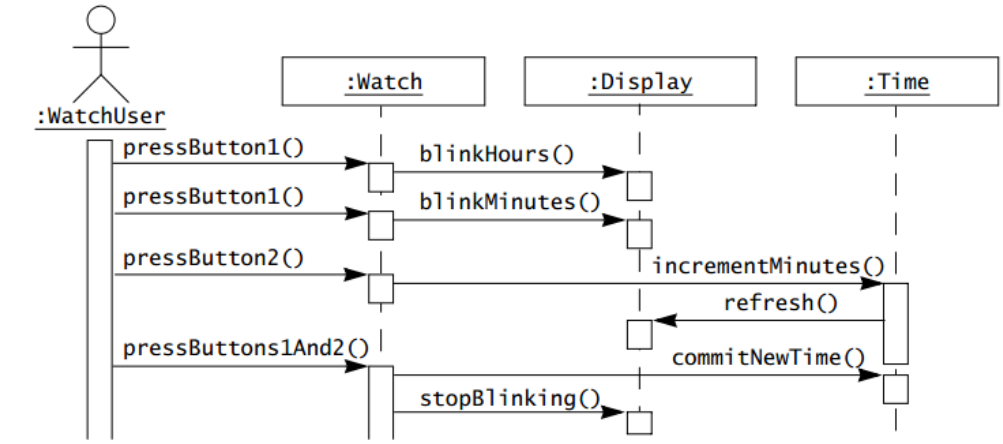
Class diagram menjelaskan sistem dari segi *object*, *classes*, *attributes*, *operations*, dan asosiasi dari sistem. Sebagai contoh pada gamabar 2.7, *class diagram* mengambarkan elements dari semua *class SimpleWatch*. Objek jam ini memiliki asosiasi ke semua objek dari kelas *PushButton*, objek dari kelas *display*, objek dari kelas *Time*, dan objek dari kelas *Battery*.



Gambar 2.7 *Class Diagram SimpleWatch*

c. Interaction Diagram

Interaction diagram digunakan untuk memformalkan prilaku dinamis dari sebuah sistem dan untuk memvisualisasikan komunikasi antara objek. *Interaction diagram* berguna untuk mengidentifikasi objek tambahan yang berpartisipasi didalam *use case*. Bren Bruegge dan Allen H Dutoit dalam bukunya menyebutkan bahwa objek yang terlibat dalam *use case* disebut *participating object*. Sebuah interaksi diagram merupakan interaksi yang terjadi diantara objek. Sebagai contoh pada gambar 2.8 adalah bentuk kasus diagram interaksi, disebut sebagai *sequence diagram*, untuk *use case SetTime* pada *SimpleWatch*.

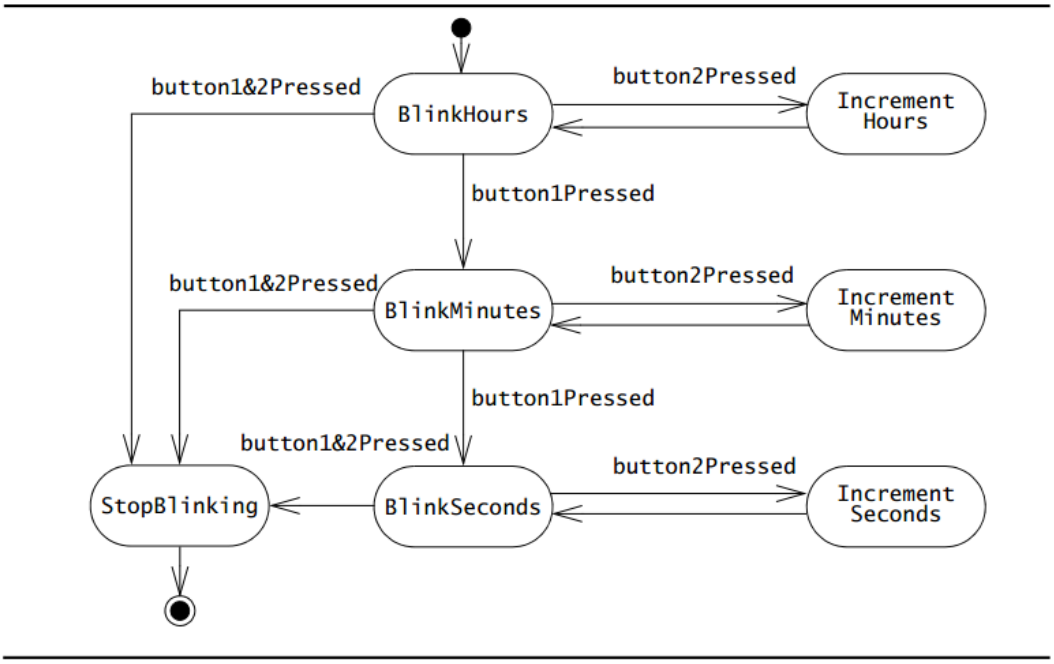


Gambar 2.8 *Sequence Diagram*

d. State Machine Diagram

State machine diagram menjelaskan tentang prilaku dinamis dari sebuah objek individu sebagai sebuah *number of state* dan transisi antara *state*. Sebuah *state* menerangkan set tertentu dari nilai untuk sebuah objek. Mengingat *state* adalah transisi yang menerangkan sebuah *future state* objek yang dapat pindah

dan kondisi yang berhubungan dengan perubahan *state*. Sebagai contoh pada gambar 2.9 sebuah *state machine diagram* untuk *SimpleWatch*. Sebuah lingkaran kecil menginisialisasikan *blinkhours* sebagai *initial state* atau kondisi awal. Dan lingkarang hitam yang dilingkari oleh lingkarang pada akhir disebut sebagai *final state* atau kondisi akhir. Dapat diperhatikan bahwa diagram ini merupakan informasi yang berbeda dengan *sequence diagram* pada gambar 2.8. *Sequence diagram* berfokus pada pertukaran pesan antara objek sebagai akibat dari peristiwa eksternal yang diciptakan oleh aktor, sedangkan *state machine diagram* berfokus pada perpindahan antara *state* sebagai hasil dari peristiwa eksternal untuk sebuah objek individu.

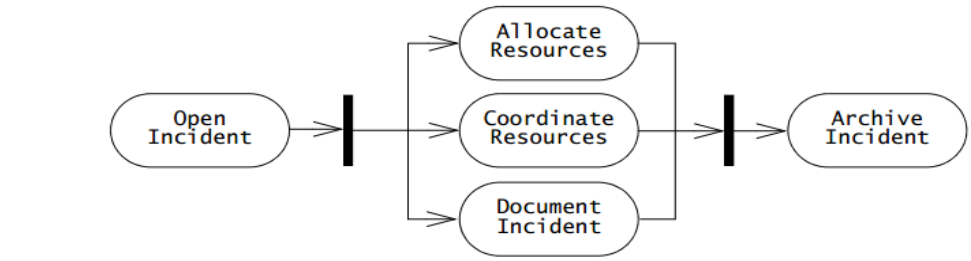


Gambar 2.9 *State Machine Diagram*

e. Activity Diagram

Activity diagram menjelaskan tentang prilaku dari sebuah sistem dalam hal *activities*. *Activities* merupakan pemodelan element yang menerangkan pelaksanaan seperangkat operasi. Pelaksanaan sebuah *activity* dapat dipicu oleh penyelesaian *activity* yang lain, oleh ketersediaan objek, atau kejadian eksternal. *Activity diagram* mirip dengan *flowchart diagram* yang dapat digunakan untuk

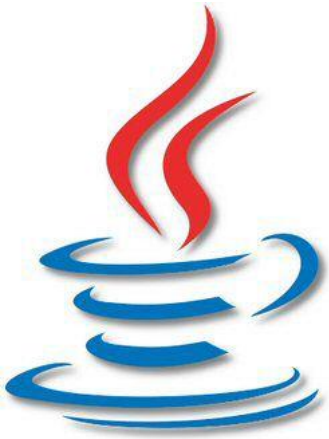
menerangkan alur control seperti alur operasi yang terjadi dan aliran data seperti objek yang bertukar pada saat operasi berlangsung.



Gambar 2.10 ***Activity Diagram***

2.6.3 Java

Java adalah suatu teknologi di dunia *software* komputer, yang merupakan suatu bahasa pemrograman dan sekaligus suatu *platform*. Sebagai bahasa pemrograman, Java dikenal sebagai bahasa pemrograman tingkat tinggi. Java mudah dipelajari, terutama bagi *programmer* yang telah mengenal C/C++. Java merupakan bahasa pemrograman berorientasi objek yang merupakan paradigma pemrograman masa depan. Sebagai bahasa pemrograman Java dirancang menjadi handal dan aman. Java juga dirancang agar dapat dijalankan di semua *platform* dan juga dirancang untuk menghasilkan aplikasi – aplikasi dengan performansi yang terbaik, seperti aplikasi *database* Oracle 8i/9i yang *core-nya* dibangun menggunakan bahasa pemrograman Java. Sedangkan Java bersifat *neutral architecture*, karena Java *Compile*r yang digunakan untuk mengkompilasi kode program Java dirancang untuk menghasilkan kode yang netral terhadap semua arsitektur perangkat keras yang disebut sebagai *Java Bytecode* **[12]**.



Gambar 2.11 Logo Java

Sebagai sebuah platform, Java terdiri atas dua bagian utama, yaitu:

a. *Java Virtual Machine* (JVM).
b. *Java Application Programming Interface* (Java API).

Sun membagi arsitektur Java membagi tiga bagian, yaitu:

a. *Enterprise Java* (J2EE) untuk aplikasi berbasis web, aplikasi system tersebar dengan beraneka ragam klien dengan kompleksitas yang tinggi. Merupakan superset dari Standar Java.
b. *Standar Java* (J2SE), ini adalah yang biasa dikenal sebagai bahasa Java.
c. *Micro Java* (J2ME) merupakan subset dari J2SE dan salah satu aplikasinya yang banyak dipakai adalah untuk *wireless device/mobiledevice.*

2.6.4 Adobe Flash

Adobe Flash (dahulu bernama Macromedia Flash) adalah salah satu perangkat lunak komputer yang merupakan produk unggulan Adobe Systems. Adobe Flash digunakan untuk membuat gambar vektor maupun animasi gambar tersebut. Berkas yang dihasilkan dari perangkat lunak ini mempunyai file extension .swf dan dapat diputar dipenjelajah web yang telah dipasangi Adobe Flash Player. Flash menggunakan bahasa pemrograman bernama ActionScript yang muncul pertama kalinya pada Flash 5.

Sebelum tahun 2005, Flash dirilis oleh Macromedia. Flash 1.0 diluncurkan pada tahun 1996 setelah Macromedia membeli program animasi vektor bernama FutureSplash. Versi terakhir yang diluncurkan di pasaran dengan menggunakan nama 'Macromedia' adalah Macromedia Flash 8. Pada tanggal 3 Desember 2005 Adobe Systems mengakuisisi Macromedia dan seluruh produknya, sehingga nama Macromedia Flash berubah menjadi Adobe Flash.

Adobe Flash merupakan sebuah program yang didesain khusus oleh Adobe dan program aplikasi standar authoring tool professional yang digunakan untuk membuat animasi dan bitmap yang sangat menarik untuk keperluan pembangunan situs web yang interaktif dan dinamis. Flash didesain dengan kemampuan untuk membuat animasi 2 dimensi yang handal dan ringan sehingga flash banyak digunakan untuk membangun dan memberikan efek animasi pada website, CD Interaktif dan yang lainnya. Selain itu aplikasi ini juga dapat digunakan untuk membuat animasi logo, movie, game, pembuatan navigasi pada situs web, tombol animasi, banner, menu interaktif, interaktif form isian, e-card, screen saver dan pembuatan aplikasi-aplikasi web lainnya. Dalam Flash, terdapat teknik-teknik membuat animasi, fasilitas action script, filter, custom easing dan dapat memasukkan video lengkap dengan fasilitas playback FLV. Keunggulan yang dimiliki oleh Flash ini adalah ia mampu diberikan sedikit code pemograman baik yang berjalan sendiri untuk mengatur animasi yang ada didalamnya atau digunakan untuk berkomunikasi dengan program lain seperti HTML, PHP, dan Database dengan pendekatan XML, dapat dikolaborasikan dengan web, karena mempunyai keunggulan antara lain kecil dalam ukuran file outputnya.

Movie-movie Flash memiliki ukuran file yang kecil dan dapat ditampilkan dengan ukuran layar yang dapat disesuaikan dengan keingginan. Aplikasi Flash merupakan sebuah standar aplikasi industri perancangan animasi web dengan peningkatan pengaturan dan perluasan kemampuan integrasi yang lebih baik. Banyak fiture-fiture baru dalam Flash yang dapat meningkatkan kreativitas dalam pembuatan isi media yang kaya dengan memanfaatkan kemampuan aplikasi tersebut secara maksimal. Fiture-fiture baru ini membantu kita lebih memusatkan

perhatian pada desain yang dibuat secara cepat, bukannya memusatkan pada cara kerja dan penggunaan aplikasi tersebut. Flash juga dapat digunakan untuk mengembangkan secara cepat aplikasi-aplikasi web yang kaya dengan pembuatan script tingkat lanjut. Di dalam aplikasinya juga tersedia sebuah alat untuk men-debug script. Dengan menggunakan Code hint untuk mempermudah dan mempercepat pembuatan dan pengembangan isi ActionScript secara otomatis. Untuk memahami keamanan Adobe Flash dapat dilihat dari beberapa sudut pandang, berdasarkan beberapa sumber referensi bahwa tidak ada perbedaan menyolok antara HTML dan JavaScript dimana didalamnya terdapat banyak tools yang dapat diambil dari SWF termasuk ActionScript. Sehingga kode data dapat terjamin keamanannya. Oleh sebab itu, semua kebutuhan data yang terdapat dalam SWF dapat diambil kembali melalui server. Keuntungan menggunakan metode yang sama dengan menggunakan aplikasi web yang standar adalah akan menjamin dan mengamankanpenyimpanan dan perpindahan data **[17]**.

2.6.5 Action Script 3.0

Adobe ActionScript merupakan bahasa pemrograman yang bekerja di dalam platform Adobe Flash. Adobe ActionScript memang dibangun sebagai cara untuk mengembangkan pemrograman interaktif secara efisien menggunakan platform aplikasi adobe Flash ActionScript mulai dari animasi yang sederhana sampai dengan yang kompleks sekalipun, penggunaan data, dan aplikasi interface yang interaktif. Pertama kali diperkenalkan dalam Flash Player 9, ActionScript merupakan bahasa pemrograman berorientasi objek didasarkan pada ECMAScript standar yang sama yang menjadi dasar JavaScript dan memberikan hasil yang luar biasa dalam kinerja dan produktifitas pengembang. ActionScript 2, versi ActionScript yang telah digunakan dalam Flash Player 8 dan sebelumnya, tetap didukung dalam Flash Player 9 dan Flash Player 10.

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

a. *Event*

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

1) *Mouse event*

Event yang berkaitan dngan penggunaan mouse.

2) *Keyboard Event*

Kejadian pada saat menekan tombol keyboard.

3) *Frame Event*

Event yang diletakkan pada *keyframe*.

4) *Movie Clip Event*

Event yang disertakan pada *movie clip*.

b. 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 (” ”).

c. *Action*

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

1) *Action Frame*: adalah *action* yang diberikan pada *keyframe*. Sebuah *keyframe* akan ditandai dengan huruf a bila pada *keyframe* tersebut terdapat sebuah action.
2) *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.

2.6.6 Adobe Photoshop CS6

Adobe Photoshop adalah sebuah program yang digunakan untuk melakukan perubahan-perubahan (manipulasi) pada gambar, yang dapat menghasilkan gambar digital dengan kualitas tinggi. Adobe Photoshop juga dapat melakukan perubahan-perubahan dari gambar atau foto hasil *scanner*. Misalnya dengan memperbesar atau memperkecil gambar yang sudah ada, melakukan rotasi (perputaran), merubah warna, memberi efek-efek khusus seperti yang terdapat pada menu *filter*. Hal yang utama dalam Adobe Photoshop adalah penggunaan istilah **[13]**:

1. ***Image***

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

a. Grafis pixel

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

b. Grafis Vektor

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

2. *Pixe*l (*picture element*)

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

2.7 Metode Pengujian

2.7.1 Metode Pengujian *Black Box*

Balck box adalah cara pengujian yang dilakukan dengan hanya menjalankan atau mengeksekusi unit atau modul kemudian diamati apakah hasil dari unit itu sesuai dengan proses yang dinginkan. Dengan kata lain, *black box* merupakan *user testing*, biasanya pengujian perangkat lunak dengan metode *black box* melibatkan *client* atau pelanggan yang memesan produk perangkat lunak tersebut. Dalam pengujian *black box* pemrogram berusaha menemukan kesalahan seperti **[15]**:

a. Fungsi – fungsi yang tidak benar atau hilang.
b. Kesalahan Interface.
c. Kesalahan dalam struktur data atau akses database internal.
d. Kesalahan kinerja.
e. Inisialisasi dan kesalahan terminasi.

2.7.2 Metode Pengujian *White Box*

Pengujian *white box* merupakan metode desain uji kasus yang menggunakan struktur control dari desain procedural untuk menghasilkan kasus-kasus uji. Dengan menggunakan metode uji coba *white box*, para pengembang software dapat menghasilkan kasus – kasus uji yang :

a. Menjamin bahwa *independent paths* dalam modul telah dilakukan sedikitnya satu kali.
b. Melakukan seluruh keputusan logikal dari sisi benar maupun salah.
c. Melakukan seluruh perulangan sesuai batasannya dan dalam batasan operasionalnya.
d. Menguji struktu data internal untuk memastikan validitasnya.

Menguji logikal *software* membutuhkan banyak waktu dan usaha, hal ini dikarenakan sifat kerusakan alami dari *software* itu sendiri, yaitu :

a. Kesalahan logika dan kesalahan asumsi secara proposional terbalik dengan kemungkinan bahwa program akan dieksekusi. Kesalahan akan selalu ada ketika mendesain dan implementasi fungsi, kondisi dan kontrol yang keluar dari alur utama. Setiap harinya pemrosesan selalu berjalan dengan baik dan dimengerti sampai bertemu "kasus sepesial" yang mengarahkan kepada kehancuran.
b. Alur logika program biasanya berkebalikan dari intuisi, yaitu tanpa disadari asumsi dari alur kontrol dan dapat mengarah kepada pada kesalahan desain yang tidak dapat terlihat hanya dengan satu kali uji coba.
c. Kesalahan *typographical*(cetak) bersifat *random*, ketika program diterjemahkan kedalam kode sumber bahasa pemrograman, maka akan terjadi kesalahan pengetikan. Banyak yang terdeteksi dengan mekanisme pemeriksaan sintaks, tetapi banyak juga yang tidak terdeteksi sampai dengan dimulainya ujicoba.

Karena alasan tersebut diatas, maka uji coba *white box* testing perlu dilakukan selain *balck box* testing **[15]**.

2.7.3 Skala Likert

Skala Likert digunakan untuk mengukur sikap, pendapat dan persepsi seseorang atau sekelompok orang tentang fenomena sosial, dengan skala Likert maka variabel yang akan diukur dujabarkan menjadi indikator variabel. Kemudian indikator tersebut dijadikan sebagai titik tolak untuk menyusun item-item instrumen yang dapat berupa pernyataan atau pertanyaan.

Jawaban setiap item instrumen Skala Likert mempunyai gradasi dari sangat positif sampai negatif yang untuk keperluan analisis kuantitatif **[14]**, setiap jawaban dapat diberi skor seperti berikut :

a. Sangat setuju diberi skor 5
b. Setuju diberi skor 4

c. Ragu-ragu diberi skor 3
d. Tidak setuju diberi skor 2
e. Sangat tidak setuju diberi skor 1

Berdasarkan data tersebut, dapat dicari prosentase masing-masing jawaban dengan menggunakan rumus :

$$Y = P/Q*100\% \quad (2.6)$$

Keterangan :

Y = Prosentase

P = Banyaknya Jawaban Responden Tiap Soal

Q = Jumlah Responden