BAB 2

LANDASAN TEORI

2.1 Permainan Video *(Video Game)*

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

Untuk melakukan perintah memanipulasi permainan *video* digunakan *game controller*, adapun *game controller* dapat berupa *joystick* yang terdiri dari satu tombol saja. Namun seiring perkembangan jaman, saat ini *game controller* tidak hanya *joystick*, melainkan dengan menggunakan sebuah *mouse* maupun *keyboard* dan juga penggunaan *joystick* yang memiliki beberapa tombol.

2.1.1 Sejarah Singkat *Video Game*

Video game pertama diciptakan oleh dua orang utama, yaitu William Higinbotham dan Steve Russell[6]. Saat itu William Higinbotham berperan sebagai orang pertama yang mendesain dan mengimplementasikan sebuah *video game*, dan Steve Russell berperan sebagai orang pertama yang menciptakan sebuah *game* yang kemudian menginspirasikan bermilyaran dollar industri *video game*.

Berdasarkan sejarah, yang pertama kali menemukan *video game* ialah *United States Department of Enery*. Khususnya ialah seorang pria bernama William Higinbotham yang menjadi kepala pada bagian *Instrumentation division for Brookhaven National Laboratory*. Beforen Brookhaven, William pernah bekerja sebelumnya di *Manhattan Project* dan telah menyaksikan ledakan atom pertama. Namun, pada tahun 1950an, masyarakat khawatir akan kekuatan atom, dan Brookhaven mencoba untuk menampilkannya untuk memasyarakatkan penelitian yang dilakukan pada acaran kunjungan tahunan. Ratusan orang berkunjung ke laboratorium setiap musim gugur untuk melihat berbagai penelitian yang sedang dilakukan disana. Pada tahun 1958, William mendapatkan ilham bagaimana membuat pengunjung agar tidak bosan pada setiap kunjungan tahunan ke laboratoriumnya, yaitu dengan cara membuat tampilan interaktif. Tampilan ini menjadi *video tennis game.*

Dalam waktu tiga minggu, permainan *video* pertama diciptakan. William menggambarkan desain asli untuk permain tersebut hanya dalam waktu beberapa jam saja, bekerja sama dengan Robert V. Dvorak, seorang spesialis teknisi, yang memasangkan *patchboard*. Mereka berdua menghabiskan waktu selama dua hari untuk *debugging* dan *running* permainan tersebut, akhirnya terciptalah sebuah permainan *video* yang diberi nama *Tennis for Two* dan menjadi suguhan paling menarik bagi pengunjung.

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, komputer menjadi barang yang sangat langka, komputer hanya dapat ditemukan di sekolah paling bergengsi pada saat itu, seperti MIT. Steve Russell merupakan murid di MIT, dan dalam masa belajarnya selama enam tahun, dan menghabiskan 200 jam, dia menciptakan permainan *video two-player* yang dinamakan *Spacewar* pada sebuah komputer DEC PDP-1. Tujuan dari permainan tersebut adalah setiap pemain melakukan *maneuver* pesawatnya ketika mencoba menembak pesawat pemain lain dengan menggunakan misil. Menggunakan empat tombol yang terpisah, setiap pemain dapat melakukan putara searah jarum jam maupun berlawanan arah jarum jam, dorongan, atau menembak dengan misil.

Spacewar diciptakan pada tahun 1961, namun pada musim semi tahun 1962, permainan tersebut telah mengalami perluasan. Pete Sampson menambahkan sebuah area bintang yang akurat ke dalam layar dengan mengintegrasikan program yang telah ada, yang dinamakan *Expensive Planetarium*. Kemudian, Dan Edwards mengoptimalkan permainan tersebut dengan menambahkan perhitungan gravitasi agar lebih meningkat lagi performanya. Kemudian, ditambahkan juga kerlip matahari ke dalam pusat tampilan yang berpengaruh kepada pesawat dan menghancurkan setiap benda yang berada di dekatnya. Terakhir, J. Martin Graetz menambahkan sebuah konsep *hyperspace*, yaitu sebuah kemampuan untuk seorang pemain yang dapat membuat pesawatnya dapat berbelok dari posisi semula ke posisi lain secara acak sehingga

membuat pemain merasa panik. Dengan keadaan ini, taktik yang menarik mulai diciptakan, seperti menembak pesawat sendiri ketika dekat dengan matahari untuk mempercepat menyusul lawan yang bergerak lambat.

Pada awal tahun 1970an, dua orang utama, Ralph Baer dan Nolan Bushnell, membawa permainan *video* ke dalam rumah agar dapat dinikmati semua orang. Kemudian, dua orang inilah yang memberikan kelahiran industri permainan *video* seperti yang kita ketahui saat ini.

Pada akhir tahun 1971, perusahaan televisi yang bernama *Magnavox* akhirnya menandatangani kontrak dengan Sanders. Pada tahun 1972, *Magnavox* menunjukkan peralatan terbarunya yang diberi nama *Magnavox Odyssey*.

Pada tahun 1983, sebuah perombakan 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[6].

A. ***Platformer***

Permainan *platform* asli terdiri dari karakter yang berlari dan melompat pada arena bermain dengan mode *side-scrolling*[6]. Beberapa contoh dari *platformers* yang terkenal diantaranya *Super Mario Bros, Sonic the Hedgehog, Pitfall!,* dan *Super Mario 64,* seperti pada gambar 2.1.



Gambar 2.1 Contoh permainan *platformers*[6]

B. **Fighting**

Permainan *fighting,* pemain bertarung dengan pemain lainnya atau dengan komputer baik menggunakan ilmu beladiri atau permainan pedang. Permainan ini berasal dari permainan *arcades,* dimana pemain dapat menandakan keinginan mereka untuk menantang yang lainnya dengan menempatkan tempat diatasnya[6]. Contoh dari permainan *video* yang berjenis *fighting* yang plaing terkenal diataranya *Double Dragon, Street Fighter,* dan *Mortal Kombat,* seperti pada gambar 2.2.



Gambar 2.2 Contoh permainan *fighting*[6]

C. ***First-Person Shooter***

First-Person Shooter merupakan sebuah *action game* yang menempatkan pemain "*behind the eyes"* pada karakter permainan[6]. Pada 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*, seperti pada gambar 2.3.



Gambar 2.3 Contoh dari permainan *First-Person Shooter*[6]

D. ***Real-Time Strategy* (RTS)**

Permainan *real-time strategy* mempunyai tujuan yaitu pemain mengoleksi berbagai sumber daya, membangun pasukan, dan mengontrol pasukannya untuk menyerang musuh. Aksi terhadap permainan ini adalah *fast-paced* dank arena merupakan permainan yang berlanjut, maka keputusan untuk strategi harus dibuat secara cepat[6]. Beberapa contoh dari permainan ini diantaranya

The Ancient Art of War, Command and Conquer, dan *Warcraft*, seperti pada gambar 2.4.



Gambar 2.4 Contoh permainan *Real-Time Strategy* (RTS)[6]

E. ***Role-Playing Game* (RPG)**

Permainan *role-playing game* mengandalkan strategi dalam setiap langkahnya dan di dalamnya terdapat cerita serta tujuan yang harus dilakukan oleh para pemainnya[6]. Salah satu contoh dari permainan ini adalah *Final Fantasy series*, seperti pada gambar 2.5.



Gambar 2.5 Contoh dari *Role-Playing Game*[6]

F. ***Survival Horror***

Permainan *Survival Horror* merupakan *subgenre* dari permainan *action-adventure* dan *first-person shotter*[6]. Permainan ini memungkinkan para pemain menjelajahi setiap bangunan atau kota dimana terdapat berbagai

macam monster dan *zombie*s. Salah satu contoh game yang paling terkenal dalam permainan jenis ini adalah *Resident Evil*, seperti pada gambar 2.6.



Gambar 2.6 Contoh dari Permainan *Survival Horror*[6]

G. ***Simulation***

Permainan simulasi diadaptasi dari sistem simulasi. Dalam permainan ini para pemain berada dalam sebuah scenario dimana para pemain mengontrol pergerakan secara besar-besaran atau sebagai satu karakter saja[6]. Salah satu contoh dari permainan ini adalah *The Sims*, seperti pada gambar 2.7.



Gambar 2.7 Contoh permainan *Simulation*[6]

H. ***Racing***

Permainan *racing* melibatkan kompetisi balap kendaraan mulai dari balap motor, mobil, maupun *go-karts*[6]. Jenis permainan ini sedikit berbeda dibanding permainan lainnya. Salah satu contoh dari permainan ini *Need for Speed*, seperti pada gambar 2.8.



Gambar 2.8 Contoh permainan *Racing*[6]

I. ***Sports***

Permainan ini merupakan jenis yang mensimulasi olahraga yang ada pada dunia nyata[6]. Salah satu contohnya adalah *Tiger Woods Golf*, seperti pada gambar 2.9.



Gambar 2.9 Contoh permainan *Sport*[6]

J. ***Puzzle***

Permainan ini merupakan penggabungan dari berbagai elemen, diantaranya pencocokan, logika, strategi, dan keberuntungan[6]. Salah satu contoh dari jenis permainan ini adalah *Tetris,* seperti pada gambar 2.10.

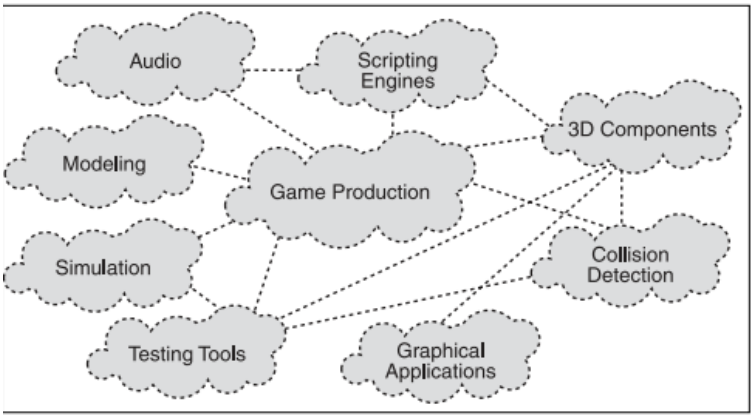


Gambar 2.10 Contoh dari permainan *Puzzle*[6]

2.1.3 Konsep Permainan Video

Menurut J. P. Flynt dan O. Salem dalam bukunya[18], 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 2.11.



Gambar 2.11 Spesialisasi peningkatan karakteristik industri *game*[18]

2.2 *Gameplay Tower Defense*

Tower Defense merupakan sebuah *sub-genre* dari permainan komputer *real-time strategy*[17]. Dalam permainan *tower defense* setiap pemain menggunakan media *mouse* untuk melakukan interaksi terhadap permainan tersebut.

Gameplay dari permainan dengan jenis *tower defense* adalah mencegah setiap musuh yang muncul untuk menuju *goal point*, dimana apabila musuh mencapai *goal point* tersebut maka darah atau nyawa dari pemain akan berkurang dan apabila habis maka pemain dinyatakan kalah. Setiap pemain dapat menempatkan *tower* untuk melakukan pertahanan dan secara otomatis akan menembak musuh yang berada di dalam jangkauannya, dalam beberapa permainan dengan jenis *tower defense* terdapat berbagai macam *tower* dengan harga yang beragam. Adapun uang yang digunakan untuk membeli *tower* didapatkan dari hasil membunuh setiap musuh yang ada. Permaianan dinyatakan

sukses apabila pemain dapat mempertahankan darah atau nyawa nya sampai waktu yang ditentukan habis atau berdasarkan jumlah musuh yang dibunuh.

2.2.1 Karakteristik *Tower Defense*

Karakteristik yang terdapat di dalam aplikasi permainan yang berjenis *tower defense* adalah sebagai berikut[17] :

A. Terdiri dari dua *point*, yaitu *point* awal tempat keluarnya musuh dan *point* akhir yang merupakan tujuan akhir musuh
B. Adanya *shop* yang digunakan untuk membeli *tower*, sehingga *tower* yang bisa digunakan bervariasi
C. Metode interaksi antara pemain dan aplikasi permainan dengan menggunakan *mouse*
D. Terdapat nyawa dan jumlah *coin* pemain
E. Terdapat jalur yang telah ditentukan sehingga musuh berjalan melewati jalur tersebut

2.3 Kecerdasan Buatan *(Artificial Intelligence)*

2.3.1 Definisi AI

Sebagian kalangan menerjemahkan *Artificial Intelligence* sebagai kecerdasan buatan, kecerdasan artifisial, intelijensia artifisial, atau intelejensia buatan. Pada bukunya[15], 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[15], yaitu :

A. ***Thinking humanly*** : *the cognitive modeling approach*

Pendekatan ini dilakukan dengan dua cara sebagai berikut :

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 :

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.

2.3.2 Teknik Pemecahan Masalah AI

Terdapat empat teknik dasar pemecahan masalah yang terdapat pada bidang *academic/traditional artificial intelligence* (AI)[15], 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[15] :

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.

2.3.2.1 Algoritma A* (*A Star Algorithm*)

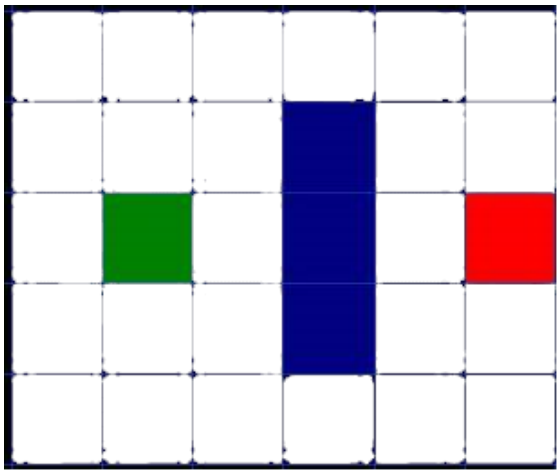
Algoritma A* (A *star*) merupakan algoritma *Best First Search* yang menggabungkan *Uniform Cost Search* dan *Greedy Best-First Search*[15]. Biaya yang diperhitungkan didapat dari biaya sebenarnya ditambah dengan biaya perkiraan, dalam notasi matematika dituliskan sebagai berikut :

$$f(n) = g(n) + h(n).....[15]$$

Dengan perhitungan biaya seperti diatas, algoritma A* adalah *complete* dan *optimal*[15]. 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[6], untuk lebih jelasnya terdapat pada gambar 2.12.



Gambar 2.12 Ilustrasi awal algoritma A*[6]

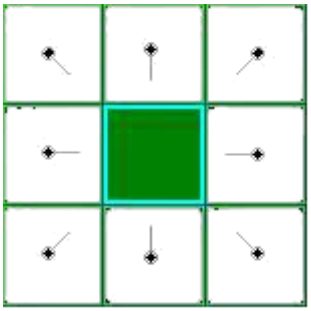
Hal pertama yang harus dilakukan adalah membuat sebuah *grid layout*, karena dalam pencarian rute pada algoritma A* akan lebih mudah apabila peta yang ada berbentuk kotak-kotak kecil, metode ini memudahkan pencarian dalam area yang didefinisikan sebagai array dua dimensi, setiap kotak memiliki status *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 2.13.



Gambar 2.13 Ilustrasi kotak awal sebagai *closed list*[6]

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

$$F = G + H.....[6]$$

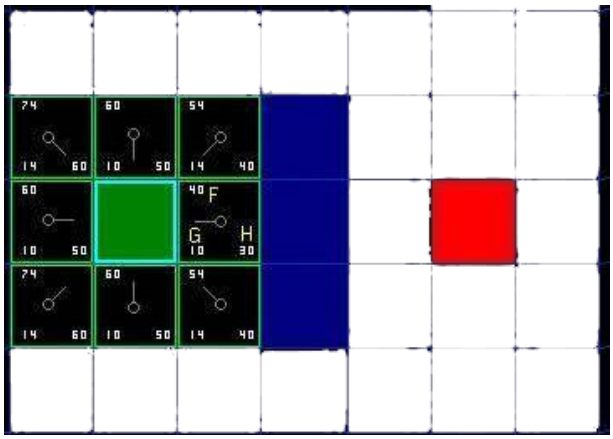
dimana,

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 2.14.



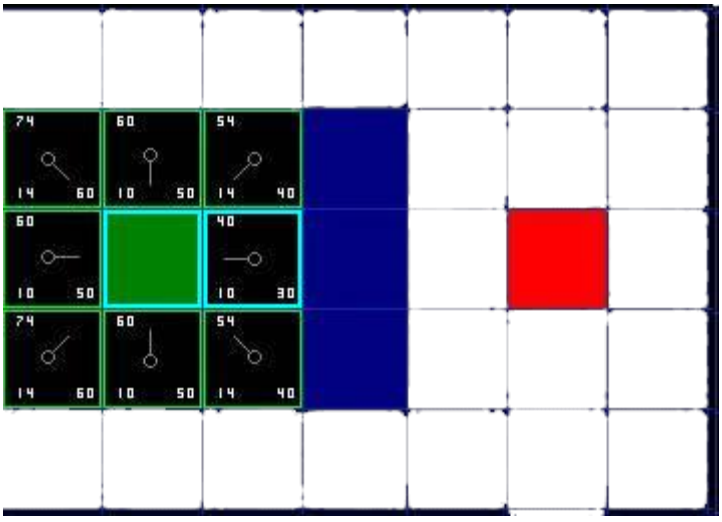
Gambar 2.14 Ilustrasi perhitungan F pada *open list*[6]

Langkah selanjutnya yang harus dilakukan adalah sebagai berikut :

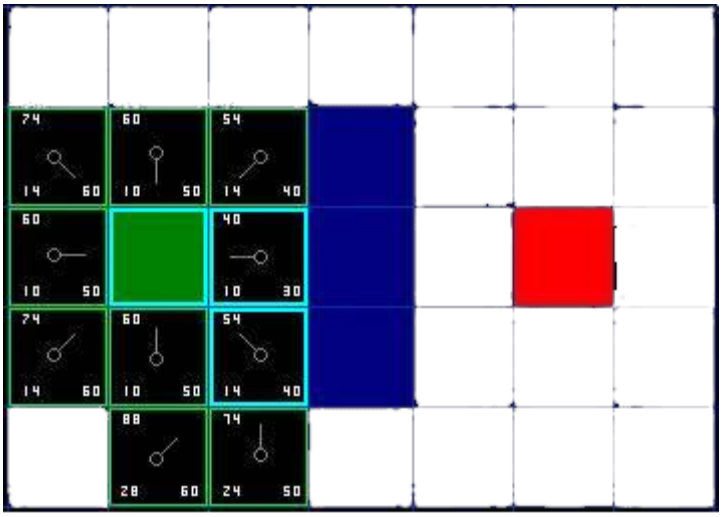
D. Pilih nilai F yang terkecil dari setiap kotak yang telah dihitung nilai F nya, kemudian keluarkan dari *open list* dan masukkan ke dalam *closed list.*

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.

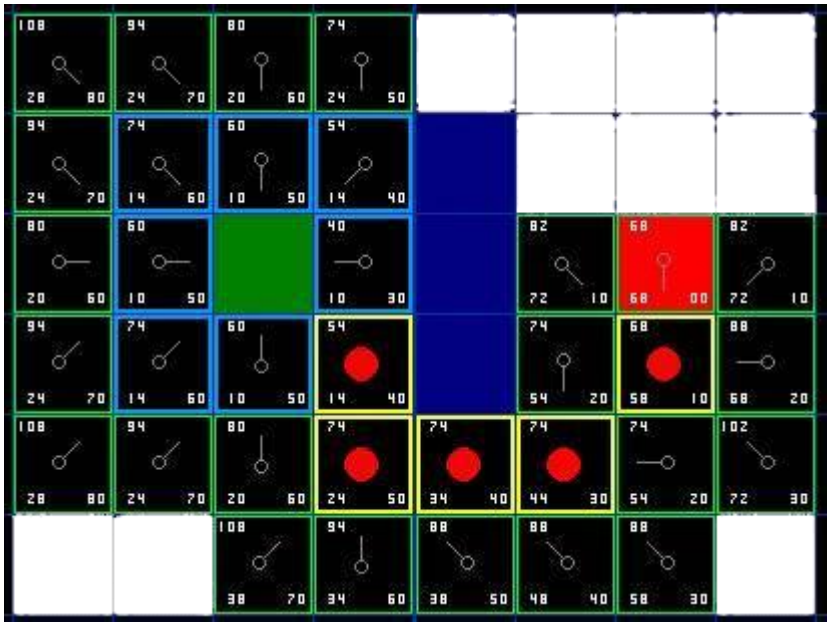


Gambar 2.15 Ilustrasi penentuan kotak selanjutnya[6]



Gambar 2.16 Ilustrasi perhitungan terhadap kotak *unwalkable*[6]

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 2.17.

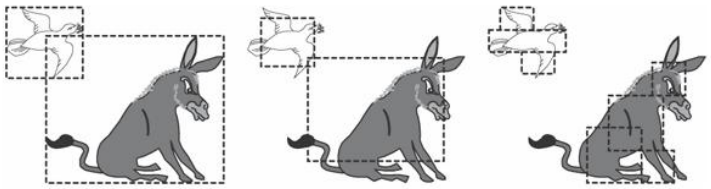


Gambar 2.17 Ilustrasi hasil akhir pencarian rute terpendek[6]

2.3.3 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*[2], dengan jarak respon yang telah diaplikasikan secara spesifik. Dalam *collision detection*, terdapat banyak sekali jenis dari *collision detection* itu sendiri, pada gambar 2.18 merupakan tiga tipe dari *collision detection*.



Gambar 2.18 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.

2.4 *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)[10]. 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 (*abstraction*), pengkapsulan (*encapsulation),* penurunan sifat (*inheritance*), dan polimorfisme (*polymorphism*) pada objek-objek yang dibentuk.

Object Oriented Programming (OOP) dibagi menjadi beberapa cirri utama[10], 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[10], 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 (*classes*) 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 d2mplementasikan 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[10].

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*[10].

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*)[10].

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*)[10]. Akan tetapi hal itu tentunya bergantung juga pada *access specifier* (yaitu, *public* dan *private*) yang diberikan dalam proses penurunan kelas.

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[11].

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.

2.5.1 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 d2kuti 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, d2kuti 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 Classes, 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.

2.5.2 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[11] :

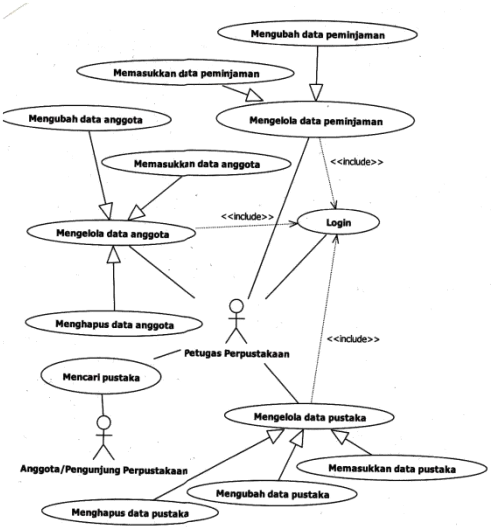
A. *Use Case Diagram*

Use Case atau diagram *use case* merupakan pemodelan yang digunakan untuk menggambarkan kelakuan (*behavior)* dari sistem yang akan dibuat[11]. *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*[11].

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 2.19.



Gambar 2.19 Contoh dari *Use Case Diagram*[11]

B. ***Activity Diagram***

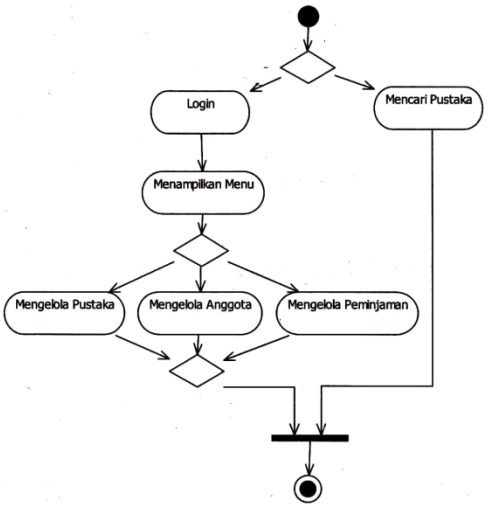
Diagram aktivitas atau *activity diagram* adalah sebuah diagram yang menggambarkan *workflow* (aliran kerja) atau aktivitas dari sebuah sistem atau proses bisnis[11]. Dalam diagram aktivitas yang perlu diperhatikan adalah bahwa diagram aktivitas menggambarkan aktivitas sistem, bukan apa yang dilakukan aktor, jadi aktivitas yang dapat dilakukan oleh sistem.

Diagram aktivitas juga banyak digunakan untuk mendefinisikan hal-hal berikut[11] :

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 2.20.



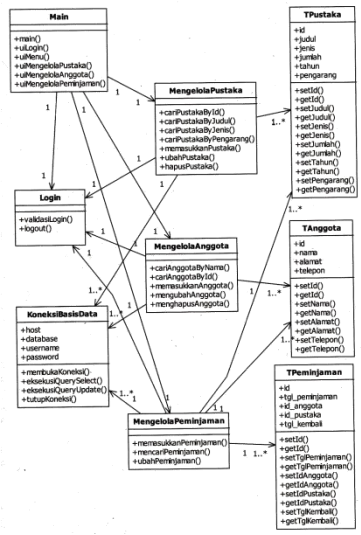
Gambar 2.20 Contoh dari *Activity Diagram*[11]

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[11].

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 2.21.

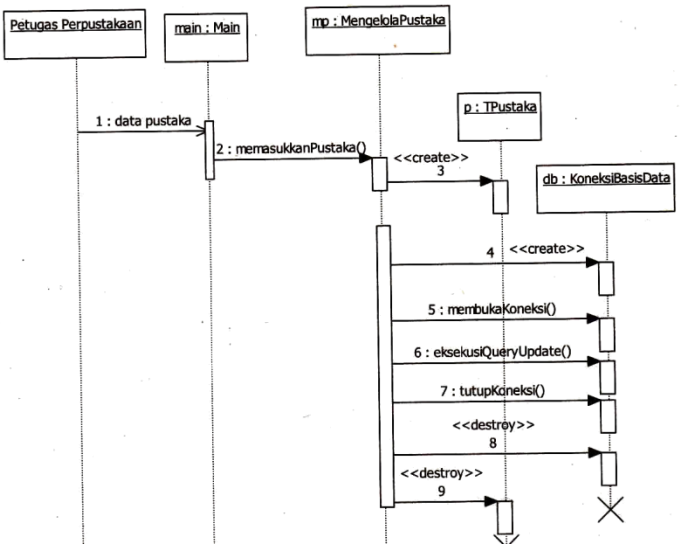


Gambar 2.21 Contoh dari *Class Diagram*[11]

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[11]. 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 2.22.



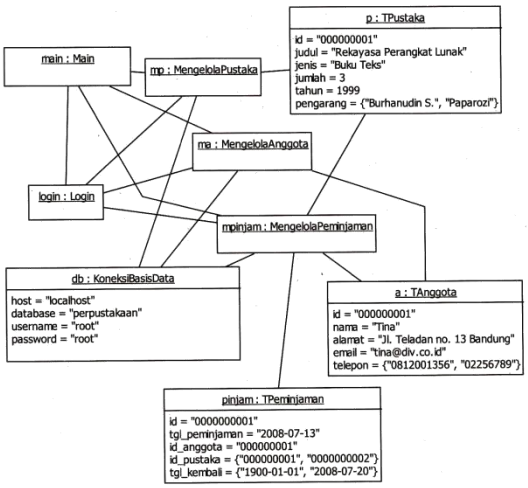
Gambar 2.22 Contoh dari *Sequence Diagram*[11]

E. ***Object Diagram***

Diagram objek menggambarkan struktur system dari segi penamaan objek dan jalannya objek dalam sistem[11]. 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 2.23.



Gambar 45 Diagram Objek Studi Kasus

Gambar 2.23 Contoh dari *Object Diagram*[11]

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[11]. Diagram komponen juga dapat digunakan untuk memodelkan hal-hal berikut[11] :

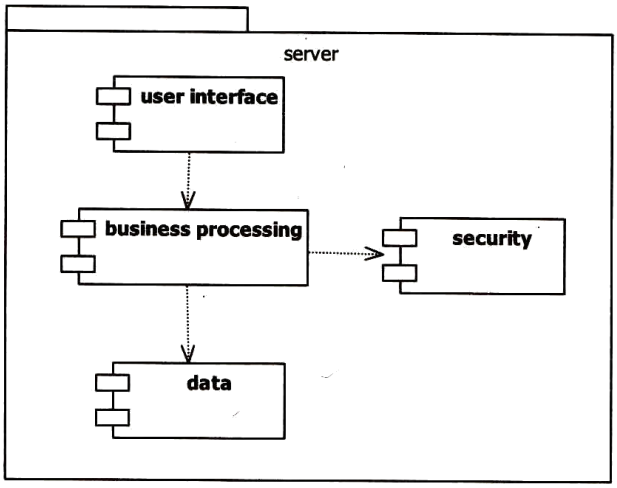
1. *Source code* program perangkat lunak
2. Komponen *executable* yang dilepas ke *user*
3. Basis data secara fisik
4. Sistem yang harus beradaptasi dengan sistem lain
5. *Framework* sistem

Adapun komponen-komponen dasar yang biasanya ada dalam suatu sistem adalah sebagai berikut[11] :

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 2.24.



Gambar 2.24 Contoh dari *Component Diagram*[11]

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[11]. Contoh penggunaan diagram ini misalnya untuk menggambarkan deskripsi dari setiap bagian mesin yang saling terkait *router* pada jaringan komputer, dll.

H. ***Package Diagram***

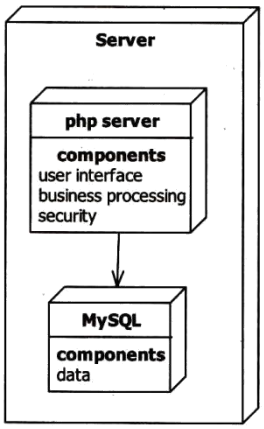
Package diagram menyediakan cara mengumpulkan elemen-elemen yang saling terkait dalam diagram UML[11].

I. ***Deployment Diagram***

Diagram deployment atau *deployment diagram* menunjukkan konfigurasi komponen dalam proses eksekusi aplikasi[11]. 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 2.25.



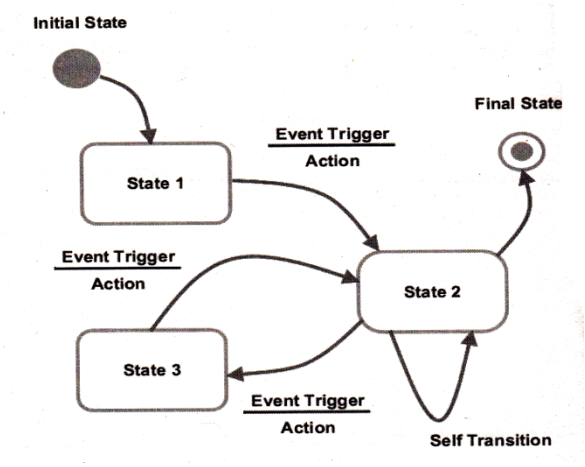
Gambar 2.25 Contoh dari *Deployment Diagram*[11]

J. ***State Machine Diagram***

Diagram mesin status digunakan untuk menggambarkan perubahan status atau transisi status dari sebuah mesin atau sistem[11]. 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[11].

Contoh dari *state machine diagram* dapat dilihat pada gambar 2.26.



Gambar 2.26 Contoh dari *State Machine Diagram*[11]

K. *Communication Diagram*

Diagram komunikasi merupakan penyederhanaan dari diagram kolaborasi (*collaboration diagram*)[11]. 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.

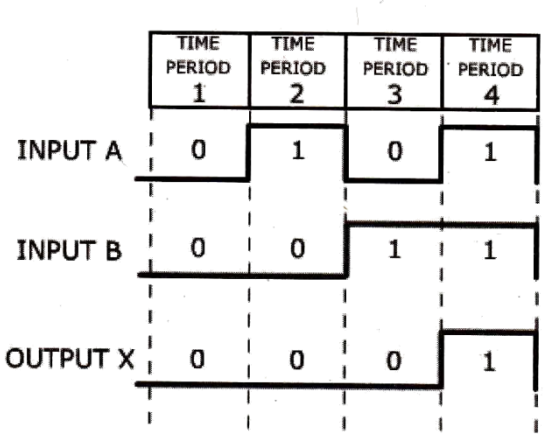
Diagram komunikasi mengelompokkan *message* pada kumpulan diagram sekuen menjadi sebuah diagram. Dalam diagram komunikasi yang dituliskan adalah operasi/metode yang dijalankan antara objek yang satu dan objek yang lainnya secara keseluruhan, oleh karena itu dapat diambil dari jalannya interaksi pada 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[11]. *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 2.27.



Gambar 2.27 Contoh dari *Timing Diagram*[11]

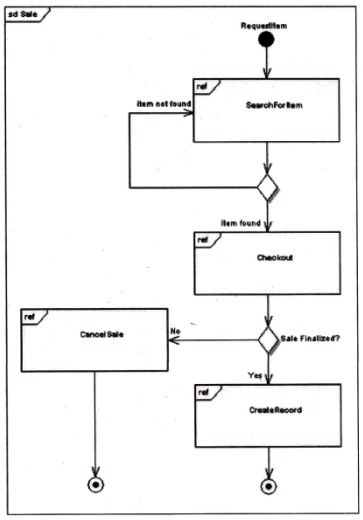
M. ***Iteraction Overview Diagram***

Iteraction overview diagram mirip dengan diagram aktivitas yang berfungsi untuk menggambarkan sekumpulan urutan aktivitas. *Iterraction overview diagram* adalah bentuk aktivitas diagram yang setiap titik merepresentasikan diagram interaksi[11]. Interaksi diagram dapat meliputi

diagram sekuen, diagram komunikasi, *interaction overview diagram*, dan *timing diagram*.

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

Contoh dari *overview diagram* dapat dilihat pada gambar 2.28.



Gambar 2.28 Contoh dari *Iteraction Overview Diagram*[11]

2.6 Teknik Pengujian Perangkat Lunak

Pengujian perangkat lunak adalah elemen kritis dari jaminan kualitas perangkat lunak dan merepresentasikan kajian pokok dari spesifikasi, desain, dan pengkodean[7].

2.6.1 **Pengujian *Black Box***

Menurut Roger S. Pressman[7], 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 :

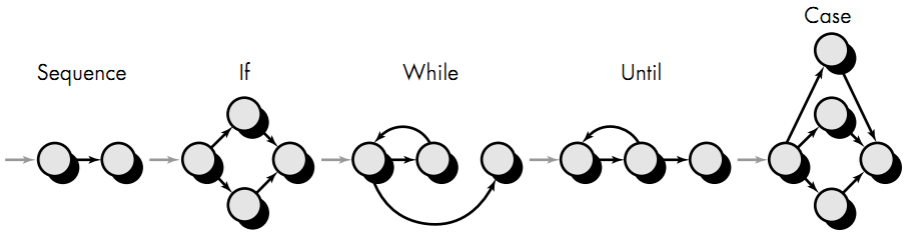
A. Fungsi-fungsi yang tidak benar atau hilang
B. Kesalahan dalam *interface*
C. Kesalahan dalam struktur data atau akses *database* eksternal
D. Kesalahan kinerja
E. Inisialisasi dan kesalahan terminasi

2.6.2 **Pengujian *White Box***

Menurut Roger S. Pressman[17], 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 2.29.



Gambar 2.29 Notasi Diagram Alir[7]

2.7 *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 *pirates of money*.

2.7.1 *Java*

Java adalah salah satu bahasa pemrograman komputer yang berorientasi objek[13], yang diciptakan oleh satu tim dari perusahaan *Sun Microsystem*, perusahaan *workstation* UNIX (*Sparc*) yang cukup terkenal. *Java* diciptakan

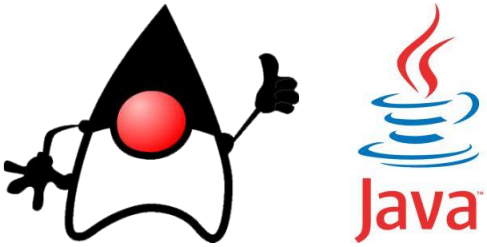
berdasarkan bahasa C++, dengan tujuan *platform independent* (dapat dijalankan pada berbagai jenis *hardware* tanpa kompilasi ulang), dengan slogan "*Write Once, Run Anywhere* (WORA), dan *Java* pada hakikatnya lebih sederhana dan memakai objek secara murni dibanding dengan bahasa pemrograman C++.

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 2.30.



Gambar 2.30 Maskot *Java* (Duke) dan *HotJava*[13]

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 *overwrited*, 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*.

2.7.2 *NetBeans* IDE

NetBeans adalah sebuah proyek *open source* yang didedikasikan untuk menyediakan produk pengembang perangkat lunak yang handal (*Netbeans* IDE dan *Netbeans Platform*) yang ditujukan untuk kebutuhan para pengembang, pengguna dan bisnis yang mengandalkan *netbeans* sebagai dasar produk mereka.

Pada bulan Juli 2000, *netbeans* dibuat *open source* oleh *Sun Microsystems*, yang menjadi sponsor proyek ini sampai Januari 2010 ketika *Sun Microsystems* menjadi anak perusahaan dari *Oracle*.