BAB 2

LANDASAN TEORI

2.1. Permainan Video (Video *Game*)

Permainan Video (Video *Game*) merupakan sebuah permainan elektronik yang di dalamnya melibatkan interaksi manusia dengan user-*interface* sebagai pemain dengan menggunakan perangkat video. System elektronik yang digunakan untuk memainkan permainan video dinamakan platform, contoh dari platform adalah Personal Computer (PC) dan konsol video *game*.

Perangkat input yang digunakan untuk menggerakkan Video *Game* disebut *game* controller. Controller sangat bervariasi pada setiap platform misalnya, pada konsol video *game* biasanya terdiri dari tombol dan joystick. Pada awal permainnan komputer sering menggunakan keyboard dan mouse untuk *Gameplay*, atau lebih umum diperlukan untuk membeli joystick secara terpisah.

2.1.1. Sejarah Singkat Video *Game*

Video *game* pertama kali di temukan pada tahun 1958 oleh William Higinbotham[10] seorang kepala pada bagian Instrumentation division for Brookhaven National Laboratory. William adalah orang pertama yang mendesain dan menciptakan permainan sangat sederhana berbentuk tenis menggunakan osiloskop sebagai *interface*. Dijalankan pada komputer analog dan dihubungkan ke osiloskop, permainan ini terlihat jelas dan cepat, yang mengejutkan adalah permainan ini tidak hanya top-down perspective seperti Pong, namun menampilkan dua sudut seperti lapangan tenis. Dua pemain harus melakukan pukulan terhadap bola secara bergantian dengan bola yang memantul ke tanah dan net searah gravitasi.

Pada tahun 1961, seorang siswa MIT bernama Steve, dia menciptakan permainan video two-player yang dinamakan Spacewar pada sebuah komputer DEC PDP-1[8]. Russell butuh hampir enam bulan dan 200 jam untuk menyelesaikan versi pertama permainan: duel sederhana antara kapal roket.

Tujuan dari permainan tersebut adalah setiap pemain melakukan maneuver pesawatnya ketika mencoba menembak pesawat pemain lain dengan menggunakan misil Menggunakan switch toggle dibangun ke dalam PDP-I, pemain mengendalikan kecepatan dan arah dari kedua kapal dan menembakkan torpedo satu sama lain.

Video *game* pertama diciptakan oleh para insinyur di Sanders Associates, sebuah Baru Hampshire berbasis kontraktor pertahanan. Seperti kontraktor besar banyak, Sanders memiliki pangsa kegiatan sensitif dan rahasia. Namun pada tahun 1967, beberapa suara yang keluar dari salah satu Sanders penelitian laboratorium memiliki banyak orang bertanya-tanya apa yang sedang terjadi.

Pada awal tahun 1966, seorang enginer muda bernama Ralph Baer, mengembangkan kembali idenya dengan membawa permainan video pada media sekunder untuk televisi. Baer dan timnya berhasil menciptakan sebuah *game* interaktif yang dapat dimainkan di layar televisi. Mereka mengembangkan permainan mengejar dan mengikutinya dengan permainan tenis video. Mereka juga memodifikasi pistol mainan sehingga dapat membedakan bintik-bintik cahaya di layar.

Pada tahun 1971, Magnavox menyewa anggota tim RCA yang proyeknya hampir dibeli. Kemudian mengatakan kepada para eksekutif Magnavox lainnya tentang permainan televisi yang dilihatnya di Sanders. Magnavox mengatur untuk demonstrasi dari permainan televisi dan segera melihat manfaat dalam idenya itu. Setelah berbulan-bulan tim bekerja secara detail, negosiasi telah diselesaikan dan kontrak ditandatangani pada akhir tahun ini. Produksi dimulai pada musim gugur, dan unit awal yang ditampilkan di dealer Magnavox pada tahun 1972. Produk yang sudah selesai itu disebut Magnavox Odyssey.

Pada tahun 1983, sebuah pembaruan besar-besaran terjadi pada industri permainan video yang mengakibatkan hal serius terhadap pasar pada saat itu. Ada beberapa faktor utama yang menyebabkan terjadinya crash, diantaranya : ekonomi yang miskin, siklus hidup alami pasar, dan persepsi pemakai yang hanya untuk main-main saja.

Pada tahun selanjutnya, berbagai perusahan pembuat permainan video berlomba-lomba untuk menciptakan perangkat dan permainan bagi para penggemar permainan video dan merupakan awal dari kebangkitan permainan video hingga saat ini.

2.1.2. Jenis-jenis permainan video

Dalam permainan video terdapat beberapa jenis permainan yang dapat dimainkan, hal ini dibedakan berdasarkan cara bermain yang dapat dilakukan oleh para pemain.

A. *ACTION Game*

Permainan *ACTION* adalah permainan yang memberikan intensitas aksi sebagai daya tarik utama. Respon refleks adalah keterampilan utama yang dibutuhkan untuk memainkan permainan ini dengan baik[19]. *Game ACTION* yang paling umum adalah shooter (Doom) dan stealth (Metal Gear). *Game ACTION* juga termasuk dalam kategori *SPORT*s, meskipun beberapa kategori *SPORT*s lebih sering jatuh ke dalam kategori simulasi. Contoh *Game ACTION* seperti pada gambar 2.1.



Gambar 2.1. Contoh permainan *ACTION Game*[19]

B. *Platformer*

Permainan platform asli terdiri dari karakter yang berlari dan melompat pada arena bermain dengan mode side-scrolling[19]. Beberapa contoh dari *Platformer*s yang terkenal diantaranya Super Mario Bros, Sonic the Hedgehog, Pitfall!, dan Super Mario 64, seperti pada gambar 2.2.



Gambar 2.2. Contoh permainan *Platformer*s[19]

C. *ADVENTURE*

Permainan *ADVENTURE* adalah permainan yang menawarkan eksplorasi dan memecahkan teka-teki sebagai daya tarik utama[19]. Permainan ini menawarkan histori cerita yang paling mengasyikkan, meskipun popularitas mereka telah menurun dalam 2 dekade terakhir. Penalaran, kreativitas, dan rasa ingin tahu adalah keterampilan yang paling umum yang diperlukan dari pemain *game* petualangan yang baik. Contoh dari *game ACTION-ADVENTURE* adalah Assassin Creed seperti pada gambar 2.3.



Gambar 2.3. Contoh permainan *ADVENTURE*[19]

D. *FIGHTING*

Permainan *FIGHTING* adalah permainan yang bergenre dimana pemain mengontrol karakter di layar dan terlibat dalam pertempuran jarak dekat dengan lawan[19]. Karakter ini cenderung memiliki kekuatan yang seimbang dan ronde dari pertandingan ini terdiri dari beberapa putaran, yang berlangsung di arena. Pemain harus menguasai teknik seperti memblokir, counter-attacking, dan chaining bersama-sama urutan serangan yang dikenal sebagai "combo". Contoh dari permainan video yang berjenis *FIGHTING* yang plaing terkenal diataranya Double Dragon, Street Fighter, dan Mortal Kombat, seperti pada gambar 2.4.



Gambar 2.4. Contoh permainan *FIGHTING*[19]

E. *FIRST-PERSON SHOOTER*

FIRST-PERSON SHOOTER (FPS) merupakan sebuah *ACTION game* yang menempatkan pemain sudut pandang orang pertama,yaitu si pemain itu sendiri pada karakter permainan[19]. Karakter pada layar monitor tidak akan terlihat dalam *game* FPS, biasanya yang akan terlihat pada layar monitor biasanya hanya tangan yang memegang senjata. permainan ini, pemain diijinkan untuk memilih berbagai senjata dan membunuh musuh dengan menembaki mereka. Salah satu contoh dari jenis permainan ini yang terkenal adalah Doom dan counter strike, seperti pada gambar 2.5.



Gambar 2.5. Contoh dari permainan *FIRST-PERSON SHOOTER*[19]

F. *REAL-TIME STRATEGY* (RTS)

Permainan *REAL-TIME STRATEGY* mempunyai tujuan yaitu Strategi permainan yang menghibur pemain melalui penalaran dan pemecahan masalah[19]. Awal permainan strategi (romance of three kingdom) tidak menggunakan cerita yang banyak, meskipun *game* yang lebih baru sangat bergantung pada narasi berkualitas. Permainan seperti Command and Conquer adalah contoh dari *game* strategi berbasis cerita. Dalam *game REAL-TIME STRATEGY* biasanya pemain di haruskan untuk mengoleksi berbagai sumber daya, membangun pasukan, dan mengontrol pasukannya untuk menyerang musuh. Aksi terhadap permainan ini adalah fast-paced dan arena merupakan permainan yang berlanjut, maka keputusan untuk strategi harus dibuat secara cepat. Beberapa contoh dari permainan ini diantaranya The Ancient Art of War, Command and Conquer, dan Age Of Empires, seperti pada gambar 2.6.



Gambar 2.6. Contoh permainan *REAL-TIME STRATEGY* (RTS)[19]

G. *ROLE-PLAYING GAME* (RPG)

Permainan *ROLE-PLAYING GAME* merupakan permainan yang menawarkan pemain kesempatan untuk inspirasikan diri dalam situasi karakter pemain[19]. *ROLE-PLAYING GAME* (RPG) membuat permainan menjadi inovatif dan bervariasi dalam menyelesaikan tujuan tertentu. Contoh RPG meliputi, Fire Emblem, Final Fantasy, Luminous Arc, dan World of Warcraft. seperti pada gambar 2.7.



Gambar 2.7. Contoh dari ROLE-PLAYING GAME[19]

H. *SIMULATION*

Permainan simulasi diadaptasikan dari sistem simulasi. Elemen utama dari permainan simulasi adalah kemampuannya untuk mencocokkan situasi seperti di dunia nyata[19]. Simulasi berusaha untuk memberikan kenikmatan melalui pemeragaan. Simulasi peperangan dan simulasi balapan mobil relatif populer dalam jenis permainan. Simulasi juga dapat mencakup simulasi situasi sosial seperti Sims. Salah satu contoh dari permainan ini adalah The Sims, seperti pada gambar 2.8.



Gambar 2.8. Contoh permainan *SIMULATION*[19]

I. *RACING*

Permainan *RACING* adalah genre video *game*, baik dalam sudut pandang orang pertama atau orang ketiga, di mana turut pemain dalam kompetisi balap dengan semua jenis balapan di daratan, udara, atau kendaraan laut. Secara umum. *Game* balap juga dapat jatuh di bawah kategori permainan olahraga[19]. melibatkan kompetisi balap kendaraan mulai dari balap BMX maupun ice skating. Jenis permainan ini sedikit berbeda dibanding permainan lainnya. Salah satu contoh dari permainan ini Need for Speed dan Nascar, seperti pada gambar 2.9.



Gambar 2.9. Contoh permainan *RACING*[19]

J. *SPORTS*

Permainan ini merupakan jenis yang mensimulasi olahraga yang ada pada dunia nyata[19]. Salah satu contohnya adalah Pro Evolution Soccer, seperti pada gambar 2.10.



Gambar 2.10.Contoh permainan *SPORT*[19]

K. *PUZZLE*

Permainan *PUZZLE* merupakan permainan yang menawarkan teka-teki sebagai daya tarik utama untuk *game*[19]. Permainan ini paling sering dirilis pada anggaran yang rendah melalui web. Orang-orang yang bermain *game-game* ini cenderung menjadi populasi tertua dari permainan bermain masyarakat. Salah satu permainan *PUZZLE* paling sukses dan terkenal adalah Tetris, Lemmings dan Minesweeper. I.Q. Cube cerdas adalah lain *PUZZLE*. Contoh *game PUZZLE* seperti pada gambar 2.11.

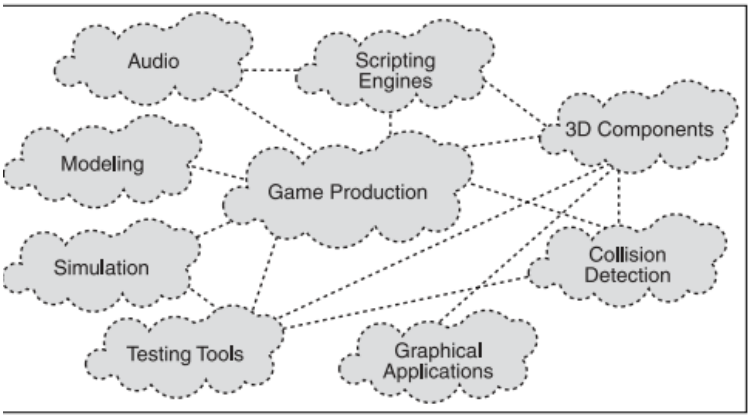


Gambar 2.11.Contoh dari permainan *PUZZLE*[19]

2.1.3. Konsep Permainan Video

Menurut J. P. Flynt dan O. Salem dalam bukunya[3], terdapat beberapa syarat yang dibutuhkan dalam membangun sebuah aplikasi permainan (*game* production) sehingga aplikasi permainan yang dibangun memiliki karakteristik tersendiri, diantaranya diterapkannya Audio yang membuat sebuah aplikasi permainan semakin menarik saat dimainkan, sebuah algoritma yang menangani deteksi tubrukan (*collision detection*) antara objek yang satu dengan yang lainnya, 3D/2D komponen yang digunakan, aplikasi yang digunakan untuk melakukan design berbagai objek yang ada pada aplikasi permainan tersebut, dan lain-lain.

Gambaran secara umum tentang apa saja yang dibutuhkan dalam membangun sebuah permainan (*game* production) terdapat pada gambar 2.12.



Gambar 2.12.Spesialisasi peningkatan karakteristik industri *game*[3]

2.1.4. Gameplay Platformer

Platformer adalah video *game* di mana *game*-play bersifat berlari, melompat, mengayun, naik tangga, dan sebagainya terhadap benda lain yang digambarkan pada tampilan layar permainan single atau scrolling-screen. Pertama kali dikembangkan kembali di awal 1980-an, *Platformer*s adalah salah satu genre paling awal dari video *game*. Sementara itu diperdebatkan permainan mana yang benar-benar awal-mula dari genre *Platformer*, jelas bahwa awal dari *game* klasik seperti Donkey Kong, Panic Space, dan Mario Bros yang sangat berpengaruh dan semua memiliki krtiteria dalam membentuk genre *Platformer*[22].

- Single Screen *Platformer*s

  Single Screen *Platformer*, seperti namanya, yang dimainkan pada layar permainan tunggal dan biasanya berisi hambatan yang harus pemain hindari dan tujuannya ialah untuk menyelesaikan permainan. Contoh terbaik dari *Platformer* layar tunggal Donkey Kong, dimana Mario berjalan pada lantai baja sementara menghindari dan melompat barel yang dilemparkan ke arahnya. Setelah tujuan layar tunggal selesai pemain biasanya bergerak ke layar yang berbeda dengan tujuan yang sama atau baru[22].

- Side and Vertical Scrolling *Platformer*s

  Side and *game Platformer* vertical di sisi lain, memiliki layar permainan / latar belakang yang bergerak atau mengikuti pemain secara bersamaan bergerak ke arah tepi layar. *Platformer*s banyak side scrolling juga ditandai dengan memiliki beberapa tingkatan, masing-masing berakhir dengan melawan bos sebelum maju ke tingkat berikutnya. Contoh *Platformer*s side scrolling termasuk *game* klasik seperti Super Mario Bros, Castlevania, Sonic the Hedgehog, dan Pitfall![22]

2.1.5. Karakteristik *Platformer*s

Karakteristik yang terdapat di dalam aplikasi permainan yang berjenis *Platformer*s adalah sebagai berikut :

A. Terdiri dari dua point, yaitu point awal tempat dimana karakter utama memulai pertarungan dan point akhir yang merupakan tujuan akhir permainan.
B. Adanya item yang dapat digunakan untuk membantu menambah live dan item lain yang harus dikoleksi, sehingga pemain bisa melanjutkan ke *level* selanjutnya.
C. Metode interaksi antara pemain dan aplikasi permainan dengan menggunakan keyboard
D. Terdapat live-bar dan jumlah score pemain

E. Terdapat jalur yang telah ditentukan sehingga dapat melewati rintangan dalam berjalan melewati jalur tersebut

2.2. AI (ARTIFICIAL INTELLIGENCE)

2.2.1. Definisi AI

Sebagian kalangan menerjemahkan *ARTIFICIAL INTELLIGENCE* sebagai kecerdasan buatan, kecerdasan artifisial, intelijensia artifisial, atau intelejensia buatan. Pada bukunya[9], Suyanto sengaja tidak menerjemahkan istilah *ARTIFICIAL INTELLIGENCE* ke bahasa Indonesia karena istilah tersebut sudah sangat akrab bagi orang Indonesia. Begitu juga dengan singkatan istilah tersebut, yaitu AI, sudah sangat melekat di berbagai media ilmiah maupun non ilmiah.

Para ahli mendefinisikan AI secara berbeda-beda tergantung pada sudut pandang mereka masing-masing. Ada yang focus pada logika berpikir manusia saja, tetapi ada juga yang mendefinisikan AI secara lebih luas pada tingkah laku manusia. Stuart Russel dan Peter Norvig mengelompokkan definisi AI, yang diperoleh dari beberapa textbook berbeda, ke dalam empat kategori[9], yaitu :

A. Thinking humanly : the cognitive modeling approach

Pendekatan ini dilakukan dengan dua cara sebagai berikut[9] :

1. Melalui introspeksi : mencoba menangkap pemikiran-pemikiran kita sendiri pada saat kita berpikir. Tetapi, seorang psikolog Barat mengatakan “how do you know that you understand?” Bagaimana anda tahu bahwa anda mengerti? Karena pada saat anda menyadari pemikiran anda. Sehingga definisi ini terkesan mengada-ngada dan tidak mungkin dilakukan.
2. Melalui eksperimen-eksperimen psikologi.

B. Acting humanly : the Turing test approach

Pada tahun 1950, Alan Turing merancang suatu ujian bagi komputer berintelijensia untuk menguji apakah komputer tersebut mampu mengelabui seorang manusia yang menginterogasinya melalui teletype (komunikasi berbasis teks jarak jauh). Jika interrogator tidak dapat membedakan yang d2nterogasi adalah manusia atau komputer, maka komputer berintelijensia tersebut lolos dari Turning 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 intelijensia.

C. Thinking rationally : the laws of thought approach

Terdapat dua masalah dalam pendekatan ini, yaitu [9]:

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”.

D. 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.

Thinking humanly dan acting humanly adalah dua definisi dalam arti yang sangat luas. Sampai saat ini, pemikiran manusia yang diluar rasio, yakni reflex dan intuitif (berhubungan dengan perasaan), belum dapat ditirukan sepenuhnya

oleh komputer. Dengan demikian, kedua definisi ini dirasa kurang tepat untuk saat ini. Jika kita menggunakan definisi ini, maka banyak produk komputasi cerdas saat ini yang tidak layak disebut sebagai produk AI.

Definisi thinking rationally terasa lebih sempit daripada acting rationally. Oleh karena itu, definisi AI yang paling tepat untuk saat ini adalah acting rationally dengan pendekatan rational agent. Hal ini berdasarkan pemikiran bahwa komputer bisa melakukan penalaran secara logis dan juga bisa melakukan aksi secara rasional berdasarkan hasil penalaran tersebut.

2.2.2. Teknik Pemecahan Masalah AI

Terdapat empat teknik dasar pemecahan masalah yang terdapat pada bidang academic/traditional *ARTIFICIAL INTELLIGENCE* (AI)[9], diantaranya adalah sebagai berikut :

A. Searching

Pada teknik searching atau pencarian ini terdiri dari beberapa langkah untuk merealisasikannya. Langkah pertama adalah mendefinisikan ruang masalah untuk suatu masalah yang dihadapi. Langkah kedua adalah mendefinisikan aturan produksi yang digunakan untuk mengubah suatu keadaan ke keadaan lainnya. Langkah terakhir adalah memilih metode pencarian yang tepat sehingga dapat menemukan solusi terbaik.

Metode-metode pencarian pada teknik searching diantaranya[9] :

1. Blind/Un-informed Search
    a. Breadth-First Search (BFS)
    b. Depth-First Search (DFS)
    c. Depth-Limited Search (DLS)
    d. Uniform Cost Search (USC)
    e. Iterative-Deepening Search (IDS)
    f. Bi-Directional Search (BDS)

2. Metode Pencarian Heuristik
    a. Generate-and-Test
    b. Hill Climbing
    c. Simulated Annealing
    d. Best-First Search (BFS)
    e. Greedy Best-First Search
    f. A* (*A star*)

B. Reasoning

Teknik reasoning atau penalaran merupakan teknik penyelesaian masalah dengan cara merepresentasikan masalah ke dalam basis pengetahuan (knowledge base) menggunakan logic atau bahasa formal (bahasa yang dipahami komputer). Teknik ini melakukan proses penalaran berdasarkan basis pengetahuannya untuk menemukan solusi.

C. Planning

Planning adalah suatu metode penyelesaian masalah dengan cara memecah masalah ke dalam sub-sub masalah yang lebih kecil, menyelesaikan sub-sub masalah satu demi satu kemudian menggabungkan solusi-solusi dari sub-sub masalah tersebut menjadi sebuah solusi lengkap dengan tetap mengingat dan menangani interaksi yang terdapat pada sub-sub masalah tersebut.

D. Learning

Pada ketiga teknik sebelumnya, seseorang harus mengetahui aturan yang berlaku untuk sistem yang akan dibangunnya. Tetapi, pada masalah tertentu terkadang suatu aturan tidak bisa didefinisikan secara benar ataupun lengkap. Hal tersebut mungkin dikarenakan data-data yang didapat tidak lengkap. Melalui teknik yang disebut learning ini, secara otomatis aturan yang diharapkan bisa berlaku umum untuk data-data yang belum pernah diketahui dapat ditemukan.

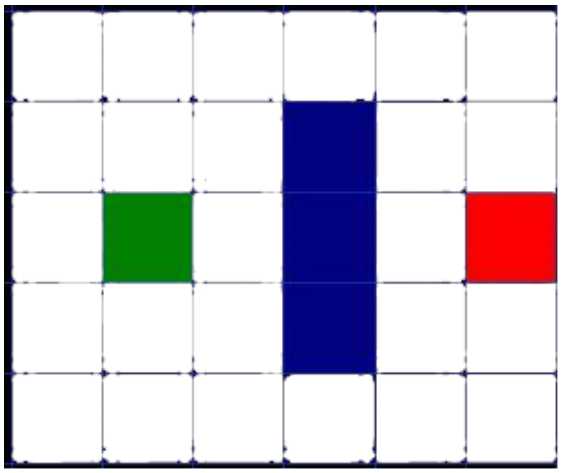
2.2.2.1. Algoritma A* (*A star Algorithm*)

Algoritma A* (*A star*) merupakan algoritma Best First Search yang menggabungkan Uniform Cost Search dan Greedy Best-First Search[13]. Biaya yang diperhitungkan didapat dari biaya sebenarnya ditambah dengan biaya perkiraan, dalam notasi matematika dituliskan sebagai berikut :

$$f(n) = g(n) + h(n).....[13]$$

Dengan perhitungan biaya seperti diatas, algoritma A* adalah complete dan optimal[13]. Untuk lebih jelasnya berikut merupakan gambaran dari penggunaan algoritma A* dalam pencarian rute terpendek dari point A menuju point B :

Asumsikan ada seseorang yang pergi dari point A menuju point B dan rute yang akan dilewati dihalangi sebuah dinding. Pada ilustrasi ini kotak berwarna hijau merupakan point A, kotak berwarna merah merupakan point B, dan kotak berwarna biru merupakan dinding yang menghalangi rute point A menuju point B[18], untuk lebih jelasnya terdapat pada gambar 2.13.



Gambar 2.13.Ilustrasi awal algoritma A*[18]

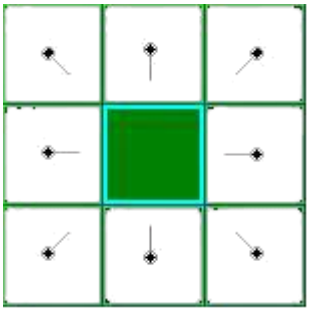
Hal pertama yang harus dilakukan adalah membuat sebuah grid layout, karena dalam pencarian rute pada algoritma A* akan lebih mudah apabila peta yang ada berbentuk kotak-kotak kecil, metode ini memudahkan pencarian dalam area yang didefinisikan sebagai array dua dimensi, setiap kotak memiliki status walkable dan unwalkable.

Pusat point pada algoritma A* dinamakan node, hal ini dikarenakan area yang dapat digunakan algoritma A* tidak hanya berbentuk kotak, akan tetapi dapat berbentuk yang lainnya, namun terkadang untuk memudahkan dalam perhitungan maka digunakanlah bentuk kotak.

Langkah-langkah yang dilakukan pada algoritma A* dalam melakukan pencarian rute terpendek dibagi menjadi beberapa tahap, yaitu :

A. Dimulai dari point A awal dan masukkan ke dalam kotak "*OPEN LIST*". "*OPEN LIST*" merupakan sebuah kotak yang berfungsi untuk menyimpan setiap nod, baik yang akan kita lewati maupun yang tidak kita lewati, pada dasarnya setiap kotak yang ada pada "*OPEN LIST*" akan dilakukan perhitungan jarak untuk menentukan arah mana yang akan diambil menuju point B.
B. Lihat semua kotak yang memungkinkan untuk dilewati dari point awal, abaikan kotak yang terdapat penghalang seperti tembok, air, dan lain-lain. Masukan kotak-kotak tersebut ke dalam *OPEN LIST*, dan untuk setiap kotak ini, simpan point A sebagai induk kotak.
C. Keluarkan kotak point awal dari *OPEN LIST* kemudian masukkan ke dalam *CLOSED LIST*. *CLOSED LIST* berfungsi untuk menyimpan kotak yang tidak akan pernah dipakai ataupun dilihat lagi.

Kotak berwarna hijau merupakan pusat dari point awal, pada sisinya dikelilingi oleh garis berwarna biru yang berarti bahwa kotak tersebut telah dimasukkan ke dalam *CLOSED LIST*, dan untuk setiap garis disekeliling kotak awal menunjukkan setiap jalur yang dapat ditempuh dimulai dari kotak pada point awal, perhatikan pada gambar 2.14.



Gambar 2.14.Ilustrasi kotak awal sebagai *CLOSED LIST*[18]

Perhitungan yang terdapat pada algoritma A* untuk melakukan pencarian rute terpendek adalah dengan menggunakan rumus sebagai berikut :

$$F = G + H.....[18]$$

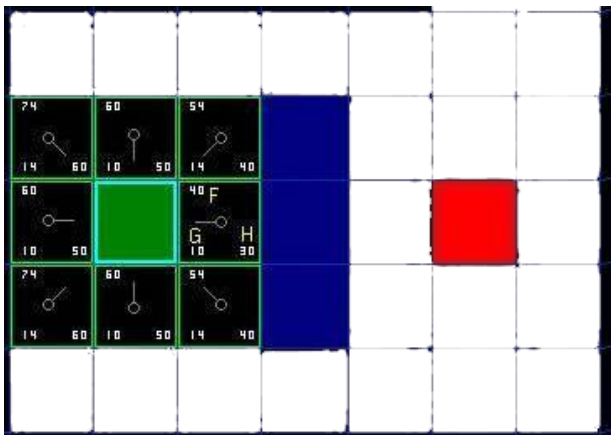
dimana,

G : nilai yang dibutuhkan untuk menempuh jarak dari point awal menuju kotak selanjutnya

H : estimasi nilai yang dibutuhkan untuk menempuh jarak dari kotak selanjutnya untuk menempuh point tujuan.

Dalam proses perhitungannya dapat dimisalkan sebuah nilai untuk G baik yang terdapat pada kotak horizontal, vertikal, dan diagonal. Pada kasus ini dimisalkan nilai horizontal dan vertikal dari point awal untuk menuju kotak selanjutnya ialah 10, dan untuk nilai diagonalnya ialah 14.

Setiap kotak memiliki nilai F, G, dan H masing-masing, jadi perhitungan tersebut berlaku bagi setiap kotak yang berada di sekitar kotak induk, untuk nilai F ditulis di atas sebelah kiri, nilai G ditulis di bawah sebelah kiri, dan nilai H ditulis di bawah sebelah kanan, seperti pada gambar 2.15.



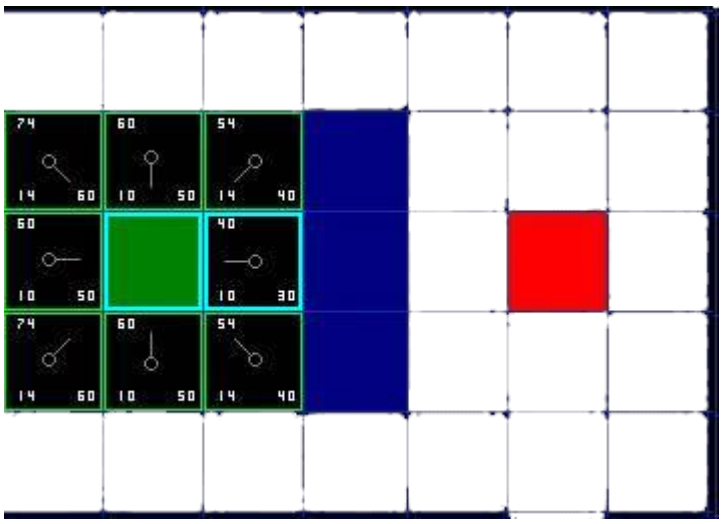
Gambar 2.15.Ilustrasi perhitungan F pada *OPEN LIST*[18]

Langkah selanjutnya yang harus dilakukan adalah sebagai berikut :

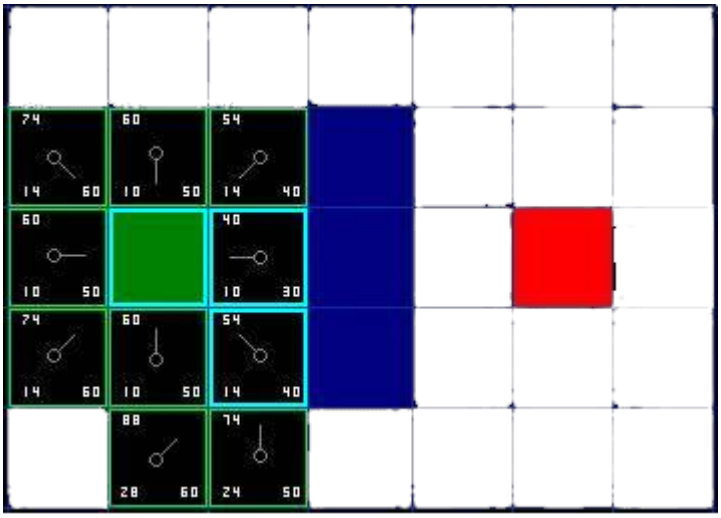
D. Pilih nilai F yang terkecil dari setiap kotak yang telah dihitung nilai F nya, kemudian keluarkan dari *OPEN LIST* dan masukkan ke dalam *CLOSED LIST*.

E. Cek semua kotak yang berada di sekitar, abaikan kotak yang unwalkable dan kotak yang terdapat pada *CLOSED LIST*, masukkan kotak yang belum ada di dalam *OPEN LIST* dan jadikan kotak selanjutnya menjadi kotak induk untuk melakukan perhitungan ulang untuk setiap kotak disekitarnya.

F. Ketika sebuah kotak sekitar telah berada di dalam *OPEN LIST*, cek apakah jalur kotak tersebut adalah yang terbaik dibanding yang jalur kotak yang lainnya. Di lain pihak, ketika nilai G untuk jalur yang baru itu paling kecil, ganti induk kotak pada kotak sekitarnya menjadi kotak yang telah terpilih, dan terakhir hitung ulang nilai F, G, dan H untuk kotak tersebut, seperti pada gambar 2.15 dan perhitungan selanjutnya pada gambar 2.16.

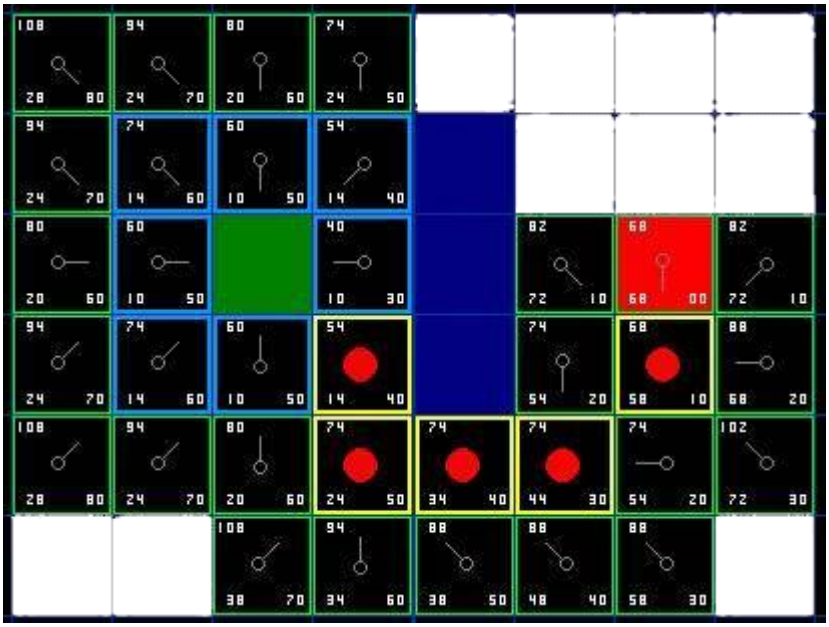


Gambar 2.16.Ilustrasi penentuan kotak selanjutnya[18]



Gambar 2.17.Ilustrasi perhitungan terhadap kotak unwalkable[18]

Setelah mengulangi setiap langkah yang ada di atas, perhitungan dapat dihentikan ketika point akhir atau yang disebut kotak target telah dimasukkan ke dalam *CLOSED LIST*, perhatikan gambar 2.18.

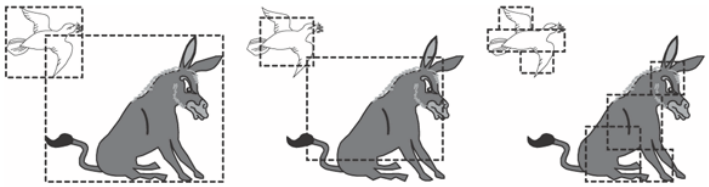


Gambar 2.18.Ilustrasi hasil akhir pencarian rute terpendek[18]

2.2.3. Metode Collision detection

Hampir setiap permainan video yang ada sudah menerapkan *collision detection* (deteksi tabrakan), baik itu dalam hal tabrakan antara sprite dengan sprite maupun antara sprite dengan peluru dan lain-lain. Dalam aplikasi permainan ada beberapa reaksi yang ditunjukkan dari *collision detection*, yaitu reaksi dari spirte yang berupa berkurangnya nyawa, berkurangnya darah, meledak, dan lain-lain.

Proses *collision* dapat dibagi menjadi dua kategori dasar, yaitu *collision detection* dan *collision* response[4], dengan jarak respon yang telah diaplikasikan secara spesifik. Dalam *collision detection*, terdapat banyak sekali jenis dari *collision detection* itu sendiri, pada gambar 2.19 merupakan tiga tipe dari *collision detection*.



Gambar 2.19.Tiga jenis collision detection[2]

Setiap objek diberi sebuah kotak sebagai acuan jika *collision* itu terjadi. Untuk gambar yang pertama, *collision detection* terjadi ketika objek buruh bertabrakan dengan kotak yang memuat objek keledai, pada jenis *collision detection* ini rentan terhadap ketidaktelitian[2], untuk gambar yang kedua, *collision detection* terjadi ketika kotak dari kedua objek tersebut bersinggungan, pada jenis *collision detection* ini sudah lebih akurat dibanding jenis yang pertama, namun untuk memilih factor reduksi yang cocok itu sulit[2], dan untuk gambar yang ketiga, *collision detection* akan terjadi ketika objek saling bersinggungan tanpa memperdulikan kotak secara keseluruhan, pada jenis *collision detection* ini adalah yang paling akurat dibandingkan dengan jenis yang lainnya[2].

Selain itu pada sumber lain[4] disebutkan bahwa *collision detection* terbagi menjadi dua jenis, yaitu *collision detection* object to object dan *collision*

detection object to world. Pada object to object yang terjadi adalah adanya tubrukan antara satu objek dengan objek lainnya yang berada di dalam satu frame, baik itu objek tersebut bertubrukan secara langsung ataupun bertubrukan antara kotak maupun lingkaran yang mengandung objek tersebut, sedangkan pada object to world yang terjadi adalah adanya tubrukan antara objek dengan frame itu sendiri, contohnya adalah tubrukan antara sprite dengan dinding pada aplikasi permainan itu sendiri sehingga apabila terjadi, maka sprite tersebut tidak dapat melewati dinding tersebut.

2.2.4. Object Oriented Programming (OOP)

Object Oriented Programming (OOP) menerapkan sifat yang lebih modular agar setiap program dapat lebih mudah dikembangkan. Dalam OOP dibutuhkan memori lebih besar dibandingkan dengan program procedural (tradisional)[6]. Dua objek yang identik akan memerlukan dua area memori berbeda walaupun dari sisi data dan proses keduanya memiliki jumlah dan jenis yang sama. Hal ini disebabkan karena data dan proses pada kedua objek tersebut dipisahkan oleh komputer.

Secara garis besar yang menjadi ciri dari OOP adalah adanya proses abstraksi (abstr*ACTION*), pengkapsulan (encapsulation), penurunan sifat (inheritance), dan polimorfisme (polymorphism) pada objek-objek yang dibentuk.

Object Oriented Programming (OOP) dibagi menjadi beberapa cirri utama[6], yaitu:

A. Kelas

Kelas (*Class*) merupakan contoh abstrak dari sebuah objek yang telah terbentuk dari proses penyederhanaan, dengan kata lain kelas (*Class*) merupakan cikal bakal dari objek (object), kemudian contoh nyata atau perwujudan dari sebuah objek dinamakan instance. Sehingga apabila kita mempunyai sebuah kelas manusia, maka beberapa instances (wujud nyata) dari kelas manusia adalah Prima, Aulia, Dewi, dan masih banyak yang lainnya.

Perbedaan antara kelas (*Class*) dengan objek (object) dalam OOP dibagi menjadi dua[6], yaitu :

1. *Class* merupakan rancangan (design) dan object merupakan perwujudan dari suatu *Class*.
2. *Class* bersifat abstrak sementara object bersifat konkrit (atau nyata).

Di dalam kelas, terdapat access modifier, *Class* modifier, dan *Class* body, dibawah ini merupakan penjelasan tentang isi dari sebuah kelas.

1. Access Modifier

Access modifier merupakan keyword yang digunakan untuk menentukan spesifikasi tingkat akses suatu variable atau method (anggota kelas). Pengaksesan yang dimaksud bisa berupa pengaksesan dalam kelas yang sama, turunan, maupun di luar kelas di mana variable dan method dideklarasikan. Dalam Java, access modifier terdiri atas :

a. Default

   Access modifier ini hanya menspesifikasikan kelas-kelas (*Class*es) di paket yang sama yang bisa mengakses variable dan method suatu kelas. Anggota kelas dengan default access bisa melihat kelas lain di paket yang sama. Apabila tidak ada access modifier pada suatu pendeklarasian kelas, berarti yang dimaksud adalah default access.

b. Public

   Apabila suatu anggota kelas (*Class* member) dideklarasikan sebagai public, maka anggota tersebut akan bisa digunakan oleh siapa saja, yaitu baik oleh kelas yang sama, kelas lain, maupun lingkungan luar kelas.

c. Protected

   Access modifier protected menspesifikasikan atau menentukan anggota kelas (*Class* member) yang hanya bisa diakses oleh method-method yang ada di dalam kelas yang sama dan subkelas-subkelas turunan dari kelas tersebut.

d. Private

   Access modifier private merupakan tingkatan akses yang sangat terbatas. Pada keadaan ini tidak ada kelas lain yang dapat mengakses anggota kelas private, sekaligus termasuk subkelas-subkelasnya.

2. *Class* Modifier

   a. Public

      Apabila suatu kelas (*Class*) dideklarasikan sebagai public, maka kelas tersebut akan bisa digunakan oleh kelas lain maupun (termasuk dari lingkungan luar) tanpa memperhatikan apakah kelas lain tersebut berasal dari paket (package) yang sama atau tidak.

   b. Abstract

      Apabila suatu kelas (*Class*) dideklarasikan sebagai abstract, maka kelas tersebut tidak akan bisa diwujudkan menjadi suatu objek. Kelas ini disusun dari satu atau lebih metode abstrak, artinya method-method dalam kelas ini dideklarasikan akan tetapi tanpa badan (tidak d2mplementasikan).

   c. Final

      Apabila suatu kelas (*Class*) dideklarasikan sebagai final, maka kelas tersebut tidak akan bisa diturunkan lagi menjadi kelas turunan yang lain. Dengan kata lain sudah selesai atau berakhir (final). Tujuannya adalah untuk mencegah perluasan yang tidak diinginkan.

   d. Extends

      Class modifier extends digunakan untuk mendeklarasikan kelas turunan dari kelas yang lainnya.

   e. Implements

      Class modifier implements digunakan untuk mendeklarasikan sebuah kelas yang akan mengimplementasikan satu atau lebih *interface* Java. Apabila *interface* Java yang akan dimplementasikan lebih dari satu, maka pendeklarasiannya dipisahkan dengan tanda koma “,”.

B. Objek

Dalam kenyataannya, sebuah objek dalam OOP adalah sebuah persilangan yang berbagi-pakai (share) sejumlah ciri dari objek umum dengan fitur (feature) dari sebuah bentuk komputer[6].

Sebuah objek secara praktis pemrograman berorientasi objek bisa didefinisikan sebagai berikut :

1. Setiap objek dimiliki oleh kelas objek, sehingga sebuah objek tidak bisa hadir tanpa sebuah kelas yang mendefinisikannya. Dengan kata lain objek adalah wujud (instance) dari sebuah kelas.
2. Sebuah objek (dan kelas yang memuatnya) adalah sebuah pengkapsulan (encapsulation) yang memasukkan data dan operasi untuk pemrosesannya.
3. Atribut-atribut (attributes) objek membantu untuk menyimpan dan menjaga status objek. Atribut-atribut ini menentukan apa yang d2ngat mengenai objek. Methode objek adalah satu-satunya cara untuk mengakses data dan memodifikasi statusnya. Cara pengaksesan dan pemodifikasian data dilakukan dengan mengirimkan sebuah pesan ke objek tersebut.

C. Abstraksi

Abstraksi dapat didefinisikan sebagai suatu proses melakukan desain *Class* dan menentukan data dan method yang akan dimiliki oleh sebuah *Class*[6].

Sebuah method abstrak mendefinisikan sebuah antarmuka dalam kelas dasar dan meninggalkan implementasi pada kelas turunan. Kelas abstrak adalah sebuah kelas yang berisi satu atau beberapa method abstrak.

D. Pengkapsulan

Pengkapsulan (encapsulation) merupakan proses pembungkusan atau penyederhanaan dari beberapa data atau method menjadi sebuah objek (object) atau kelas (*Class*)[6].

E. Pewarisan Sifat

Penurunan atau pewarisan sifat (inheritance) ini merupakan cirri utama dari OOP dimana sifat-sifat yang terdapat pada kelas induk (base *Class*) akan dimiliki oleh kelas turunannya (derived *Class*)[6]. Akan tetapi hal itu tentunya bergantung juga pada access specifier (yaitu, public dan private) yang diberikan dalam proses penurunan kelas.

2.2.5. Unified Modelling Language (UML)

Pada perkembangan teknik pemrograman berorientasi objek, muncul sebuah standarisasi bahasa pemodelan untuk pembangunan perangkat lunak yang dibangun dengan menggunakan teknik pemrograman berorientasi objek, yaitu *Unified Modelling Language* (UML). Adapun pengertian dari UML adalah salah satu standar bahasa yang banyak digunakan di dunia industri untuk mendefinisikan requirement, membuat analisis dan desain, serta menggambarkan arsitektur dalam pemrograman berorientasi objek[7].

UML muncul karena adanya kebutuhan pemodelan visual untuk menspesifikasikan, menggambarkan, membangun, dan dokumentasi dari sistem perangkat lunak. Dalam hal ini UML merupakan suatu bahasa visual untuk melakukan pemodelan dan komunikasi mengenai sebuah sistem dengan menggunakan diagram dan teks-teks pendukung.

2.2.6. Sejarah UML

Bahasa pemrograman berorientasi objek yang pertama dikembangkan dikenal dengan nama Simula-67 yang dikembangkan pada tahun 1967. Bahasa pemrograman ini kurang berkembang dan dikembangkan lebih lanjut, namun dengan kemunculannya telah memberikan sumbangan yang besar pada developer pengembang bahasa pemrograman berorientasi objek selanjutnya.

Perkembangan aktif dari pemrograman berorientasi objek mulai menggeliat ketika berkembangnya bahasa pemrograman Smalltalk pada awal 1980-an yang kemudian dikuti dengan perkembangan bahasa pemrograman berorientasi objek yang lainnya seperti C objek, C++, Eiffel, dan CLOS. Secara actual, penggunaan bahasa pemrograman berorientasi objek pada saat itu masih terbatas, namun telah banyak menarik perhatian di saat itu. Sekitar lima tahun setelah Smalltalk berkembang, maka berkembang puka metode pengembangan berorientasi objek. Metode yang pertama diperkenalkan oleh Sally Shalaer dan Edward Yourdon, diikuti oleh Grady Booch, James R. Rumbaugh, Michael R. Blaha, William Lorensen, Frederick Eddy, William Premerlani, dan masih banyak lagi.

Karena banyaknya metodologi-metodologi yang berkembang pesat saat itu, maka muncullah ide untuk membuat sebuah bahasa yang dapat dimengerti semua orang. Usaha penyatuan ini banyak mengambil dari metodologi-metodologi yang berkembang pada saat itu. Maka dibuat bahasa yang merupakan gabungan dari beberapa konsep seperti konsep Object Modelling Technique (OMT) dari Rumbaugh dan Booch, konsep The *Class*es, Responsibilities, Collaborators (CRC) dari Rebecca Wirfs-Brock, konsep pemikiran Ivar Jacobson, dan beberapa konsep lainnya dimana James R. Rumbaugh, Grady Booch, dan Ivan Jacobson bergabung dalam sebuah perusahaan yang bernama Rational *Software* Corporation menghasilkan bahasa yang disebut dengan Unified Modeling Language (UML).

Pada 1996, Object Management Group (OMG) mengajukan proposal agar adanya standarisasi pemodelan berorientasi objek dan pada bulan September 1997 UML diakomodasi oleh OMG sehingga sampai saat ini UML telah memberikan kontribusinya yang cukup besar di dalam metodologi berorientasi objek dan hal-hal hanya terkait di dalamnya.

2.2.7. Diagram UML

UML menggunakan berbagai macam diagram dengan fungsi masing-masing untuk menggambarkan setiap proses dari sistem berorientasi objek. Berikut merupakan beberapa diagram UML diantaranya[7] :

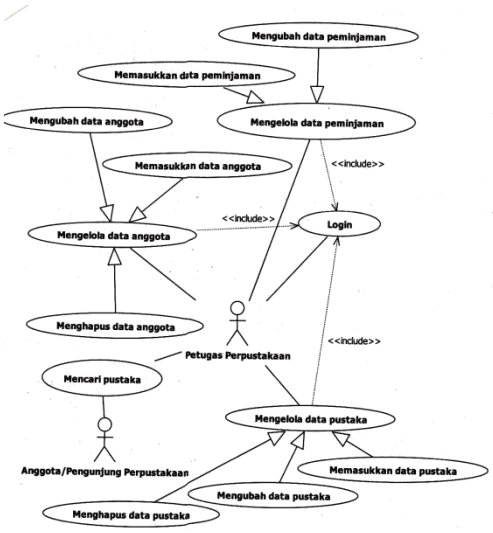
1. *USE CASE DIAGRAM*

Use Case atau diagram *Use Case* merupakan pemodelan yang digunakan untuk menggambarkan kelakuan (behavior) dari sistem yang akan dibuat[7]. *Use Case* mendeskripsikan sebuah interaksi antara satu atau lebih aktor dengan sistem yang akan dibuat. Secara kasar, *Use Case* digunakan untuk mengetahui fungsi apa saja yang ada di dalam sebuah sistem dan siapa saja yang berhak menggunakan fungsi-fungsi tersebut.

Syarat penamaan pada *Use Case* adalah nama didefinisikan sesimpel mungkin dan dapat dipahami. Ada dua hal utama pada *Use Case* yaitu pendefinisian apa yang disebut aktor dan *Use Case*[7].

1. Aktor merupakan orang, proses, atau sistem lain yang berinteraksi dengan sistem yang akan dibuat diluar sistem yang akan dibuat itu sendiri, jadi walaupun simbol dari aktor adalah gambar orang, tapi aktor belum tentu merupakan orang.
2. *Use Case* merupakan fungsionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesar antarunit atau aktor.

Contoh dari *USE CASE DIAGRAM* dapat dilihat pada gambar 2.20.



Gambar 2.20.Contoh dari USE CASE DIAGRAM[7]

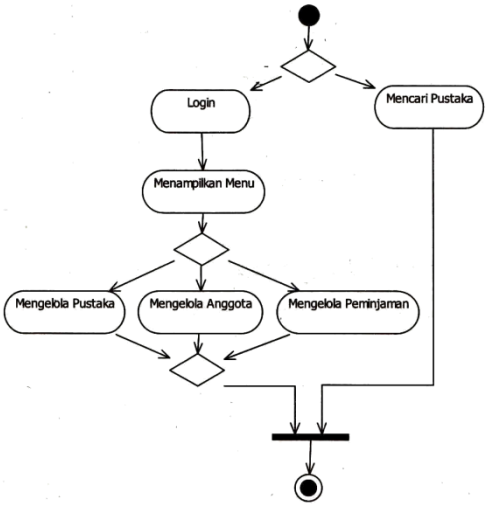
B. *ACTIVITY DIAGRAM*

Diagram aktivitas atau *ACTIVITY DIAGRAM* adalah sebuah diagram yang menggambarkan workflow (aliran kerja) atau aktivitas dari sebuah sistem atau proses bisnis[6]. Dalam diagram aktivitas yang perlu diperhatikan adalah bahwa diagram aktivitas menggambarkan aktivitas sistem, bukan apa yang dilakukan aktor, jadi aktivitas yang dapat dilakukan oleh sistem.

Diagram aktivitas juga banyak digunakan untuk mendefinisikan hal-hal berikut[7] :

1. Rancangan proses bisnis di mana setiap urutan aktivitas yang digambarkan merupakan proses bisnis sistem yang didefinisikan.
2. Urutan aau pengelompokan tampilan dari sistem/user *interface* di mana setiap aktivitas dianggap memiliki sebuah rancangan antarmuka tampilan.
3. Rancangan pengunjian di mana setiap aktivitas dianggap memerlukan sebuah pengujian yang perlu didefinisikan kasus ujinya.

Contoh dari *ACTIVITY DIAGRAM* dapat dilihat pada gambar 2.21.



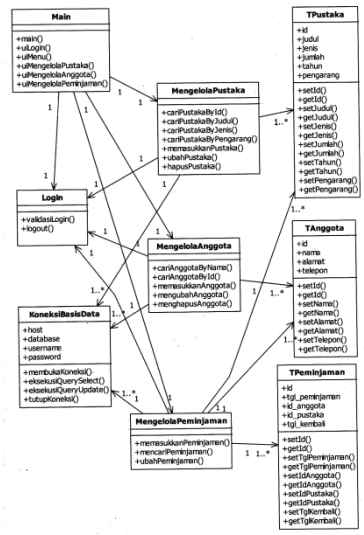
Gambar 2.21.Contoh dari ACTIVITY DIAGRAM[7]

C. *CLASS DIAGRAM*

Diagram kelas atau *CLASS DIAGRAM* menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang akan dibuat untuk membangun sistem. Kelas memiliki apa yang disebut atribut dan metode atau operasi[7].

1. Atribut merupakan variabel-variabel yang dimiliki oleh suatu kelas
2. Operasi atau metode adalah fungsi-fungsi yang dimiliki oleh suatu kelas

Contoh dari *CLASS DIAGRAM* dapat dilihat pada gambar 2.22.

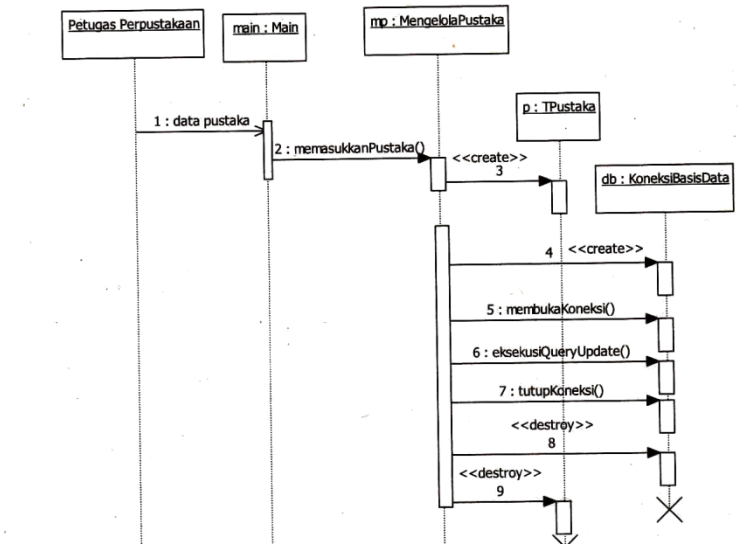


Gambar 2.22.Contoh dari *CLASS DIAGRAM*[7]

D. *SEQUENCE DIAGRAM*

Diagram sekuen adalah diagram yang menggambarkan kelakuan objek pada *Use Case* dengan mendeskripsikan waktu hidup objek dan message yang dikirimkan dan diterima antarobjek[7]. Oleh karena itu untuk menggambarkan diagram sekuen maka harus diketahui objek-objek yang terlibat dalam sebuah *Use Case* beserta metode-metode yang dimiliki kelas yang d2nstansiasi menjadi objek itu.

Banyaknya diagram sekuen yang harus digambarkan adalah sebanyak pendefinisian *Use Case* yang memiliki prose situ sendiri atau yang penting semua *Use Case* yang telah didefinisikan interaksi jalannya pesan sudah dicakup pada diagram sekuen sehingga semakin banyak *Use Case* yang didefinisikan maka diagram sekuen yang harus dibuat juga semakin banyak.
Contoh dari *SEQUENCE DIAGRAM* dapat dilihat pada gambar 2.23.



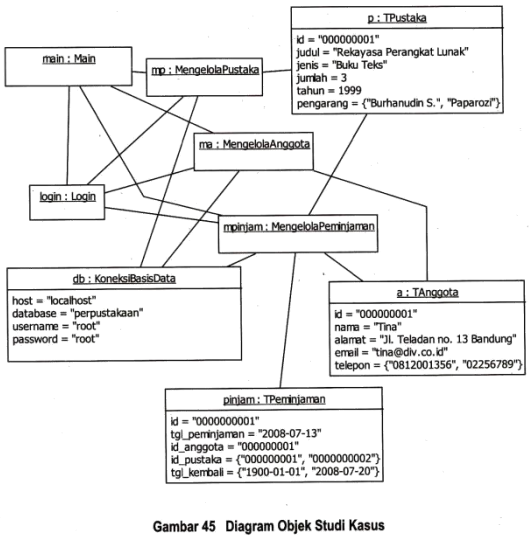
Gambar 2.23.Contoh dari SEQUENCE DIAGRAM[7]

E. *OBJECT DIAGRAM*

Diagram objek menggambarkan struktur system dari segi penamaan objek dan jalannya objek dalam sistem[7]. Pada diagram objek harus dipastikan semua kelas yang sudah didefinisikan pada diagram kelas harus dipakai objeknya, karena jika tidak, pendefinisian kelas itu tidak dapat dipertanggungjawabkan.

Untuk apa mendefinisikan sebuah kelas sedangkan pada jalannya sistem, objeknya tidak pernah dipakai. Hubungan link pada diagram objek merupakan hubungan memakai dan dipakai di mana dua buah objek akan dihubungkan oleh link jika ada objek yang dipakai oleh objek lainnya.

Contoh dari *OBJECT DIAGRAM* dapat dilihat pada gambar 2.24.



Gambar 45 Diagram Objek Studi Kasus

Gambar 2.24.Contoh dari *OBJECT DIAGRAM*[7]

F. *COMPONENT DIAGRAM*

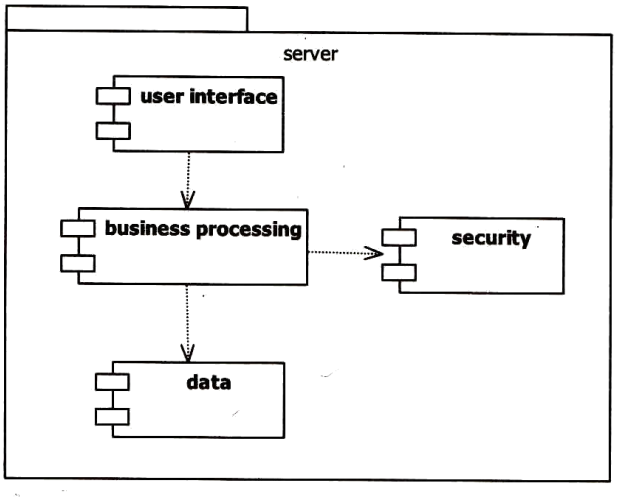
Diagram komponen dibuat untuk menunjukkan organisasi dan kebergantungan di antara kumpulan komponen dalam sebuah sistem. Diagram komponen focus pada komponen sistem yang dibutuhkan dan ada di dalam sistem[7]. Diagram komponen juga dapat digunakan untuk memodelkan hal-hal berikut[7] :

1. Source code program perangkat lunak
2. Komponen executable yang dilepas ke user
3. Basis data secara fisik
4. Sistem yang harus beradaptasi dengan sistem lain
5. Framework sistem

Adapun komponen-komponen dasar yang biasanya ada dalam suatu sistem adalah sebagai berikut[7] :

1. Komponen user *interface* yang menangani tampilan
2. Komponen business processing yang menangani fungsi-fungsi proses bisnis
3. Komponen data yang menangani manipulasi data
4. Komponen security yang menangani keamanan sistem

Contoh dari *COMPONENT DIAGRAM* dapat dilihat pada gambar 2.25.



Gambar 2.25.Contoh dari COMPONENT DIAGRAM[7]

G. Composite Structure Diagram

Diagram ini dapat digunakan untuk menggambarkan struktur dari bagian-bagian yang saling terhubung maupun mendeskripsikan struktur pada saat berjalan (runtime) dari instance yang saling terhubung[7]. Contoh penggunaan diagram ini misalnya untuk menggambarkan deskripsi dari setiap bagian mesin yang saling terkait router pada jaringan komputer, dll.

H. Package Diagram

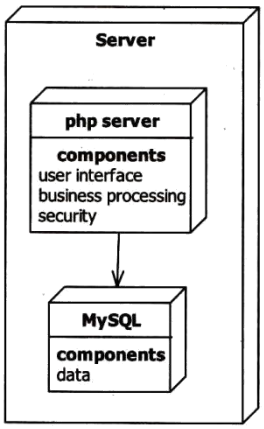
Package diagram menyediakan cara mengumpulkan elemen-elemen yang saling terkait dalam diagram UML[7].

I. *DEPLOYMENT DIAGRAM*

Diagram deployment atau *DEPLOYMENT DIAGRAM* menunjukkan konfigurasi komponen dalam proses eksekusi aplikasi[7]. Diagram deployment juga dapat digunkan untuk memodelkan hal-hal berikut :

1. Sistem tambahan (embedded system) yang menggambarkan rancangan device, node, dan selanjutnya
2. Sistem client/server
3. Sistem terdistribusi murni
4. Rekayasa ulang aplikasi

Contoh dari *DEPLOYMENT DIAGRAM* dapat dilihat pada gambar 2.26.



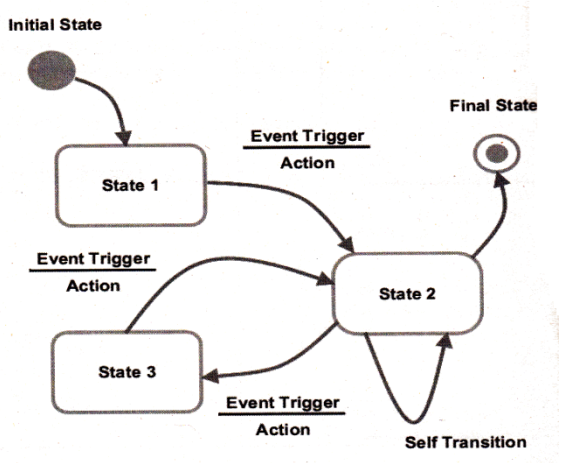
Gambar 2.26.Contoh dari DEPLOYMENT DIAGRAM[7]

J. *STATE MACHINE DIAGRAM*

Diagram mesin status digunakan untuk menggambarkan perubahan status atau transisi status dari sebuah mesin atau sistem[7]. Perubahan tersebut digambarkan dalam suatu graf berarah. *STATE MACHINE DIAGRAM* merupakan pengembangan dari diagram Finite State Automata dengan penambahan beberapa fitur dan konsep baru.

Diagram ini cocok digunakan untuk menggambarkan alur interaksi pengguna dengan sistem[7].

Contoh dari *STATE MACHINE DIAGRAM* dapat dilihat pada gambar 2.27.



Gambar 2.27.Contoh dari STATE MACHINE DIAGRAM[7]

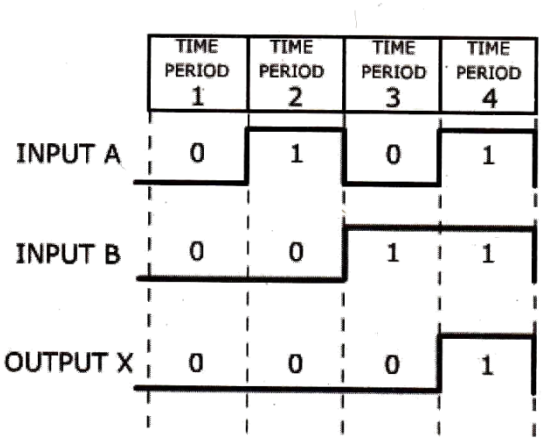
K. Communication Diagram

Diagram komunikasi merupakan penyederhanaan dari diagram kolaborasi (collaboration diagram)[7]. Diagram ini menggambarkan interaksi antarobjek/bagian dalam bentuk urutan pengiriman pesan. Diagram komunikasi merepresentasikan informasi yang diperoleh dari Diagram Kelas, Diagram Sekuen, dan Diagram *Use Case* untuk mendeskripsikan gabungan antara struktur statis dan tingkah laku dinamis dari suatu sistem.

Diagram komunikasi mengelompokkan message pada kumpulan diagram sekuen menjadi sebuah diagram. Dalam diagram komunikasi yang dituliskan adalah operasi/metode yang dijalankan antara objek yang satu dan objek yang lainnya secara keseluruhan, oleh karena itu dapat diambil dari jalannya interaksi pada semua diagram sekuen. Penomoran metode dapat dilakukan berdasarkan urutan dijalankannya metode/operasi di antara objek yang satu dengan objek yang lainnya atau objek itu sendiri.

L. *TIMING DIAGRAM*

TIMING DIAGRAM merupakan diagram yang focus pada penggambaran terkait batasan waktu. *TIMING DIAGRAM* digunakan untuk menggambarkan tingkah laku sistem dalam periode waktu tertentu[7]. *TIMING DIAGRAM* biasanya digunakan untuk mendeskripsikan operasi dari alat digital karena penggambaran secara visual akan lebih mudah dipahami daripada dengan kata-kata. Aliran waktu pada *TIMING DIAGRAM* dibaca dari kiri ke kanan.
Contoh dari *TIMING DIAGRAM* dapat dilihat pada gambar 2.28.



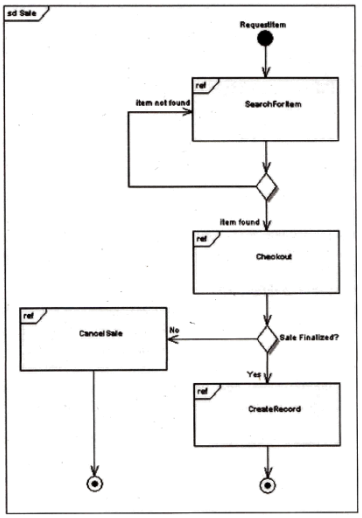
Gambar 2.28.Contoh dari *TIMING DIAGRAM*[7]

M. *ITERACTION OVERVIEW DIAGRAM*

ITERACTION OVERVIEW DIAGRAM mirip dengan diagram aktivitas yang berfungsi untuk menggambarkan sekumpulan urutan aktivitas. Iterr*ACTION OVER*view diagram adalah bentuk aktivitas diagram yang setiap titik merepresentasikan diagram interaksi[7]. Interaksi diagram dapat meliputi diagram sekuen, diagram komunikasi, inter*ACTION OVER*view diagram, dan *TIMING DIAGRAM*.

Hampir semua notasi pada inter*ACTION OVER*view diagram sama dengan notasi pada diagram aktivitas. Sebagai contoh initial, final, decision, merge, fork, dan join nodes sama seperti pada diagram aktivitas. Tambahan pada inter*ACTION OVER*view diagram adalah inter*ACTION* accurrence dan inter*ACTION* element.

Contoh dari *OVER*view diagram dapat dilihat pada gambar 2.29.



Gambar 2.29.Contoh dari ITERACTION OVERVIEW DIAGRAM[7]

2.2.8. Teknik Pengujian Perangkat Lunak

Pengujian perangkat lunak adalah elemen kritis dari jaminan kualitas perangkat lunak dan merepresentasikan kajian pokok dari spesifikasi, desain, dan pengkodean[5].

2.2.9. Pengujian *Black box*

Menurut Roger S. Pressman[5], pengujian *black box* berfokus pada persyaratan fungsional perangkat lunak. Dengan demikian, pengujian *black box* menungkinkan perekayasa perangkat lunak mendapatkan serangkaian kondisi input yang sepenuhnya menggunakan semua persyaratan fungsional untuk suatu program. Pengujian *black box* bukan merupakan alternatif dari teknik *White Box*, tetapi merupakan pendekatan komplementer yang kemungkinan besar mampu mengungkap kelas kesalahan daripada metode *White Box*.

Pengujian *black box* berusaha menemukan kesalahan dalam kategori sebagai berikut :

A. Fungsi-fungsi yang tidak benar atau hilang
B. Kesalahan dalam *interface*

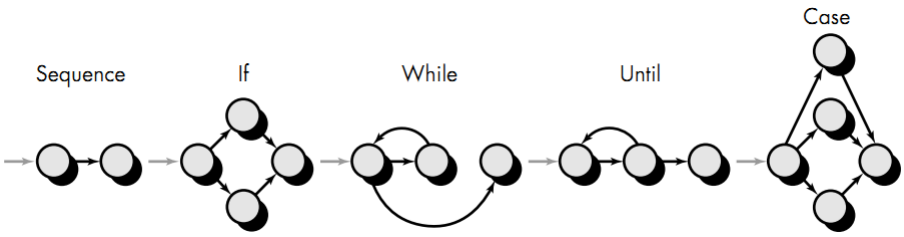
C. Kesalahan dalam struktur data atau akses database eksternal
D. Kesalahan kinerja
E. Inisialisasi dan kesalahan terminasi

2.2.10. Pengujian *White Box*

Menurut Roger S. Pressman[5], pengujian *White Box*, yang kadang-kadang disebut pengujian glass box, adalah metode desain test case yang menggunakan struktur kontrol desain procedural untuk memperoleh test case. Dengan menggunakan metode pengujian *White Box*, perekayasa sistem dapat melakukan test case sebagai berikut :

A. Memberikan jaminan bahwa semua jalur independen pada suatu modul telah digunakan paling tidak satu kali
B. Menggunakan semua keputusan logis pada sisi true dan false
C. Mengeksekusi semua loop pada batasan mereka dan pada batas operasional mereka
D. Menggunakan struktur data internal untuk menjamin validitasnya.

Pengujian *White Box* yang berupa notasi diagram alir dapat dilihat pada gambar 2.30.



Gambar 2.30.Notasi Diagram Alir[5]

2.2.11. Tools yang Digunakan

Dalam pembuatan sebuah aplikasi tentu dibutuhkan satu atau beberapa tools yang dalam hal ini berupa *Software* (perangkat lunak). Berikut merupakan beberapa perangkat lunak yang digunakan untuk membangun sebuah aplikasi permainan *The Legend Of Cursed Amulet*.

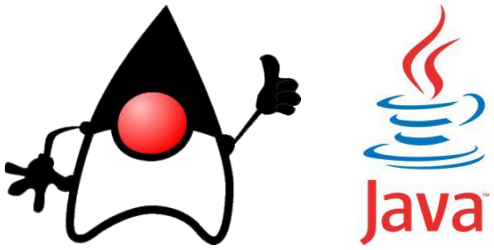
2.2.12. Java

Java adalah salah satu bahasa pemrograman komputer yang berorientasi objek[11], yang diciptakan oleh satu tim dari perusahaan Sun Microsystem, perusahaan workstation UNIX (Sparc) yang cukup terkenal. Java diciptakan berdasarkan bahasa C++, dengan tujuan platform independent (dapat dijalankan pada berbagai jenis *Hardware* tanpa kompilasi ulang), dengan slogan "Write Once, Run Anywhere (WORA), dan Java pada hakikatnya lebih sederhana dan memakai objek secara murni dibanding dengan bahasa pemrograman C++.

Asal usul java dimulai pada tahun 1991 ketika Sun Microsystem melakukan penelitian terhadap berbagai produk elektronika. James Gosling (kepala proyek penelitian) beserta Patrick Naughton ditugaskan untuk merancang perangkat lunak aplikasi yang independen, tidak bergantung pada jenis perangkat keras, agar bisa dipakai pada berbagai peralatan elektronik itu. James Gosling kemudian memakai bahasa pemrograman C++ untuk menulis beberapa aplikasi untuk peralatan mikro, namun ternyata dia mengalami banyak kesulitan. Setiap kali sebuah peralatan mikro menggunakan jenis microchip yang berbeda, program yang digunakannya harus dikompilasi ulang. Berdasarkan hal itu kemudian munculah ide dari James Gosling untuk menciptakan sebuah bahasa baru, yang ditulis berdasarkan C++ namun dengan beberapa perbaikan. Bahasa ini kemudian disebut Oak (nama pohon yang terlihat dari jendela ruangan kerja James Gosling), yang digunakan untuk membuat perangkat lunak cerdas bagi peralatan elektronika buatan Sun Mirosystem.

Pada tahun 1994, ketika Web mulai popiler nama bahasa Oak kemudian diganti menjadi Java (salah satu sebabnya ternyata nama Oak telah terlebih dahulu dipakai untuk hal lain). Menurut gosip, Gosling bingung mencari nama untuk bahasa baru tersebut karena setiap nama yang dipikirkannya untuk nama bahasa pemrogramannya ternyata sudah ada yang memakai, sampai suatu ketika dia mampir di sebuah kafe untuk minum kopi, dan ia meminta "Java". Timbullah idenya untuk memakai nama Java. Menurut Gosling nama ini cukup cool.

Java kemudian dipakai untuk membuat sebuah browser yang bernama WebRunner. Setelah berhasil maka browser ini kemudian dinamakan HotJava dengan simbol secangkir kopi panas, seperti pada gambar 2.31.



Gambar 2.31.Maskot Java (Duke) dan HotJava[11]

Menurut Definisi yang diberikan oleh Sun Microsystem, Java adalah bahasa pemrograman simple, object oriented, distributed, robust, secure, architecture neutral, portable, high-performance, multithreaded, dan dynamic.

A. Java bahasa yang simple : Rancangan bahasa Java dibuat sedemikian rupa sehingga dengan cepat dapat dikenali dan dipahami oleh pemrogram pemula sekalipun karena mirip dengan bahasa C / C++ dan kedua lebih sederhana dari C/C++ dalam beberapa hal mengingat berbagai hal yang sulit pada C/C++ sudah tidak ada pada Java, misalnya pemakaian pointer.
B. Java adalah bahsa yang object-oriented : Konsep objek diterapkan pada pembuatan program Java, bahkan Java menggunakan konsep objek secara murnni sehingga tidak mungkin seseorang membuat program Java tanpa mendefinisikan kelas dan menggunakan objek.
C. Java adalah bahasa yang distributed : Bahasa Java dirancang sedemikian rupa sehingga mendukung pembuatan aplikasi dalam jaringan komputer. Java menyediakan beberapa kelas objek untuk mendukung aplikasi jaringan yang dihimpun dalam paket java.net.
D. Java adalah bahasa yang robust : Rancangan Java dibuat sedemikian rupa hingga mengurangi kemungkinan menjadi beku (freeze) ketika dijalankan. Kemungkinan error pun diminimalkan, misalnya dengan menerapkan strong-type sehingga program bisa terhindar dari kesalahan karena tipe data yang keliru, Java juga menerapkan model memori yang dapat mencegah memori corrupt dan *OVER*writed, misalnya dengan membuang pemakaian pointer.

E. Java adalah bahasa yang secure : Secara otomatis Java menerapkan pengamanan terhadap aplikasi sehingga mengurangi kemungkinan terjadinya serangan dari pengguna jaringan.

F. Java adalah bahasa yang architecture neutral : Program Java yang telah dikompilasi dapat dijalankan pada berbagai mesin dan tidak hanya bergantung pada satu arsitektur komputer saja.

G. Java adalah bahasa yang portable : Program Java sangat mudah dipindahkan dari satu mesin ke mesin lain tanpa harus dikompilasi ulang.

H. Java adalah bahasa yang high-performance : Pada hakikatnya program C yang telah dikompilasi berjalan jauh lebih cepat dari Java, karena hasil kompilasi C adalah bahasa mesin dari komputer yang dipakai untuk mengompilasi, namun hasil kompilasi ini belum tentu bisa jalan pada komputer yang berbeda arsitektur. Program Java dikompilasi menjadi bytecode yang bisa berjalan di berbagai mesin, namun ketika dijalankan bytecode ini harus diterjemahkan oleh JVM ke dalam bahasa mesin yang sesuai, sehingga lebih lambat dari C, namun karena Java dapat digunakan pada jaringan komputer yang kenyataannya tidak terlalu cepat karena hambatan saluran komunikasi maka program Java masih terasa cukup cepat.

I. Java adalah bahasa yang multithreaded : Java dirancang untuk menangani berbagai aplikasi yang berjalan secara bersamaan, misalnya memainkan lagu, sambil melakukan download, dan pengguna sedang melakukan scrolling pada window yang diamatinya.

J. Java adalah bahasa yang dynamic : Bahasa Java dirancang untuk beradaptasi dengan lingkungan yang dapat berubah dengan cepat. Java dapat memanggil berbagai kelas objek pada saat dibutuhkan, walaupun harus melalui jaringan. Di samping itu Java memiliki tepresentasi run-time sehingga objek yang sedang digunakan oleh suatu aplikasi dapat diperiksa berasal dari kelas yang mana melalui informasi run-time.

2.2.13. Greenfoot

Greenfoot adalah pengembangan interaktif di lingkungan Java yang dirancang terutama untuk tujuan pendidikan di sekolah tinggi dan tingkat sarjana. Hal ini memungkinkan untuk melakukan pengembangan aplikasi grafis dua dimensi dengan mudah, seperti simulasi dan permainan interaktif[18].

Greenfoot mengajarkan Java dengan berbasis orientasi objek. Membuat sebuah 'actors' yang hidup di 'worlds' untuk membangun permainan, simulasi, dan program grafis lainnya. Greenfoot adalah visual dan interaktif. Visualisasi dan alat interaksi yang dibangun terhadap lingkungan. Para aktor yang diprogram dalam kode Java standar tekstual, memberikan kombinasi pengalaman pemrograman dalam bahasa berbasis teks tradisional dengan eksekusi visual.

Interface pada greenfoot adalah IDE lengkap yang meliputi manajemen proyek, auto-completion, syntax highlighting, dan alat-alat lainnya yang umum untuk kebanyakan IDE. Greenfoot digunakan oleh ribuan institusi di seluruh dunia. Serta menyediakan transisi mudah ke lingkungan lain, seperti BlueJ dan IDE lebih profesiona[l8].