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 [7], 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. 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. Side Scrolling

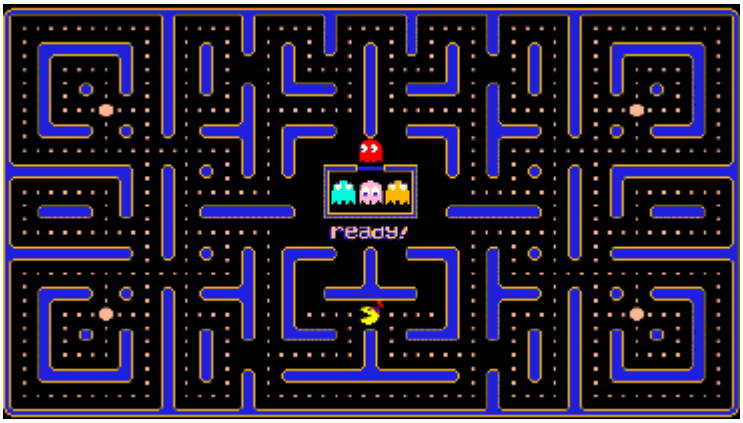
Genre game yang melibatkan karakter untuk bergerak dari kiri ke kanan dan lompat. Metode serangan jenis permainan ini umumnya melompat-lompat di kepala karakter musuh. Contoh *game*: Mario bros dan sonic seperti gambar berikut [8].



Gambar 2.1 Contoh Permainan *Side Scrolling*

B. Maze Game

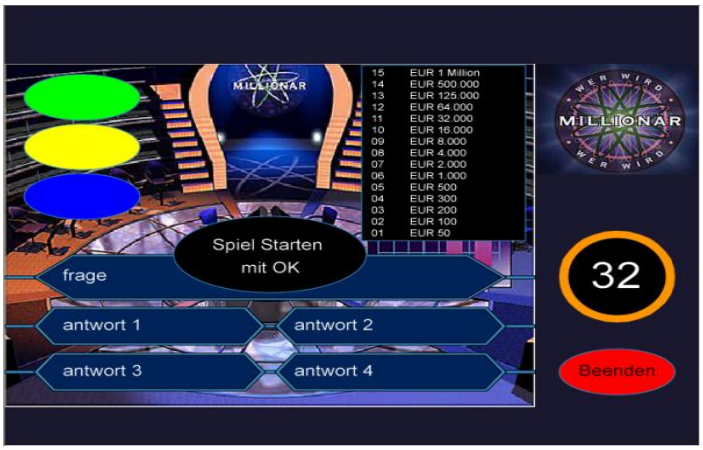
Genre game yang membutuhkan kecepatan berpikir dan bereaksi serta berunsur ketepatan menavigasi. Contoh *game*: *Pac-Man* dan *Bomberman* seperti gambar berikut [8].



Gambar 2.2 Contoh Permainan Maze Atau Labirin

C. Quiz Game

Game jenis ini biasanya mengajukan pertnyaan dan selalu menggunakan pilihan ganda. Game ini lebih berfokus pada pendidikan daripada hiburan. Contoh game ini adalah *who wants to be a milioner* Seperti gambar berikut [8].



Gambar 2.3 Contoh Game Quiz

2.2. Game Edukasi

Penerapan *game* edukasi bermula dari perkembangan *video game* yang sangat pesat dan menjadikannya sebagai media efektif yang interaktif dan banyak dikembangkan di perindustrian. Melihat kepopuleran *game* tersebut, para pendidik berpikir bahwa mereka mempunyai kesempatan yang baik untuk menggunakan

komponen rancangan *game* dan menerapkannya pada kurikulum dengan penggunaan industri berbasis *game*. *Game* harus memiliki desain antarmuka yang interaktif dan mengandung unsur menyenangkan.

2.2.1. Pengertian *Game* Edukasi

Game edukasi adalah salah satu bentuk *genre game* yang digunakan untuk memberikan pembelajaran kepada penggunanya melalui media permainan yang unik dan menarik. *Game* edukasi biasanya ditujukan untuk anak-anak sekolah dasar. Dalam proses kegiatan belajar mengajar, *game* edukasi dapat menarik minat anak sekolah sehingga menimbulkan suasana belajar yang menyenangkan dan dapat tercapainya tujuan pembelajaran secara tepat dan berhasil yang disesuaikan dengan situasi dan kondisi.

Game edukasi adalah *game* yang khusus dirancang untuk mengajarkan *user* suatu pembelajaran tertentu, pengembangan konsep dan pemahaman dan membimbing mereka dalam melatih kemampuan mereka, serta memotivasi mereka untuk memainkannya.

2.2.2 Kriteria *Game* Edukasi

Perancangan *game* edukasi yang baik haruslah memenuhi kriteria dari *game* edukasi itu sendiri. Berikut ini adalah beberapa kriteria dari sebuah *game* edukasi, yaitu:

1. Nilai Keseluruhan (*Overall Value*)

Nilai keseluruhan dari suatu *game* terpusat pada desain dan panjang durasi *game*. Aplikasi dibangun dengan desain yang menarik dan interaktif.

2. Dapat Digunakan (*Usability*)

Mudah digunakan dan diakses adalah poin penting bagi pembuat *game*. Aplikasi dirancang dengan *interface* yang *user friendly* sehingga pengguna dengan mudah dapat mengakses aplikasi.

3. Keakuratan (*Accuracy*)

Keakuratan diartikan sebagai bagaimana kesuksesan model/gambaran sebuah *game* dapat dituangkan ke dalam percobaan atau perancangannya.

4. Kesesuaian (*Appropriateness*)

Kesesuaian dapat diartikan bagaimana isi dan desain *game* dapat diadaptasikan terhadap keperluan pengguna dengan baik. Aplikasi menyediakan menu dan fitur yang diperlukan pengguna untuk membantu pemahaman pengguna dalam menggunakan aplikasi.

5. Relevan (*Relevance*)

Relevan artinya dapat mengaplikasikan isi *game* ke *target user*. Agar dapat relevan terhadap pengguna, sistem harus bisa membimbing dalam pencapaian tujuan pembelajaran.

6. Objektifitas (*Objectives*)

Objektifitas menentukan tujuan pengguna dan kriteria dari kesuksesan atau kegagalan. Dalam sebuah aplikasi, objektifitas adalah usaha untuk mempelajari hasil dari permainan.

7. Umpan Balik (*Feedback*)

Untuk membantu pemahaman pengguna bahwa permainan (*performance*) mereka sesuai dengan objek *game* atau tidak, *feedback* harus disediakan. Aplikasi menyajikan animasi dan efek suara yang mengindikasikan kesuksesan atau kegagalan permainan.

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, 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 [10], 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) [10], 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:

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* Biaya yang diperhitungkan didapat dari biaya sebenarnya ditambah dengan biaya perkiraan, dalam notasi matematika dituliskan sebagai berikut :

$$f(n) = g(n) + h(n).....$$

Keterangan:

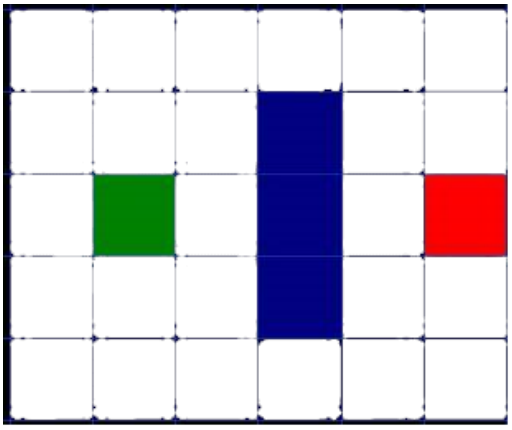
f(n) = fungsi evaluasi (jumlah g(n) dengan h(n))

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

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

Dengan perhitungan biaya seperti diatas, algoritma A* adalah *complete* dan *optimal*. 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, untuk lebih jelasnya terdapat pada gambar 2.4.



Gambar 2.4 Ilustrasi awal algoritma A*

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*.

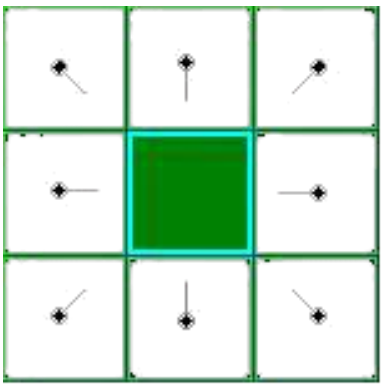
Pusat *point* pada algoritma A* dinamakan *node*, hal ini dikarenakan area yang dapat digunakan algoritma A* tidak hanya berbentuk kotak, akan tetapi dapat berbentuk yang lainnya, namun terkadang untuk memudahkan dalam perhitungan maka digunakanlah bentuk kotak.

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 2.5.



Gambar 2.5 Ilustrasi kotak awal sebagai *closed list*

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

$$F = G + H.....$$

dimana,

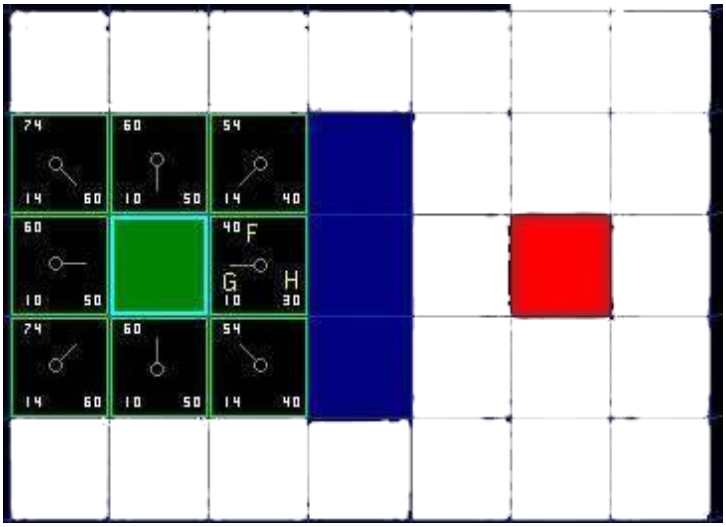
G : nilai yang dibutuhkan untuk menempuh jarak dari *point* awal menuju kotak selanjutnya

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 2.6.



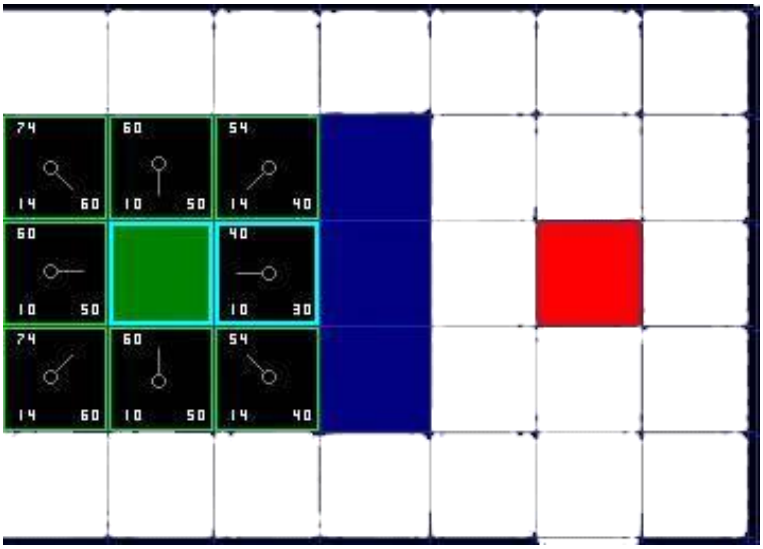
Gambar 2.6 Ilustrasi perhitungan F pada *open list*

Langkah selanjutnya yang harus dilakukan adalah sebagai berikut :

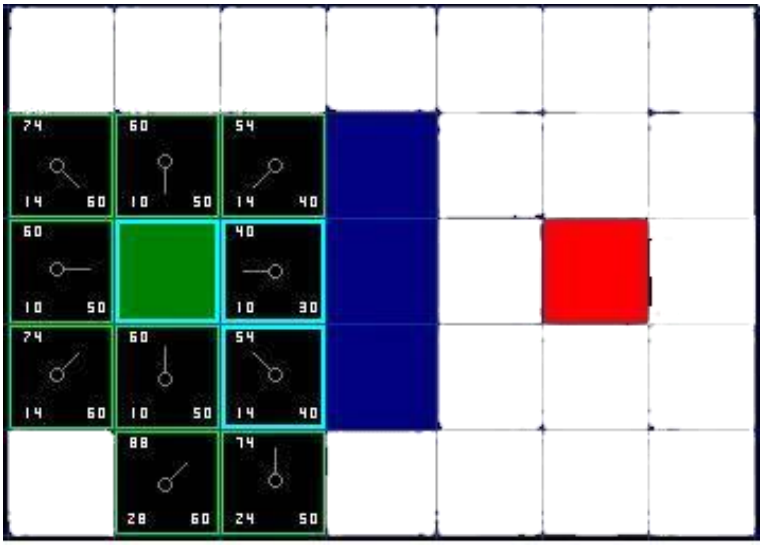
D. Pilih nilai F yang terkecil dari setiap kotak yang telah dihitung nilai F nya, kemudian keluarkan dari *open list* dan masukkan ke dalam *closed list.*

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.

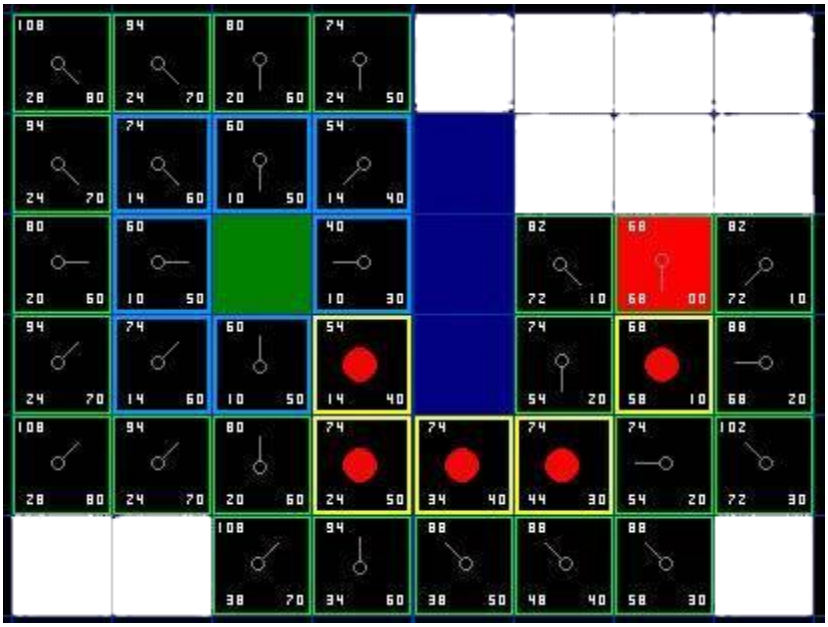


Gambar 2.7 Ilustrasi penentuan kotak selanjutnya



Gambar 2.8 Ilustrasi perhitungan terhadap kotak *unwalkable*

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 2.9.



Gambar 2.9 Ilustrasi hasil akhir pencarian rute terpendek

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) [11]. 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, 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 [11].

Perbedaan antara kelas (*class*) dengan objek (*object*) dalam OOP dibagi menjadi dua, 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 computer [11].

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*.

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*).

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.

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 [11].

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 :

A. ***Use Case Diagram***

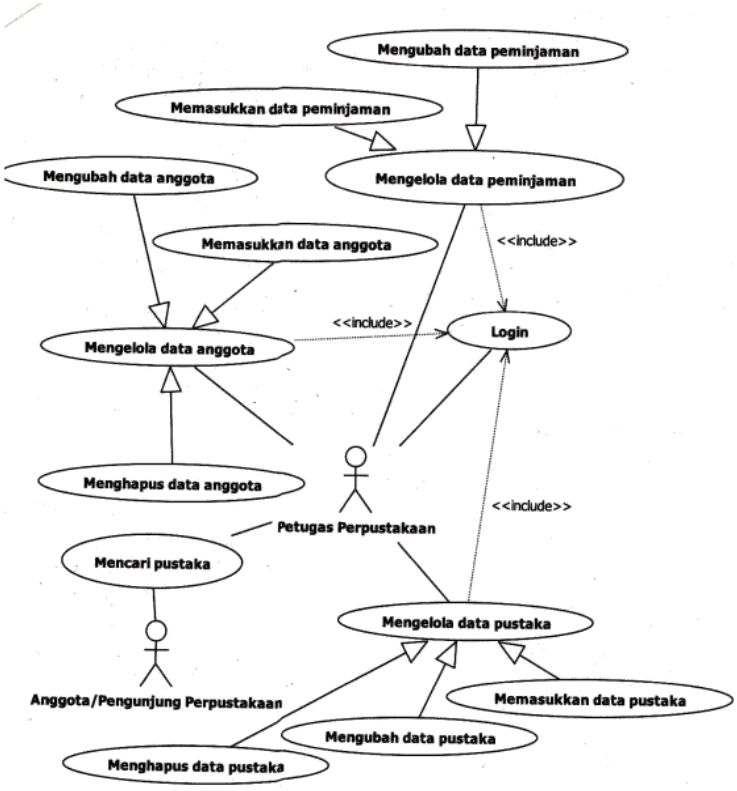
Use Case atau diagram *use case* merupakan pemodelan yang digunakan untuk menggambarkan kelakuan (*behavior)* dari sistem yang akan dibuat. *Use case* mendeskripsikan sebuah interaksi antara satu atau lebih aktor dengan sistem yang akan dibuat. Secara kasar, *use case* digunakan untuk mengetahui

fungsi apa saja yang ada di dalam sebuah sistem dan siapa saja yang berhak menggunakan fungsi-fungsi tersebut [11].

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*.

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 2.10.



Gambar 2.10 Contoh dari *Use Case Diagram*

B. ***Activity Diagram***

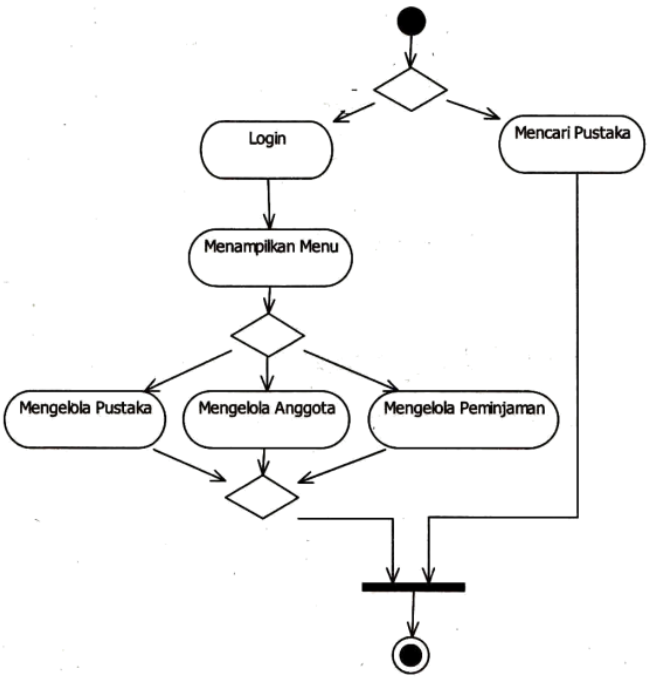
Diagram aktivitas atau *activity diagram* adalah sebuah diagram yang menggambarkan *workflow* (aliran kerja) atau aktivitas dari sebuah sistem atau

proses bisnis. 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 :

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 211.



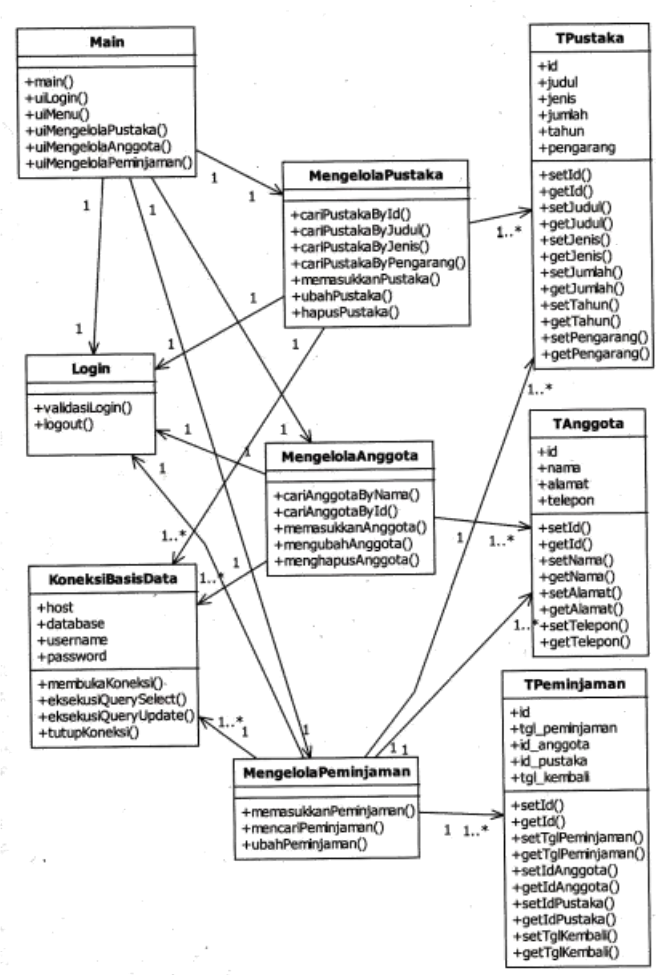
Gambar 2.11 Contoh dari *Activity Diagram*

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.

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 2.12.



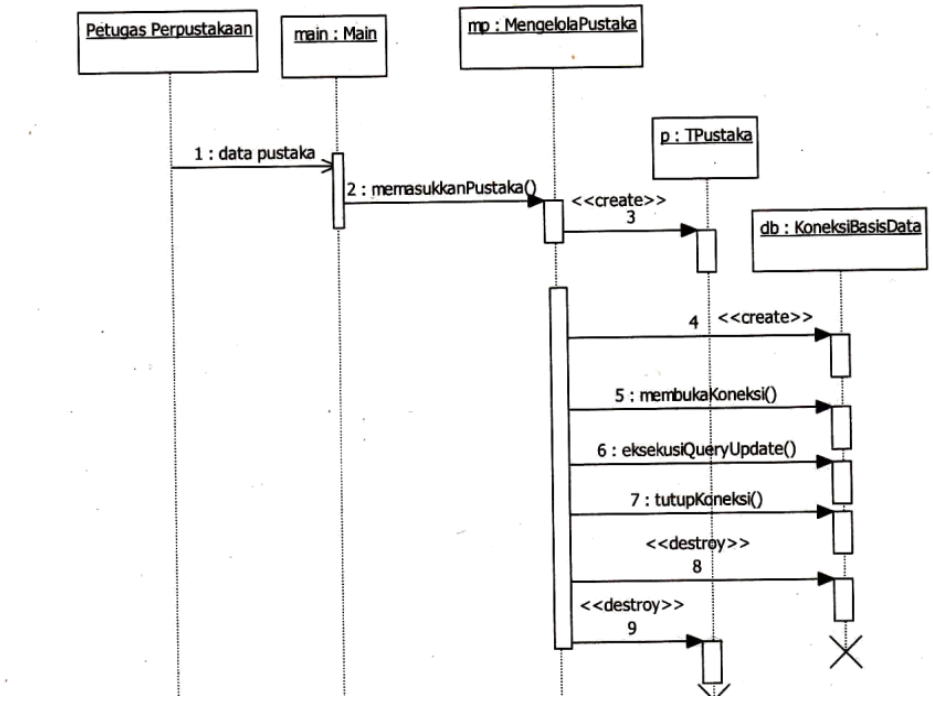
Gambar 2.12 Contoh dari *Class Diagram*

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. 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 2.13.



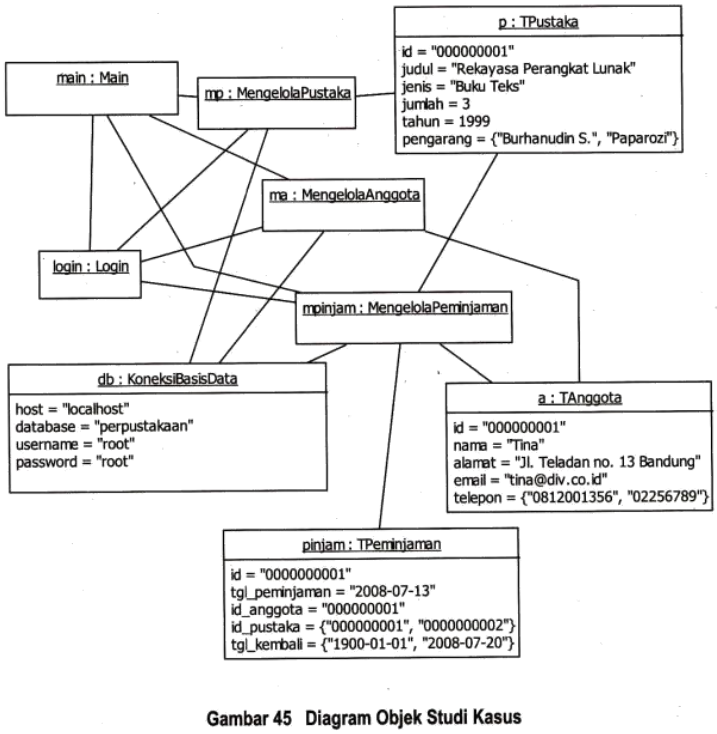
Gambar 2.13 Contoh dari *Sequence Diagram*

E. *Object Diagram*

Diagram objek menggambarkan struktur system dari segi penamaan objek dan jalannya objek dalam sistem. 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 2.14.



Gambar 45 Diagram Objek Studi Kasus

Gambar 2.14 Contoh dari *Object Diagram*

F. *Component Diagram*

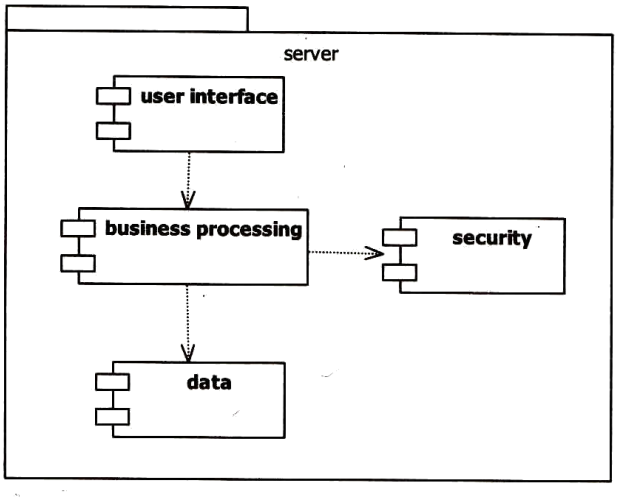
Diagram komponen dibuat untuk menunjukkan organisasi dan kebergantungan di antara kumpulan komponen dalam sebuah sistem. Diagram komponen focus pada komponen sistem yang dibutuhkan dan ada di dalam sistem. Diagram komponen juga dapat digunakan untuk memodelkan hal-hal berikut :

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 :

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 2.15.



Gambar 2.15 Contoh dari *Component Diagram*

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

H. ***Package Diagram***

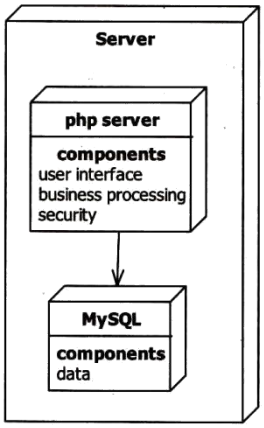
Package diagram menyediakan cara mengumpulkan elemen-elemen yang saling terkait dalam diagram UML.

I. ***Deployment Diagram***

Diagram deployment atau *deployment diagram* menunjukkan konfigurasi komponen dalam proses eksekusi aplikasi. 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 2.16.



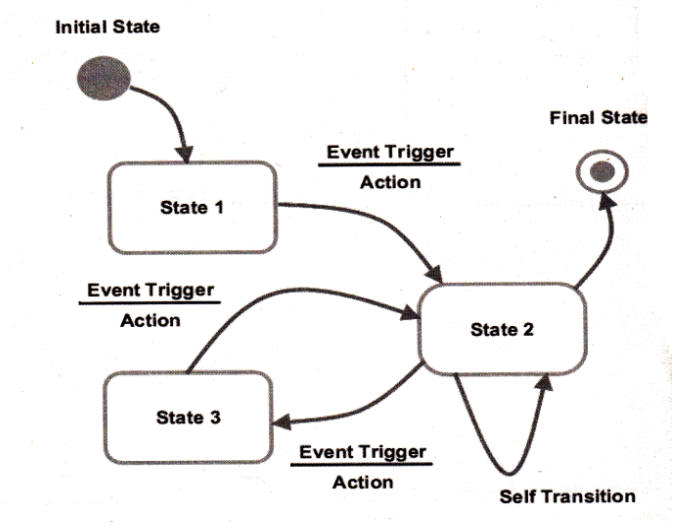
Gambar 2.16 Contoh dari *Deployment Diagram*

J. ***State Machine Diagram***

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

Contoh dari *state machine diagram* dapat dilihat pada gambar 2.17.



Gambar 2.17 Contoh dari *State Machine Diagram*

K. *Communication Diagram*

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

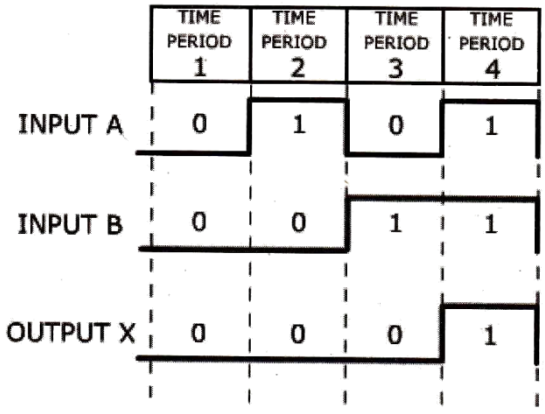
Diagram komunikasi mengelompokkan *message* pada kumpulan diagram sekuen menjadi sebuah diagram. Dalam diagram komunikasi yang dituliskan adalah operasi/metode yang dijalankan antara objek yang satu dan objek yang lainnya secara keseluruhan, oleh karena itu dapat diambil dari jalannya interaksi pada 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. *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 2.18.



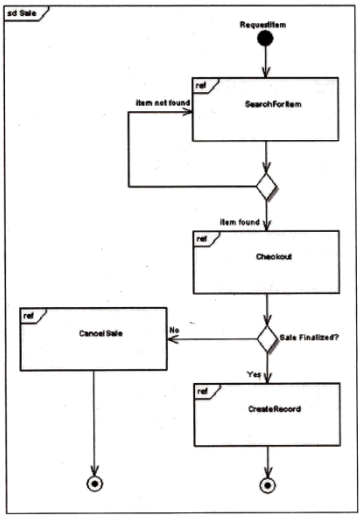
Gambar 2.18 Contoh dari *Timing Diagram*

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. 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 2.19.



Gambar 2.19 Contoh dari *Iteraction Overview Diagram*

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.

2.6.1 Pengujian *Black Box*

Menurut Roger S. Pressman [12], 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

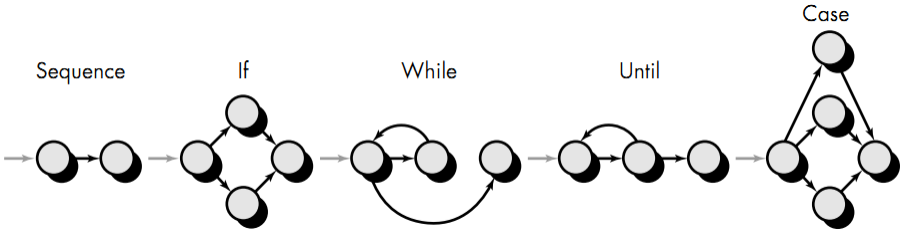
D. Kesalahan kinerja
E. Inisialisasi dan kesalahan terminasi

2.6.2 **Pengujian *White Box***

Menurut Roger S. Pressman[12], 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 2.20.



Gambar 2.20 Notasi Diagram Alir

2.7. Matematika

Matematika merupakan ilmu universal yang mendasari perkembangan teknologi modern, mempunyai peran penting dalam berbagai disiplin dan memajukan daya pikir manusia . Perkembangan pesat di bidang teknologi informasi dan komunikasi dewasa ini dilandasi oleh perkembangan matematika di bidang teori bilangan, aljabar, analisis, teori peluang dan matematika diskrit.

Untuk menguasai dan mencipta teknologi di masa depan diperlukan penguasaan matematika yang kuat sejak dini.

Mata pelajaran matematika perlu diberikan kepada semua peserta didik mulai dari sekolah dasar (SD) untuk membekali peserta didik dengan kemampuan berpikir logis, analitis, sistematis, kritis, dan kreatif, serta kemampuan bekerjasama. Kompetensi tersebut diperlukan agar peserta didik dapat memiliki kemampuan memperoleh, mengelola, dan memanfaatkan informasi untuk bertahan hidup pada keadaan yang selalu berubah, tidak pasti, dan kompetitif.
Evolusi matematika dapat dipandang sebagai sederetan abstraksi yang selalu bertambah banyak, atau perkataan lainnya perluasan pokok masalah. Abstraksi mula-mula, yang juga berlaku pada banyak binatang, adalah tentang bilangan: pernyataan bahwa dua apel dan dua jeruk (sebagai contoh) memiliki jumlah yang sama.

Selain mengetahui cara mencacah objek-objek fisika, manusia prasejarah juga mengenali cara mencacah besaran abstrak, seperti waktu (hari, musim, tahun). Aritmetika dasar (penjumlahan, pengurangan, perkalian, dan pembagian) mengikuti secara alami.

Belajar matematika sangat berguna untuk memecahkan permasalahan dikehidupan sehari-hari. Untuk itu belajar matematika harus dilakukan dengan sungguh- sungguh terutama pembelajaran di Kelas Tinggi. Kelas tinggi terdiri dari kelas 4, kelas 5 dan kelas 6. Pelajaran ini identik dengan hitung-menghitung. Dengan diajarkanya matematika, diharapkan peserta didik mampu melakukan operasi yang berhubungan dengan penghitungan bilangan.

Ruang lingkup mata pelajaran matematika yang diberikan pada satuan pendidikan SD/MI meliputi aspek-aspek bilangan, geometri, pengukuran, dan pengolahan data.

2.8. Pengertian Multimedia

Multimedia dapat diartikan sebagai penggunaan beberapa media yang berbeda untuk menggabungkan dan menyampaikan informasi dalam bentuk teks, *audio*, grafik, animasi, dan *video*.

Adapun definisi multimedia menurut beberapa ahli, yaitu:

1. Kombinasi dari komputer dan *video.*

2. Kombinasi dari tiga elemen, yaitu suara, gambar, dan teks.

3. Kombinasi dari paling sedikit dua media input atau output. Media ini dapat berupa *audio* (suara, musik), animasi, *video*, teks, grafik dan gambar.

4. Alat yang dapat menciptakan presentasi yang dinamis dan interaktif yang mengkombinasikan teks, grafik, animasi, *audio* dan *video.*

5. Multimedia adalah pemanfaatan komputer untuk membuat dan menggabungkan teks, grafik, *audio*, *video*, dengan menggunakan *tool* yang memungkinkan pemakai berinteraksi, berkreasi, dan berkomunikasi.dalam konteks komputer.

6. Multimedia merupakan kombinasi dari teks, gambar, seni grafik, suara, animasi dan elemen-elemen *video* yang dimanipulasi secara digital. Tampilan dan cita rasa dari proyek multimedia harus menyenangkan, estetis, mengundang dan mengikat. Proyek harus memuat konsistensi visual, hanya dengan menggunakan elemen-elemen yang mendukung pesan keseluruhan dari program.

2.8.1. Jenis Multimedia

Multimedia terbagi menjadi dua jenis, yaitu:

1. Multimedia Linier

Multimedia linier adalah suatu multimedia yang tidak dilengkapi oleh alat pengontrol apapun yang dapat dioperasikan oleh pengguna. Multimedia ini berjalan sekuensial (berurutan), contohnya: TV dan film.

2. Multimedia Interaktif

Multimedia interaktif dilengkapi oleh alat pengontrol yang dapat dioperasikan oleh pengguna, sehingga pengguna dapat memilih apa yang dikehendaki untuk proses selanjutnya. Contohnya: *video game*, pembelajaran interaktif, dan lain-lain.

2.8.2. Kriteria Multimedia

Ada beberapa kriteria multimedia menurut Sigit dkk., yaitu sebagai berikut:

1. Memilki lebih dari satu media yang konvergen, misalnya menggabungkan unsur *audio* dan visual.

2. Bersifat interaktif, dalam pengertian memiliki kemampuan untuk mengakomodasi respon pengguna.

3. Bersifat mandiri, dalam pengertian memberi kemudahan dan kelengkapan isi sedemikian rupa sehingga pengguna bisa menggunakan tanpa bimbingan orang lain.

2.8.3. Kelebihan Pembelajaran Menggunakan Multimedia

Multimedia memiliki beberapa keuntungan, yaitu:

1. Multimedia masuk akal sehingga dapat meningkatkan pembelajaran.

2. Multimedia meningkatkan ekspresi diri dengan membiarkan pelajar untuk memutuskan sendiri.

3. Multimedia membuat pelajar menjadi pemilik sehingga mereka bisa menciptakan apa yang hendak mereka pelajari.

4. Multimedia menciptakan suasana yang aktif, sehingga pelajar dapat terlibat langsung.

5. Multimedia dapat menjembatani komunikasi pelajar dengan instruktur.

6. Pemakaian multimedia sudah tidak asng lagi karena telah digunakan dalam kehidupan sehari-hari seperti *video game* dan televisi.

2.9 *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 *Fun Math For Smart.*

2.9.1 Html 5

HTML5 adalah Generasi selanjutnya dari HTML sebelumnya. HTML5 diciptakan dalam pengembangan bahasa HTML untuk memperbaiki konten dan memperbarui teknologi multimedia yang sudah ada pada HTML 4 dan versi sebelumnya agar mudah dijalankan oleh browser dan mudah dimengerti. HTML5 merupakan kerjasama antara World Wide Web Consortium (W3C) dan Web Hypertext Application Teknologi Working Group [13]. (WHATWG). WHATWG bekerja dengan bentuk web dan aplikasi, dan W3C bekerja dengan XHTML 2.0. Pada tahun 2006, mereka memutuskan untuk bekerja sama dan membuat versi baru dari HTML yaitu HTML5.

2.9.2 Javascript

Javascript adalah bahasa skrip yang ditempelkan pada kode HTML dan diproses di sisi klien. Dengan adanya bahasa ini, kemampuan dokumen HTML menjadi semakin luas. Sebagai contoh, dengan menggunakan JavaScript dimungkinkan untuk memvalidasi masukan masukan pada formulir sebelum formulir dikirimkan ke server.

Javascript bukanlah bahasa Java dan merupakan dua bahasa yang berbeda. Javascript diinterpretasikan oleh klien (kodenya bisa dilihat pada sisi klien), sedangkan kode Java dikompilasi oleh pemrogram dan hasil kompilasinyalah yang dijalankan oleh klien [14].

2.9.3 Construct 2

Construct 2 adalah game editor berbasis HTML5 dikembangkan oleh perusahaan yang bernama Scirra Ltd. Construct 2 ditujukan terutama untuk non-programmer untuk membuat game menggunakan visual editor dan berbasis sistem logika. Construct 2 adalah hasil pengembangan dari *Construct Classic*. Construct 2 juga memiliki *famework Javascript* untuk programmer untuk menggunakan bersama editor *drag-and-drop*.