BAB 2

LANDASAN TEORI

2.1 Permainan

Game adalah kata dalam bahasa Inggris yang berarti permainan atau pertandingan. *Game* bisa diartikan sebagai aktivitas terstruktur atau seni terstruktur, yang biasanya dilakukan untuk bersenang-senang dan kadang digunakan sebagai alat pembelajaran. Saat ini perkembangan *games* di komputer sangat cepat. Para pengelola industri *game* berlomba-lomba untuk menciptakan game yang lebih nyata dan menarik untuk para pemainnya. Hal inilah yang membuat perkembangan *games* di komputer sangat cepat. Sehingga *games* bukan hanya sekedar permainan untuk mengisi waktu luang atau sekedar hobi. Melainkan sebuah cara untuk meningkatkan kreatifitas dan tingkat intelektual para penggunanya.

Game secara umum dibagi menjadi lima bagian diantaranya:

1. *Arcade/Action*, *game* yang memiliki interaksi dengan manusia yang tinggi dan menggunakan grafik yang beragam, seperti: *game Quake* dan *Freelancer*.
2. *Puzzle*, *game* yang didalamnya terdapat unsur logika, seperti : *game Tetris dan maze*.
3. *Card*, *game* seperti *Poker* dan *BlackJack*.

4. *Strategy*, *game* yang memiliki suatu teknik dan pemikiran serta manajemen didalamnya seperti : *game Command and Conquer* dan *Warcraft*.
5. *Sport*, *game* seperti permainan olah raga yang sesungguhnya seperti *game* FIFA 2000.

2.1.1 *Puzzle*

Permainan *puzzle* adalah permainan yang fokus pada tantangan konseptual dan logis, meskipun kadang-kadang ada beberapa bagian yang ditambahkan seperti waktu yang terbatas permainan atau kejadian yang lainnya. Meskipun banyak *game action* dan *game* petualangan yang melibatkan teka-teki seperti mendapatkan objek yang tidak dapat diakses, permainan *puzzle* yang sebenarnya berfokus pada pemecahan teka-teki pada permainan tersebut. Dalam permainan *puzzle* biasanya melibatkan bentuk, warna, atau simbol, dan pemain harus menggerakan/menggunakan mereka ke dalam pola tertentu.

Biasanya memecahkan koleksi *puzzle* dilakukan secara acak, dalam permainan *puzzle* akan diberikan serangkaian *puzzle* yang terkait dengan satu tema. Tema ini bisa melibatkan pengenalan pola, logika, atau pemahaman proses. Permainan ini biasanya memiliki seperangkat aturan sederhana, dimana pemain menggerakan potongan objek dalam ubin tertentu, jalur maupun ruang kosong. Pemain harus harus membongkar petunjuk untuk menang, yang kemudian akan maju ke tingkat berikutnya. Melengkapi setiap teka-teki biasanya akan mengarah ke tantangan yang lebih sulit, meskipun beberapa permainan menghindari untuk memberikan tingkat yang lebih mudah daripada yang lebih sulit.

Game Pacman (*trademark* milik Namco) adalah satu *game* dalam *genre* ini. *Game* ini akan memuat suatu alur jalan yang harus dilalui oleh pemain dan disediakan *bad guy*/*monster* yang bersifat merusak dan berusaha untuk menghalangi pemain untuk melewati jalan tersebut. *Game* disusun agar pemain bisa mencapai satu titik *exit* tertentu atau keluar ke *level* yang lebih rumit. Jumlah, kecepatan dan tingkat intelegensia dari *bad guy/monster* akan menentukan tingkat kesulitan dari *game*.

2.2 Kecerdasan Buatan

Istilah kecerdasan berasal dari bahasa Inggris: "*Artificial Intelligence*", para ilmuan memiliki dua cara pandang yang berbeda tentang AI. Yang pertama adalah memandang AI sebagai bidang ilmu yang hanya fokus pada proses berpikir. Sedangkan yang kedua adalah memandang AI sebagai bidang ilmu yang fokus pada tingkah laku. Cara pandang kedua memandang AI secara lebih luas karena suatu tingkah laku selalu didahului dengan proses berpikir. Berdasarkan pendekatannya definisi AI terbagi menjadi empat kategori [11], yaitu :

1. ***Thinking humanly*** : *the cognitive modelling approach*

   Pendekatan ini dilakukan dengan dua cara sebagai berikut :

   1. Melalui introspeksi : mencoba menangkap pemikiran-pemikiran kita sendiri pada saat kita berpikir.
   2. Melalui eksperimen-eksperimen psikologi.

2. ***Acting humanly*** : *the turing test approach*

   Pada tahun 1950, Alan Turing merancang suatu ujian bagi komputer berintelejensia untuk menguji apakah komputer tersebut mampu mengelabui

seorang manusia yang menginterogasinya melalui *teletype* (komunikasi berbasis teks jarak jauh). Jika *interrogator* tidak dapat membedakan yang diinterogasi adalah manusia atau komputer, maka komputer berintelejensia tersebut lolos dari *Turing test*. Komputer tersebut perlu memiliki kemampuan *Natural Language Processing, Knowledge Representation, Automated Reasoning, Machine Learning, Computer Vision, Robotics*. *Turing test* sengaja menghindari interaksi fisik antara interrogator dan komputer karena simulasi fisik manusia tidak memerlukan intelejensia.

3. ***Thinking rationally*** : *the laws of thought approach*

   Terdapat dua masalah dalam pendekatan ini, yaitu :

   1. Tidak mudah untuk membuat pengetahuan informal dan menyatakan pengetahuan tersebut ke dalam *formal term* yang diperlukan oleh notasi logika, khususnya ketika pengetahuan tersebut memiliki kepastian kurang dari 100%.
   2. Terdapat perbedaan besar antara dapat memecahkan masalah "dalam prinsip" dan memecahkannya "dalam dunia nyata".

4. ***Acting rationally*** : *the rational agent approach*

   Membuat inferensi yang logis merupakan bagian dari suatu *rational agent*. Hal ini disebabkan satu-satunya cara untuk melakukan aksi secara rasional adalah dengan menalar secara logis. Dengan menalar secara logis, maka bisa didapatkan kesimpulan bahwa aksi yang diberikan akan mencapai tujuan atau tidak. Jika mencapai tujuan, maka *agent* dapat melakukan aksi berdasarkan kesimpulan tersebut.

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

a. 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 :

1. Persepsi (*vision* & *speech*).
2. Bahasa alami (*understanding, generation & translation*).
3. Pemikiran yang bersifat *commonsense*.
4. *Robot control*.

b. Formal task

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

1. Permainan/*games*.
2. Matematika (geometri, logika, kalkulus, integral, pembuktian).

c. 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 :

1. Analisis finansial.
2. Analisis medikal.
3. Analisis ilmu pengetahuan.

4. Rekayasa (desain, pencarian, kegagalan, perencanaan, manufaktur).

2.3 Algoritma A* (*A Star*)

Algoritma A* adalah algoritma yang menggabungkan algoritma Dijkstra dan algoritma *Best First Search* untuk mengkompensasikan kelemahan-kelemahan dari masing-masing algoritma. Selain menghitung biaya yang diperlukan untuk berjalan dari suatu *node* ke *node* lainnya, algoritma A* menggunakan *heuristic* untuk memprioritaskan pemeriksaan *node-node* pada arah yang benar.

Algoritma ini pertama kali diperkenalkan pada 1968 oleh Peter Hart, Nils Nilsson, dan Bertram Raphael. Dalam ilmu komputer, A* (yang diucapkan dengan "A star") merupakan salah satu algoritma pencarian graph terbaik yang mampu menemukan jalur dengan biaya pengeluaran paling sedikit dari titik permulaan yang diberikan sampai ke titik tujuan yang diharapkan.

Algoritma ini menggunakan fungsi *distance* + *cost*, dimana cost antara simpul adalah jaraknya dan perkiraan *cost* dari suatu simpul ke simpul tujuan adalah penjumlahan jarak dari simpul tersebut ke simpul tujuan. Agar lebih jelas dapat ditunjukkan seperti berikut ini.

$$\boldsymbol{f(n) = g(n) + h(n)} \qquad \textbf{(4)}$$

dengan :

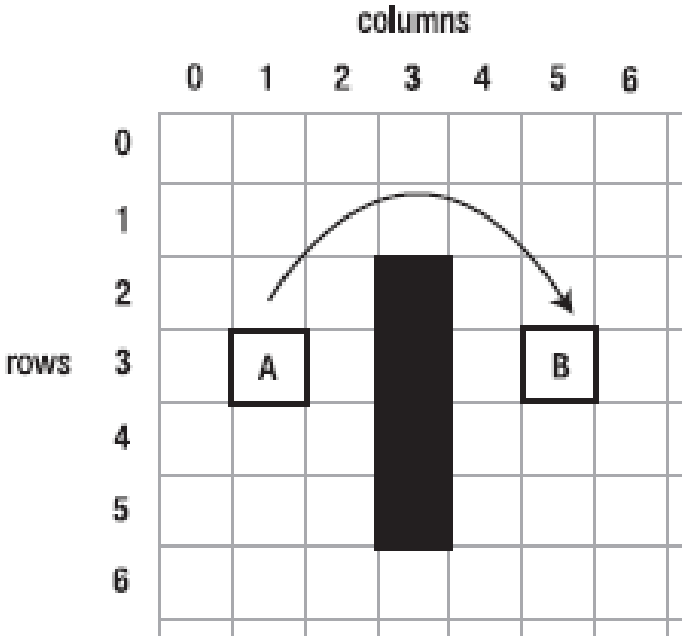
$f(n) =$ fungsi evaluasi

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

$h(n) =$ estimasi biaya untuk sampai pada suatu tujuan mulai dari n

A* dapat dijamin keoptimalannya untuk sembarang *heuristic* yang berarti bahwa tidak ada satupun algoritma lain yang menggunakan *heuristic* yang sama untuk mengecek lebih sedikit titik dari A*, kecuali ketika ada beberapa solusi parsial dimana h dapat dengan tepat memprediksi ongkos jalur minimal [11].

Untuk lebih jelas, berikut ini adalah cara kerja algoritma A* dapat dilihat pada Gambar 2.1.



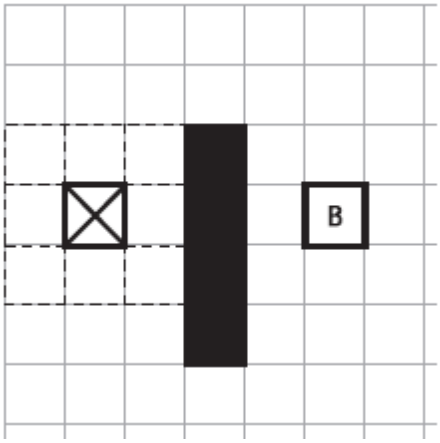
Gambar 2.1 Posisi titik awal dan titik tujuan.

Pada gambar diatas terdapat 3 objek, yaitu : kotak A, kotak B dan kotak berwarna hitam, dimana kotak A adalah titik awal, kotak B adalah titik tujuan dan kotak berwarna hitam adalah sebagai penghalang. Langkah berikutnya adalah proses pencarian jalan terpendek menuju titik tujuan, dengan langkah sebagai berikut :

1. Membagi proses pencarian jalan menjadi langkah-langkah kecil.
2. Mencari tahu langkah berikutnya yang akan di ambil.

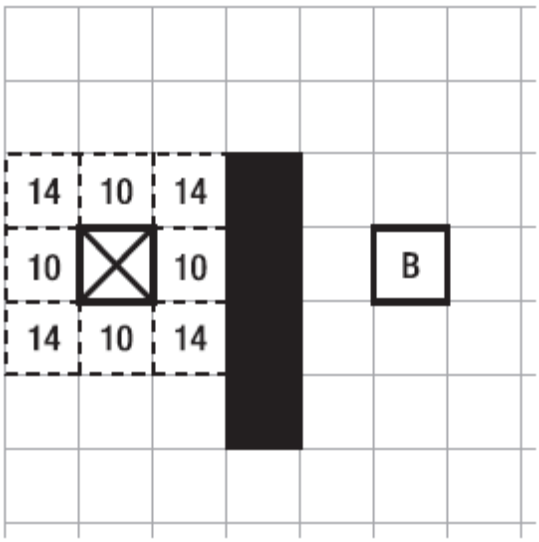
3. Mengambil langkah dan ulangi proses pencarian sampai mendapatkan jalan yang terpendek.

Dalam algoritma A* setiap langkah pencarian jalur disebut *node*. Algoritma A* akan mulai mencari jalan terpendek pada titik A (titik awal), titik A disebut *parent node*. Selanjutnya memeriksa 8 kotak yang berada di sekitar *parent node* untuk memilih jalan yang akan diambil selanjutnya, jika ada *node* yang terhalang oleh dinding atau objek yang tidak bisa dilewati maka akan diabaikan.



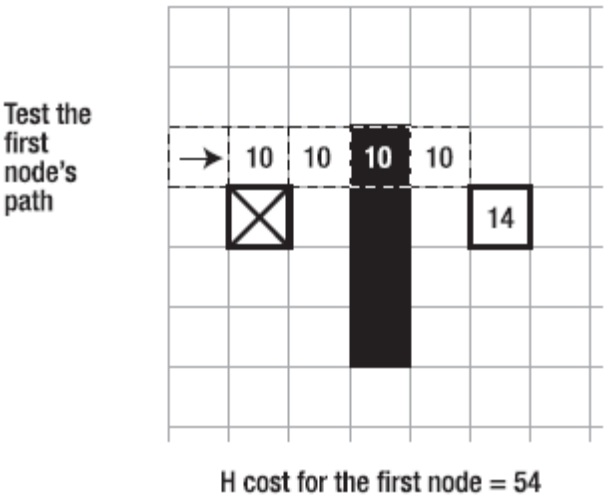
Gambar 2.2 Pencarian *node* di sekitar *parent node*.

Pergerakan didalam kotak hanya mempunyai dua dan mempunyai biaya yang berbeda yaitu, gerak secara diagonal yang mempunyai biaya 14 dan gerak secara langsung (horizontal dan vertikal) mempunyai biaya 10. Biaya ini akan digunakan sebagai pembanding. Proses awal pencarian akan dicari semua jalan yang memungkinkan dari titik awal menuju titik tujuan. Setiap biaya yang berada di sekitar *parent node* adalah G.



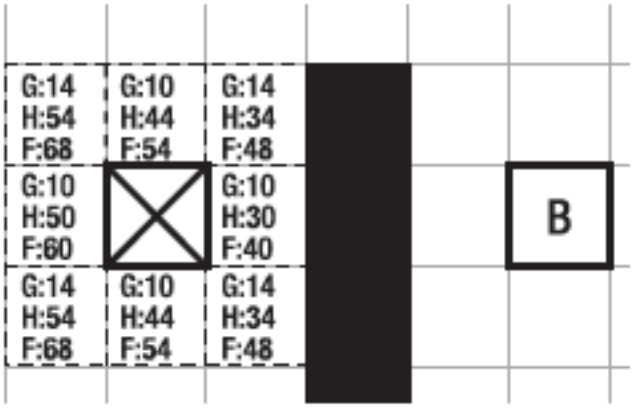
Gambar 2.3 Menghitung nilai G disekitar *node*

Tahap selanjutnya mencari *node* terdekat menuju titik tujuan, yaitu titik B. Perhitungan biaya perjalanan dihitung mulai dari *node* pertama menuju ke titik B, maka biaya perjalanan yang didapat adalah 54 karena ada node yang terhalang dinding maka nilai node tersebut tidak dianggap jadi hasil akhir dari biaya yang didapat adalah 44.



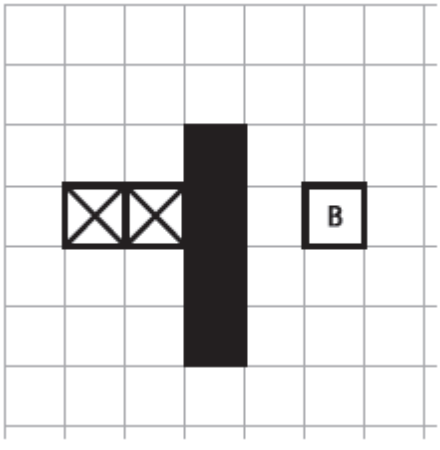
Gambar 2.4 Hasil biaya setiap *node* menuju titik B

Jarak pencarian ini disebut *heuristic*. *Heuristic* adalah tahapan proses uji coba pencarian. Algoritma A* akan mencari *node* mana yang memiliki jalur yang paling sedikit biayanya, sehingga mencoba mencari semua *node* yang berada disekitar *node parent*. Biaya untuk setiap jalur *heuristic* disebut H. Jika biaya G dan H digabungkan maka akan mendapatkan nilai akhir yang disebut F. Hasilnya akan terlihat seperti pada Gambar 2.5 dimana nilai setiap *node* akan diketahui.



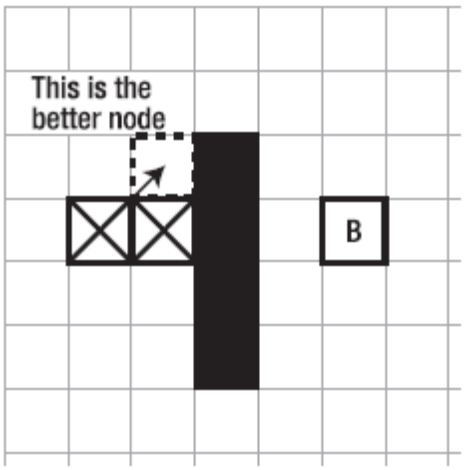
Gambar 2.5 Pencarian nilai F

Setelah nilai G dan H diketahui selanjutnya mencari nilai F, *node* yang mempunyai nilai terkecil akan dipilih sebagai jalur berikutnya.



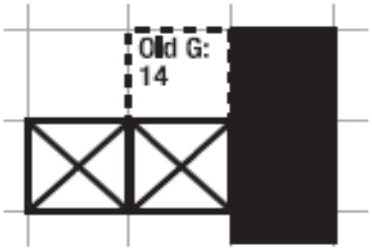
Gambar 2.6 Node yang memiliki nilai F terkecils

Node ini sekarang berpotensial menjadi *parent node* yang baru. Setiap *parent node* berpotensial memiliki langkah menuju titik B. Sebenarnya *parent node* yang baru ini bukan langkah yang terbaik., akan lebih baik jika bergerak secara diagonal.



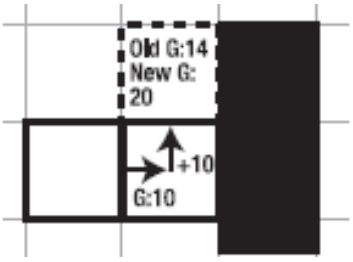
Gambar 2.7 Proses pencarian node

Algoritma A* memiliki sistem untuk memeriksa kembali *node* untuk mencari *node parent* yang terbaik. Dalam proses pencariannya terbagi menjadi dua, yaitu *closed list* dan *open list.* Dalam pencarian *closed list,* node tidak perlu diperiksa walaupun *node parent* ditemukan, sedangkan dalam pencarian *open list* untuk setiap *node* disekitar parent akan diperiksa. Ketika *parent node* baru mencari semua *node* yang ada disekitarnya, nilai G yang sebelumnya akan disimpan dalam *open list*.



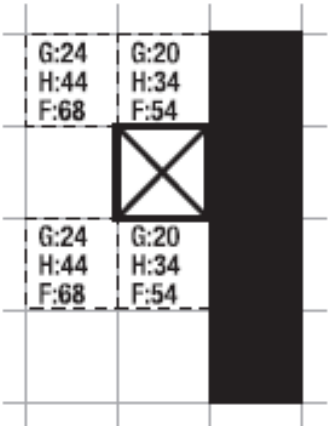
Gambar 2.8 Nilai G yang lama disimpan dalam *open list.*

Untuk mencari nilai G yang baru caranya mengambil nilai G yang baru dari *parent node* kemudian ditambah 10. Karena nilai G yang baru lebih besar dari nilai G yang sebelumnya maka *parent node* tidak berubah.



Gambar 2.9 Nilai G dari *parent node* yang baru

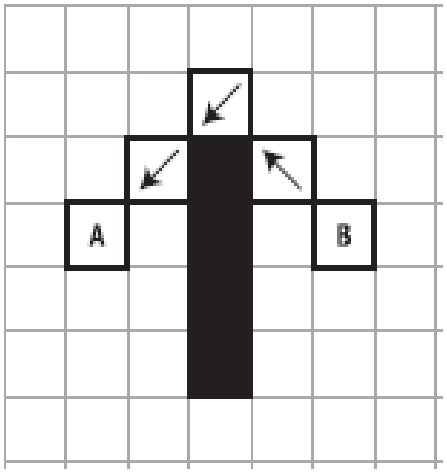
Karena nilai G yang baru lebih besar dari nilai G yang sebelumnya maka *parent node* tidak berubah.



Gambar 2.10 Hasil biaya dari *node* yang baru

Karena tidak adanya *node* yang berubah maka jalur yang dicari tidak berhasil, algoritma A* akan terus mencari sampai menemukan *node* berikutnya yang mempunyai nilai F terkecil.

Setelah menemukan *node* baru kemudian akan dihubungkan dengan *node parent* sehingga akan membentuk sebuah jalur.



Gambar 2.11 Hasil pencarian algoritma A*

Ketika sudah sampai ke titik tujuan, maka proses pencarian akan dihentikan, kemudian titik A akan bergerak mengikuti jalur yang telah dicari menuju titik tujuan.

2.3.1 Kompleksitas Algoritma A*

Kompleksitas waktu dari A* sangat bergantung dari *heuristic* yang digunakannya. Pada kasus terburuk, jumlah titik yang diperiksa berjumlah eksponensial terhadap panjang solusi (jalur terpendek), tetapi A* akan memiliki kompleksitas waktu polinomial apabila fungsi memenuhi kondisi berikut :

$$|h(x) - h*(x)| \leq O(\log h*(x)) \qquad (6)$$

dimana h* adalah *heuristic* optimal, yaitu ongkos sebenarnya dari jalur x ke tujuan. Dengan kata lain, galat dari h tidak akan melaju lebih cepat dari logaritma *heuristic* optimal h* yang memberikan jarak sebenarnya dari x ke tujuan [11].

2.4 Flash

Pada tahun 1980-an, ketika SMA Jonathan Gay menggunakan komputer Apple II yang sudah tua umurnya untuk membuat program gambar komputer. Dia mengajukan dan mengikutsertakan program yang dikenal sebagai SuperPaint dalam pameran sains sekolah. Setelah dia menang, program buatannya menarik perhatian bagi pengembang software lokal Charlie Jackson. Jackson sedang dalam proses memulai sebuah perusahaan yang dikenal sebagai Silicon Pantai Software, yang akan menghasilkan program untuk jenis komputer Macintosh. Dengan sedikit dana yang tersedia untuk mempekerjakan para pengembang berpengalaman, Jackson Gay disewa untuk membantu menciptakan program-programnya. Pada tahun 1993 Jonathan Gay, Charlie Jackson, dan Michelle Welsh mengembangkan SmartSketch. Aplikasi tersebut dibangun untuk menggambar dengan mode vektor. Adobe Flash dahulu dikembangkan oleh Macromedia dan dikenali dengan nama Macromedia Flash. Flash pertama kali 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 adalah Macromedia Flash 8. Pada tanggal 3 Desember 2005 Adobe Systems mengakuisisi Macromedia dan seluruh produknya, sehingga nama Macromedia Flash berubah menjadi Adobe Flash.

Hingga saat ini flash banyak diaplikasikan untuk membuat animasi, iklan, multimedia interaktif, integrasi video pada halaman web, dan sekarang banyak dikembangkan untuk memperkaya aplikasi yang berjalan di internet.

2.5 Actionscipt

Actionscript adalah suatu *programming language* (bahasa pemograman) yang dapat ditambahkan pada flash *document* (baik itu pada *frame*, *movie clip*, atau *button*) untuk dapat membuat suatu animasi yang lebih interaktif. ActionScript 1.0 rilis pertama kali pada Flash 5 yang merupakan pengembangan "*Action*" di Flash 4, dan masih digunakan hingga Flash MX (Flash 6). ActionScript 2.0 kemudian dirilis pada Flash MX 2004 (Flash 7), Flash 8, sampai Flash CS3 (Flash 9). Sebagai generasi penerus ActionScript 1.0 dengan kelebihannya dari kemampuan *compile time checking, strict-typing* pada variabel, *class-based syntax* (yang sebelumnya *prototype based*).

ActionScript 3.0 akhirnya digunakan pada Flash CS3 (Flash 9), Flex 2, Flex 3. ActionScript versi 3.0 ini berupa sebuah restrukturisasi fundamental dari model pemrograman sebelumnya. Penggunaannya yang luas terutama dalam pengembangan RIA (*Rich Internet Application*) dengan hadirnya Flex yang menawarkan hal serupa seperti AJAX, JavaFX, dan Microsoft Silverlight.

2.6 *Unified Modeling Language* (UML)

Unified Modeling Language (UML) adalah bahasa spesifikasi standar untuk mendokumentasikan, menspesifikasikan, dan membangun sistem perangkat lunak. UML tidak berdasarkan pada bahasa pemrograman tertentu. Standar spesifikasi UML dijadikan standar defacto oleh OMG (*Object Management Group*) pada tahun 1997. UML yang berorientasikan *object* mempunyai beberapa notasi standar. Spesifikasi ini menjadi populer dan standar karena sebelum adanya UML, telah ada berbagai macam spesifikasi yang berbeda. Hal ini menyulitkan

komunikasi antar pengembang perangkat lunak. Untuk itu beberapa pengembang spesifikasi yang sangat berpengaruh berkumpul untuk membuat standar baru. UML dirintis oleh Grady Booch, James Rumbaugh dan Ivar Jacobson. Komponen yang terdapat pada UML, diantaranya:

1. *Use case diagram*

   Use case diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah "apa" yang diperbuat sistem, dan bukan "bagaimana". Sebuah *use case* merepresentasikan sebuah interaksi antara aktor dengan sistem. *Use case* merupakan sebuah pekerjaan tertentu, misalnya *login* ke sistem, meng-*create* sebuah daftar belanja, dan sebagainya. Seorang/sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu.

2. *Sequence diagram*

   Sequence diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, *display*, dan sebagainya) berupa *message* yang digambarkan terhadap waktu. *Sequence* diagram terdiri atas dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait).

3. *Collaboration diagram*

   Collaboration diagram juga menggambarkan interaksi antar objek seperti *sequence* diagram, tetapi lebih menekankan pada peran masing-masing objek dan bukan pada waktu penyampaian *message*. Setiap *message*

memiliki *sequence number*, di mana *message* dari level tertinggi memiliki nomor 1. *Messages* dari level yang sama memiliki prefiks yang sama.

4. *Statechart diagram*

   Statechart diagram menggambarkan transisi dan perubahan keadaan (dari satu *state* ke *state* lainnya). Suatu objek pada sistem sebagai akibat dari *stimuli* yang diterima. Pada umumnya *statechart* diagram menggambarkan *class* tertentu (satu *class* dapat memiliki lebih dari satu *statechart diagram*).

5. *Activity diagram*

   Activity diagram menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi. Diagram aktivitas mendeskripsikan aksi-aksi dan hasilnya. Diagram aktivitas berupa operasi-operasi dan aktivitas-aktivitas di *use-case*.

6. *Class diagram*

   Class adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek. *Class diagram* menggambarkan struktur dan deskripsi *class package* dan objek beserta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi, dan lain-lain. *Class diagram* menunjukkan aspek statik sistem terutama untuk mendukung kebutuhan

fungsional sistem. Kebutuhan fungsional berarti layanan-layanan yang harus disediakan sistem ke pemakai.

7. *Object diagram*

   Sebuah diagram obyek dalam *Unified Modeling Language* (UML), adalah diagram yang menunjukkan suatu pandangan lengkap atau sebagian dari struktur system dimodelkan pada waktu tertentu. Sebuah diagram obyek berfokus pada beberapa set tertentu variabel dan atribut, dan hubungan antara variabel. Satu set berkorelasi diagram objek memberikan wawasan tentang bagaimana pandangan sewenang-wenang sistem diharapkan berkembang dari waktu ke waktu. Obyek diagram lebih konkret dari diagram kelas, dan sering digunakan untuk memberikan contoh-contoh, atau bertindak sebagai uji kasus untuk diagram kelas. Hanya aspek-aspek dari model yang menarik saat ini perlu ditunjukkan pada diagram objek.

8. *Component diagram*

   Component diagram menggambarkan struktur dan hubungan antar komponen piranti lunak, termasuk ketergantungan (*dependency*) di antaranya. Komponen piranti lunak adalah modul berisi *code*, baik berisi *source code* maupun *binary code*, baik library maupun *executable*, baik yang muncul pada *compile time*, *link time*, maupun *runtime*. Umumnya komponen terbentuk dari beberapa *class* atau *package*, tapi dapat juga dari komponen-komponen yang lebih kecil. Komponen dapat juga berupa *interface*, yaitu kumpulan layanan yang disediakan sebuah komponen untuk komponen lain.

9. *Deployment diagram*

   Deployment/*physical* diagram menggambarkan detail bagaimana komponen di-*deploy* dalam infrastruktur sistem, di mana komponen akan terletak (pada mesin, server atau piranti keras apa), bagaimana kemampuan jaringan pada lokasi tersebut, spesifikasi server, dan hal-hal lain yang bersifat fisikal. Sebuah *node* adalah server, *workstation*, atau piranti keras lain yang digunakan untuk men-*deploy* komponen dalam lingkungan sebenarnya.

2.7 Adobe Photoshop

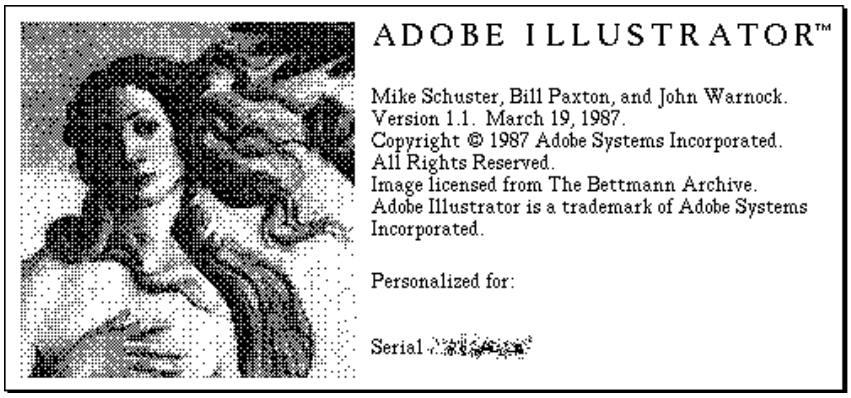
Pada akhir 1980-an, *Thomas Knoll* dan *John Knoll* mengembangkan sebuah pemrograman perisian grafik yang bernama program *pixel imaging* dibuat pertama kali, kemudian Image Pro, dan akhirnya Photoshop. Mereka membuat suatu persembahan diterima dengan baik kepada Russell Brown di Syarikat Adobe dan kemudian Adobe Photoshop dilesenkan pada tahun 1988. Bermulalah perkembangan suatu sejarah yang dikenali sebagai Adobe Photoshop.

Adobe Photoshop, atau biasa disebut Photoshop, adalah penyunting grafik yang dikembangkan dan diterbitkan oleh Adobe System yang diciptakan untuk penyuntingan foto/gambar dan pembuatan kesan gambar. "Photoshop" bermaksud "kedai foto" dalam Bahasa Inggris. Perisian ini banyak digunakan oleh juru gambar digital dan perusahaan iklan sehingga dianggap sebagai pasaran utama untuk perisian pengolahan gambar, dan bersama Adobe Acrobat, dianggap sebagai produk terbaik yang pernah dihasilkan oleh Adobe Systems. Versi kelapan aplikasi ini disebut dengan nama Photoshop CS, versi kesembilan disebut

Photoshop CS2, Photoshop CS3, Photoshop CS4 dan versi terbaru yang akan dilancarkan April 2010 ini adalah Photoshop CS5.

2.8 *Adobe Illustrator*

Adobe Illustrator pertama kali dibuat pada tahun 1986, sebagai aplikasi pembuat desain grafis, huruf, dan logo. Sebenarnnya Adobe Illustrator diciptakan dengan tujuan menggabungkan persamaan matematika untuk menciptakan garis halus dan melengkung serta bentuk objek yang dikenal sebagai kurva Bezier. Untuk menekankan gagasan yang mampu memproduksi dan mengedit dan menyimpan garis lengkung dan beralur, Adobe memilih gambar "The Birth of Venus" dari Sandro Botticelli untuk menjadi *cover* pemasaran mereka. Selama bertahun-tahun dan beberapa versi, Adobe menambahkan fitur-fitur baru dan perbaikan.



Gambar 2.12 Logo Adobe Illustrator[1]

Adobe Illustrator adalah produk pedamping dari Adobe Photoshop. Jika Photoshop ditujukan untuk memanipulasi foto digital dan gaya foto realistik ilustrasi komputer, sedangkan Illustrator ditujukan untuk kreasi logo, huruf dan desain grafis.

Aplikasi ini banyak digunakan oleh desainer profesional, mulai desainer iklan, majalah, cover, *banner* sampai desainer web. Adobe Illustrator mendukung format RGB (*Red, Green, Blue*) dan CMYK (*Cyan, Magenta, Yellow, Key - Black*) yang biasa digunakan untuk proses pencetakan.