BAB 2
LANDASAN TEORI

2.1. *Video game*

*Video game*s adalah permainan elektronik yang melibatkan interaksi antarmuka dengan pengguna untuk menghasilkan umpan balik secara visual pada perangkat video. Kata video pada *Video game* tradisional disebut perangkat layar raster. Namun dengan semakin dipakainya istilah "*Video game*", kini kata permainan video dapat digunakan untuk menyebut permainan pada perangkat layar apapun. Sistem elektronik yang digunakan untuk bermain *Video game* dikenal sebagai platform, contoh ini adalah komputer pribadi dan konsol permainan video. Platform ini dari tingkatan besar seperti komputer mainframe sampai yang kecil seperti perangkat mobile. *Video game* khusus seperti *game* arcade, sementara sebelumnya umum, telah berangsur-angsur menurun digunakan. *Video game* telah pergi untuk menjadi sebuah bentuk seni dan industry [4].

Sebuah *video game* adalah permainan yang biasanya melibatkan *player* berinteraksi dengan alat pengendali untuk menghasilkan umpan balik secara visual dalam sebuah layar video. Perangkat input yang digunakan untuk memanipulasi permainan video disebut pengendali permainan, dan bervariasi di seluruh platform. Sebagai contoh, controller mungkin hanya terdiri dari sebuah tombol dan joystick, sementara yang lain mungkin mempunyai selusin tombol dan satu atau lebih joystick.

Video game biasanya menggunakan sarana tambahan menyediakan interaktivitas dan informasi untuk pemain. Audio hampir universal, menggunakan perangkat reproduksi suara, seperti speaker dan headphone. Umpan balik lain mungkin datang melalui peripheral haptic, seperti getaran atau kekuatan, dengan getaran kadang-kadang digunakan untuk mensimulasikan umpan balik kekuatan.

.

2.1.1 Pengertian *Game*

Pengertian *game* menurut beberapa ahli [4] :

1. Menurut Agustinus Nilwan dalam bukunya Pemrograman Animasi dan *Game* Profesional terbitan Elex Media Komputindo, *game* merupakan permainan komputer yang dibuat dengan teknik dan metode animasi. Jika ingin mendalami pengunaan animasi haruslah memahami pembuatan *game*. Atau jika ingin membuat *game*, maka haruslah memahami teknik dan metode animasi, sebab keduanya saling berkaitan.
2. Menurut Clark C. Abt, *Game* adalah kegiatan yang melibatkan keputusan pemain, berupaya mencapai tujuan dengan dibatasi oleh konteks tertentu (misalnya, dibatasi oleh peraturan).
3. Menurut Chris Crawford, seorang computer *game* designer mengemukakan bahwa *game*, pada intinya adalah sebuah interaktif, aktivitas yang berpusat pada sebuah pencapaian, ada pelaku aktif *(player),* ada pelaku pasif *(NPC*).
4. Menurut David Parlett, *Game* adalah sesuatu yang memiliki "akhir dan cara mencapainya" : artinya ada tujuan, hasil dan serangkaian peraturan untuk mencapai keduanya.
5. Menurut Roger Caillois, seorang sosiolog Perancis, dalam bukunya yang berjudul Les jeux et les hommes menyatakan *game* adalah aktivitas yang mencakup karakteristik berikut: *fun* (bebas bermain adalah pilihan bukan kewajiban), *separate* (terpisah), *uncertain, non-productive, governed by rules* (ada aturan), *fictitious* (pura-pura).
6. Menurut Greg Costikyan, *Game* adalah “sebentuk karya seni di mana peserta, yang disebut Pemain, membuat keputusan untuk mengelola sumberdaya yang dimilikinya melalui benda di dalam *game* demi mencapai tujuan”.

2.1.2 Jenis – Jenis *Game*

Berikut ini akan dijabarkan beberapa jenis *game* berdasarkan cara pembuatannya, cara pemasarannya, mesin yang menjalankannya. Jenis *game* diantaranya adalah [4] :

1. *Game* PC

   Game yang dimainkan pada PC (Personal Computer) yang memiliki kelebihan yaitu memiliki tampilan antarmuka yang baik untuk input maupun output, output visual kualitas tinggi karena layar computer biasanya memiliki resolusi yang jauh lebih tinggi dibandingkan dengan layar televise biasa.

2. *Game* Console

   Game yang dijalankan pada suatu mesin spesifik yang biasanya tersedia di rumah pribadi, seperti Microsoft Xbox 360, Nintendo Wii dan sebagainya.

3. *Game* Arcade

   Game yang dijalankan pada mesin dengan input dan output audio visual yang telah terintegrasi dan tersedia ditempat-tempat umum, seperti mal, bandara dan sebagainya.

4. *Game* Online

   Game yang hanya dapat dimainkan secara online melalui LAN atau internet.

2.1.3 *Genre Game*

Chris Crawford mencatat bahwa "keadaan desain komputer game berubah dengan cepat karena itu kita akan mengharapkan taksonomi disajikan di sini untuk menjadi usang atau tidak memadai dalam waktu singkat. Hampir semua jenis klasifikasi genre, soal genre tertentu setiap video game individu terbuka untuk interpretasi pribadi. Selain itu, penting untuk dapat memikirkan setiap permainan individu sebagai milik beberapa genre sekaligus, Game dibagi menjadi beberapa genre, berikut ini adalah macam-macam genre game [4]:

1. *Action Shooting* (tembak – menembak)

   Menembak , memukul , bisa juga menebas, tergantung cerita dan tokoh di dalamnya, *Video game* jenis ini sangat memerlukan kecepatan refleks, koordinasi mata-tangan, juga timing, inti dari *game* jenis ini adalah tembak menembak.

   Contoh : GTA dan Crysis.

2. *Fighting* (pertarungan)

   Ada yang mengelompokan *Video game* fighting di bagian Aksi, namun sebenarnya berbeda, jenis ini memang memerlukan kecepatan refleks dan koordinasi mata-tangan, tetapi inti dari *game* ini adalah penguasaan jurus (hafal caranya dan lancar mengeksekusinya), pengenalan karakter dan timing sangatlah penting untuk mengalahkan lawan secepat mungkin.

   Contoh : Mortal Kombat dan Tekken.

3. *Adventure* (Petualangan)

   Memasuki gua bawah tanah, melompati bebatuan di antara lahar, bergelayutan dari pohon satu ke pohon lain, bergulat dengan ular sambil mencari kunci untuk membuka pintu kuil legendaris, atau sekedar mencari telepon umum untuk mendapatkan misi berikutnya, itulah beberapa dari banyak hal yang karakter pemain harus lakukan dan lalui dalam *Video game* jenis ini.

   Contoh : Kings Quest, dan Space Quest.

4. *Strategy* (strategi)

   Video game strategi biasanya memberikan pemain atas kendali tidak hanya satu orang tapi minimal sekelompok orang dengan berbagai jenis tipe kemampuan, sampai kendaraan, bahkan hingga pembangunan berbagai bangunan, pabrik dan pusat pelatihan tempur, tergantung dari tema ceritanya. Kebanyakan *game* stategi adalah *game* perang. Contoh : Warcraft.

5. *Simulation* (Simulasi)

   Video game jenis ini seringkali menggambarkan dunia di dalamnya sedekat mungkin dengan dunia nyata dan memperhatikan dengan detil

berbagai faktor. Dari mencari jodoh dan pekerjaan, membangun rumah, gedung hingga kota, mengatur pajak dan dana kota hingga keputusan memecat atau menambah karyawan. Dunia kehidupan rumah tangga sampai bisnis membangun konglomerasi, dari berjualan limun pinggir jalan hingga membangun laboratorium cloning. *Video game* jenis ini membuat pemain harus berpikir untuk mendirikan, membangun dan mengatasi masalah dengan menggunakan dana yang terbatas.

6. *Puzzle* (menyusun)

   Video game jenis ini sesuai namanya berintikan mengenai pemecahan teka-teki, baik itu menyusun balok, menyamakan warna bola, memecahkan perhitungan matematika, melewati labirin, sampai mendorong-dorong kota masuk ke tempat yang seharusnya, itu semua termasuk dalam jenis ini. Sering pula *game* jenis ini adalah juga unsur *game* dalam *Video game* petualangan maupun *game* edukasi

7. *Sport game* (Olahraga)

   Game ini merupakan adaptasi dari kenyataan, membutuhkan kelincahan dan juga strategi dalam memainkannya. *Game* berupa kompetisi antara dua pemain atau lebih, di mana pemain dapat berupa individual atau tim. Contoh *game* tipe ini antara lain sepakbola, bola basket, tenis, dan bilyard.

8. *RPG* (Role Playing *Game*)

   Video game jenis ini sesuai dengan terjemahannya, bermain peran, memiliki penekanan pada tokoh/peran perwakilan pemain di dalam *game*, yang biasanya adalah tokoh utamanya, dimana seiring permainannya, karakter tersebut dapat berubah dan berkembang ke arah yang diinginkan pemain dalam berbagai parameter yang biasanya ditentukan dengan naiknya level, baik dari status kepintaran, kecepatan dan kekuatan karakter, senjata yang semakin sakti, ataupun jumlah teman maupun mahluk peliharaan.

9. *Education* (edukasi)

   Game edukasi merupakan paket software yang menciptakan kemampuan pada lingkungan game yang diberikan sebagai alat bantu untuk memotivasi atau membantu siswa untuk melalui prosedur *game* secara teliti untuk

mengembangkan kemampuannya. Developer yang membuatnya, harus memperhitungkan berbagai hal agar game ini benar-benar dapat mendidik, menambah pengetahuan dan meningkatkan ketrampilan yang memainkannya. Target segmentasi pemain harus pula disesuaikan dengan tingkat kesulitan dan design visual ataupun animasinya

2.1.4 Kecerdasan Buatan

Kecerdasan Buatan (bahasa Inggris: *Artificial Intelligence* atau *AI*) didefinisikan sebagai kecerdasan yang ditunjukkan oleh suatu *entitas* buatan. Sistem seperti ini umumnya dianggap komputer. Kecerdasan diciptakan dan dimasukkan ke dalam suatu mesin (komputer) agar dapat melakukan pekerjaan seperti yang dapat dilakukan manusia. Beberapa macam bidang yang menggunakan kecerdasan buatan antara lain sistem pakar, permainan komputer (*games*), logika *fuzzy*, jaringan syaraf tiruan dan robotika [3].

Banyak hal yang kelihatannya sulit untuk kecerdasan manusia, tetapi untuk Informatika relatif tidak bermasalah. Seperti contoh: mentransformasikan persamaan, menyelesaikan persamaan integral, membuat permainan catur atau Backgammon. Di sisi lain, hal yang bagi manusia kelihatannya menuntut sedikit kecerdasan, sampai sekarang masih sulit untuk direalisasikan dalam Informatika. Seperti contoh: Pengenalan Obyek/Muka, bermain sepak bola.

AI memiliki konotasi fiksi ilmiah yang kuat, AI membentuk cabang yang sangat penting pada ilmu komputer, berhubungan dengan perilaku, pembelajaran dan adaptasi yang cerdas dalam sebuah mesin.

Penelitian dalam AI menyangkut pembuatan mesin untuk mengotomatisasikan tugas-tugas yang membutuhkan perilaku cerdas. Termasuk contohnya adalah pengendalian, perencanaan dan penjadwalan, kemampuan untuk menjawab diagnosa dan pertanyaan pelanggan, serta pengenalan tulisan tangan, suara dan wajah. Hal-hal seperti itu telah menjadi disiplin ilmu tersendiri, yang memusatkan perhatian pada penyediaan solusi masalah kehidupan yang nyata. Sistem AI sekarang ini sering digunakan dalam bidang ekonomi, obat-obatan,

teknik dan militer, seperti yang telah dibangun dalam beberapa aplikasi perangkat lunak komputer rumah dan *Video game*.

2.1.5 Algoritma Djikstra

Algoritma Dijkstra ditemukan oleh Edsger W. Djikstra yang merupakan salah satu varian bentuk algoritma popular dalam pemecahan persoalan yang terkait optimasi dan bersifat sederhana. Algoritma ini menyelesaikan masalah mencari sebuah lintasan terpendek (sebuah lintasan yang mempunyai panjang minimum) dari verteks a ke verteks z dalam graph berbobot, bobot tersebut adalah bilangan positif jadi tidak dapat dilalui oleh node negatif [5].

Algoritma Djikstra melibatkan pemasangan label pada verteks. Misalkan L(v) menyatakan label dari verteks v. pada setiap pembahasan beberapa verteks mempunyai label sementara dan yang lain mempunyai label tetap. Misalkan T menyatakan himpunan verteks yang mempunyai label sementara. Dalam menggambarkan algoritma tersebut verteks-verteks yang mempunyai label tetap akan dilingkari. Selanjutnya, jika L(v) adalah label tetap dari verteks v, maka L(v) merupakan panjang lintasan terpendek dari a ke v. Sebelumnya semua verteks mempunyai label sementara. Setiap iterasi dari algoritma tersebut merubah status label dari sementara menjadi tetap, sehingga algoritma dapat berakhir ketika z menerima sebuah label tetap. Pada bagian ini L(z) merupakan panjang lintasan terpendek dari a ke z. pada algoritma Djikstra node digunakan, karena algoritma Djikstra menggunakan digram pohon (*tree*) untuk penentuan jalur lintasa terpendek dan menggunakn *graph* yang berarah.

Algoritma Djikstra mencari panjang lintasan terpendek dari verteks a ke z dalam sebuah *graph* berbobot tersambung.

Langkah-langkah dalammenentukan lintasan terpendek pada algoritma Djikstra yaitu:

1. Pada awalnya pilih node dengan bobot yang terendah dari node yang belum terpilih, diinisialisasikan dengan ‘0’ dan yang sudah terpilih diinisialisasikan dengan ‘1’.
2. Bentuk tabel terdiri dari node, status, bobot dan *predecessor*. Lengkapi kolombobot yang diperoleh dari jarak node sumber ke semua node yang langsung terhubung dengan node tersebut.
3. Jika node sumber ditemukan maka tetapkan sebagai node terpilih.
4. Tetapkan node terpilih dengan label permanen dan perbaharui node yang langsung terhubung
5. Tentukan node sementara yang tehubung pada node yang sudah terpilih sebelumnya dan merupakan bobot terkecil di lihat dari tabel dan tentukan sebagai node terpilih berikutnya.
6. Apakah node terpilih merupakan node tujuan? Jika ya, maka kumpulkan node terpilih atau *predecessor* merupakan rangakaian yang menunjukan lintasan terpendek.
7. Begitu seterusnya hinga semua node terpilih

Procedure algoritma Djikstra adalah sebagai berikut:

Misal |1<|2<|3, merupakan adj vertek dari s ke v1,v2,v3 dan dikatakan bahwa sp dari s-V1 adalah |1, serta berlaku untuk semua part dari ske setiap v dalam G, karean | adalah bernilai positif, dan dapat dinotasikan distance/jarak pada d[v1]=|1,d[v2]<=|2, dst.

1. D[v1] = 0.7, karena hanya terdapat 1 path
2. D[2] = 1.1 → 1.2, ada path lain (s-v1-v3) tapi tidak update
3. D[3] =2.3 → 1, ada path lain (s-v1-v3)
    - Path d[2] dan d[3] mengalami update untuk mendapatkan shortest path
4. D[4] = 1.1
5. D[5] = 1.2

```
Procedure Djikstra;
{ djikstra menghitung cost shortest path dr vertex
   1 ke tiap vertex dari directed graph }
Begin
S := {1};
for i : = 2 to n do
    D [i] :=  C [1, i] ; {inisialisai D}
for I := 1 to n-1 do begin
    Pilih vertex w dalam V-S sedemikian sehingga D
    [W] adalah minimum ;
    Tambahkan w ke S;
    for tiap vertex v dalam V-S do
    D[v] := min (D[v], D[w] + c[w,  v])
  end
end; {Djikstra}
```

Jika menggunakan algoritma Djikstra untuk menentukan jalur terpendek dari suatu *graph*, maka akan menemukan jalur yang terbaik, karena pada waktu penentuan jalur yang akan dipilih, akan dianlisis bobot dari node yang belum terpilih, lalu dipilih node dengan bobot yang terkecil. Jika ternyata ada bobot yang lebih kecil melalu node tertentu, maka bobot akan dapat berubah. Algoritma

Djikstra akan berhenti ketik semua node sudah terpilih, dan dengan algoritma Djikstra ini dapan menemukan jarak terpendek dari seluruh node, tidak hanya untuk node dari asal dan tertentu saja.

Algoritma Djikstra menggunakan waktu sebesar O(V*logV+E) dimana V dan E adalah banyaknya verteks dan *are*. Kompleksitas algoritma Djikstra adalah $O(n^2)$. sehingga untuk mencari semua pasangan verteks terpendek, total waktu asimptotik komputasinya adalah T(n)=n. $O(n^2)= O(n^3)$, algoritma Djikstra lebih menguntungkan dari sisi running time [5].

2.1 Aplikasi Pembangun Perangkat Lunak

Aplikasi yang digunakan untuk membangun aplikasi ini adalah OOP, JAVA, UML, Greenfoot dan Corel Draw. Berikut ini penjelasannya :

2.4.1 OOP (*Object Oriented Programming*)

Pemrograman berorientasi objek (Inggris : *object-oriented programming* disingkat OOP) merupakan paradigma pemrograman yang berorientasikan kepada objek. Semua data dan fungsi di dalam paradigma ini dibungkus dalam *kelas-kelas* atau *objek-objek*. Bandingkan dengan logika pemrograman terstruktur. Setiap objek dapat menerima pesan, memproses data, dan mengirim pesan ke objek lainnya.

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

Konsep dasar dari Pemrograman Berorientasi Objek Pemrograman orientasi-objek menekankan konsep berikut [6] :

1. Kelas kumpulan atas definisi data dan fungsi-fungsi dalam suatu unit untuk suatu tujuan tertentu. Sebagai contoh '*class of dog*' adalah suatu unit yang terdiri atas definisi-definisi data dan fungsi-fungsi yang menunjuk pada

berbagai macam perilaku/turunan dari anjing. Sebuah class adalah dasar dari modularitas dan struktur dalam pemrograman berorientasi object. Sebuah *class* secara *tipikal* sebaiknya dapat dikenali oleh seorang *non-programmer* sekalipun terkait dengan domain permasalahan yang ada, dan kode yang terdapat dalam sebuah class sebaiknya (relatif) bersifat mandiri dan independen (sebagaimana kode tersebut digunakan jika tidak menggunakan OOP). Dengan modularitas, struktur dari sebuah program akan terkait dengan aspek-aspek dalam masalah yang akan diselesaikan melalui program tersebut. Cara seperti ini akan menyederhanakan pemetaan dari masalah ke sebuah program ataupun sebaliknya.

2. Objek membungkus data dan fungsi bersama menjadi suatu unit dalam sebuah program komputer, objek merupakan dasar dari modularitas dan struktur dalam sebuah program komputer berorientasi objek.

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

4. Enkapsulasi Memastikan pengguna sebuah objek tidak dapat mengganti keadaan dalam dari sebuah objek dengan cara yang tidak layak, hanya metode dalam objek tersebut yang diberi izin untuk mengakses keadaannya. Setiap objek mengakses interface yang menyebutkan bagaimana objek lainnya dapat berinteraksi dengannya. Objek lainnya tidak akan mengetahui dan tergantung kepada representasi dalam objek tersebut.

5. Polimorfisme melalui pengiriman pesan. Tidak bergantung kepada pemanggilan subrutin, bahasa orientasi objek dapat mengirim pesan, metode tertentu yang berhubungan dengan sebuah pengiriman pesan

tergantung kepada objek tertentu di mana pesa tersebut dikirim. Contohnya, bila sebuah burung menerima pesan "gerak cepat", dia akan menggerakan sayapnya dan terbang. Bila seekor singa menerima pesan yang sama, dia akan menggerakkan kakinya dan berlari. Keduanya menjawab sebuah pesan yang sama, namun yang sesuai dengan kemampuan hewan tersebut. Ini disebut polimorfisme karena sebuah variabel tungal dalam program dapat memegang berbagai jenis objek yang berbeda selagi program berjalan, dan teks program yang sama dapat memanggil beberapa metode yang berbeda di saat yang berbeda dalam pemanggilan yang sama. Hal ini berlawanan dengan bahasa fungsional yang mencapai polimorfisme melalui penggunaan fungsi kelas-pertama.

Dengan menggunakan OOP maka dalam melakukan pemecahan suatu masalah kita tidak melihat bagaimana cara menyelesaikan suatu masalah tersebut (terstruktur) tetapi objek-objek apa yang dapat melakukan pemecahan masalah tersebut. Sebagai contoh anggap kita memiliki sebuah departemen yang memiliki manager, sekretaris, petugas administrasi data dan lainnya. Misal manager tersebut ingin memperoleh data dari bag administrasi maka manager tersebut tidak harus mengambilnya langsung tetapi dapat menyuruh petugas bag administrasi untuk mengambilnya. Pada kasus tersebut seorang manager tidak harus mengetahui bagaimana cara mengambil data tersebut tetapi manager bisa mendapatkan data tersebut melalui objek petugas adminiistrasi. Jadi untuk menyelesaikan suatu masalah dengan kolaborasi antar objek-objek yang ada karena setiap objek memiliki deskripsi tugasnya sendiri.

2.4.2 JAVA

Java adalah bahasa pemrograman yang disusun oleh James Gosling yang dibantu oleh rekan-rekannya seperti Patrick Naugton, Chris Warth, Ed Frank, dan Mike Sheridan di suatu perusahaan perangkat lunak yang bernama Susn Microsystems, pada tahun 1991 (Raharjo.B, Heryanto. I, Arif Haryano, 2009, hlm. 1-2). Bahasa pemrograman ini mula-mula diinisialisasi dengan nama “Oak”, namun oada tahun 1995 diganti namanya menjadi “Java”.Bahasa ini banyak

mengadopsi sintaksis yang terdapat pada C dan C++ namun dengan sintaksis model objek yang lebih sederhana serta dukungan rutin-rutin aras bawah yang minimal. Aplikasi-aplikasi berbasis java umumnya dikompilasi ke dalam p-code (bytecode) dan dapat dijalankan pada berbagai Mesin Virtual Java (JVM). Java merupakan bahasa pemrograman yang bersifat umum/non-spesifik (general purpose), dan secara khusus didisain untuk memanfaatkan dependensi implementasi seminimal mungkin. Karena fungsionalitasnya yang memungkinkan aplikasi java mampu berjalan di beberapa platform sistem operasi yang berbeda, java dikenal pula dengan selogannya, "Tulis sekali, jalankan di mana pun". Saat ini java merupakan bahasa pemrograman yang paling populer digunakan, dan secara luas dimanfaatkan dalam pengembangan berbagai jenis perangkat lunak aplikasi ataupun aplikasi berbasis web.

Sun Myrosystems telah mendefinisikan tiga buah edisi dari Java 2 yaitu sebagai berikut :

1. **Java 2 Standard Edition (J2SE),** yang digunakan untuk mengembangkan aplikasi-aplikasi *desktop* dan *applet* (aplikasi untuk Java yang dapat dijalankan di dalam *browser web*).
2. **Java 2 Enterprise Edition (J2EE),** merupakan *superset* dari J2SE yang memperbolehkan kita untuk mengembangkan aplikasi-aplikasi berskala besar(*enterprise*), yaitu dengan melakukan pembuatan aplikasi-aplikasi di sisi *server* dengan menggunakan EJBs (*Enterprise JavaBeans*), aplikasi *web* dengan menggunakan *Servlet* dan JSP (*JavaServer Pages*) dan teknologi lainnya seperti CORBA (*Common Object Request Broker Architecture*) dan XML (*Extensible Markup Language).*
3. **Java 2 Micro Edition (J2ME),** merupakan subset dari J2SE yang digunakan untuk menangani pemrograman di dalam perangkat-perangkat kecil, yang tidak memungkinkan untuk mendukung implementasi J2SE secara penuh seperti pada teknologi *mobile.*

2.4.3 UML (*Unified Modeling Language*)

UML (*Unified Modeling Language*) adalah bahasa spesifikasi standar untuk mendokumentasikan, menspesifikasikan, dan membangun system (Flowler, 2006). *Unified Modeling Language* (UML) adalah himpunan struktur dan teknik untuk pemodelan desain program berorientasi objek (OOP) serta aplikasinya.

UML adalah metodologi untuk mengembangkan sistem OOP dan sekelompok perangkat tool untuk mendukung pengembangan sistem tersebut. UML mulai diperkenalkan oleh Object Management Group, sebuah organisasi yang telah mengembangkan model, teknologi, dan standar OOP sejak tahun 1980-an. Sekarang UML sudah mulai banyak digunakan oleh para praktisi OOP [5]. Logo UML dapat dilihat pada gambar 2.1.



Gambar 2.1 Logo UML

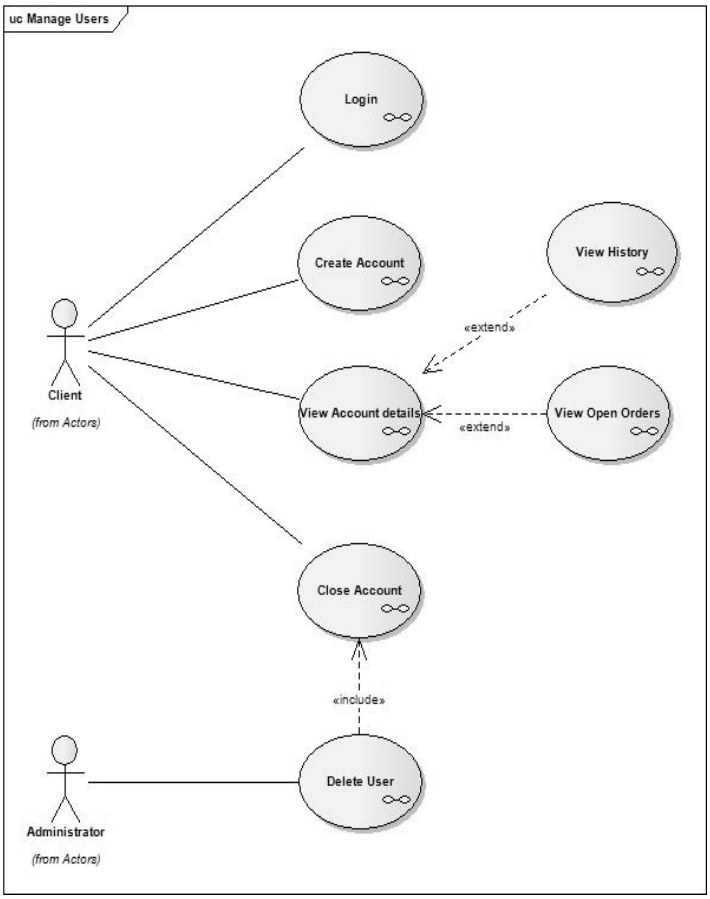
UML suatu bahasa yang digunakan untuk menentukan, memvisualisasikan, membangun, dan mendokumentasikan suatu sistem informasi. UML dikembangkan sebagai suatu alat untuk analisis dan desain berorientasi objek oleh Grady Booch, Jim Rumbaugh, dan Ivar Jacobson. Namun demikian UML dapat digunakan untuk memahami dan mendokumentasikan setiap sistem informasi. Penggunaan UML dalam industri terus meningkat, ini merupakan standar terbuka yang menjadikannya sebagai bahasa pemodelan yang umum dalam industri peranti lunak dan pengembangan sistem.

Bagian-bagian utama dari UML adalah *view*, *diagram*, *model element*, dan *general mechanism* [7]. *Diagram* berbentuk grafik yang menunjukkan simbol elemen model yang disusun untuk mengilustrasikan bagian atau aspek tertentu

dari sistem. Sebuah diagram merupakan bagian dari suatu view tertentu dan ketika digambarkan biasanya

2.4.3.1 *Usecase Diagram*

Usecase adalah abstraksi dari interaksi antara system dan actor. *Usecase* bekerja dengan cara mendeskripsikan tipe interaksi antara user sebuah system dengan sistemnya sendiri melalui sebuah cerita bagaimana sebuah system dipakai. *Usecase* merupakan konstruksi untuk mendeskripsikan bagaimana *system* akan terlihat di mata user. Sedangkan *Usecase* diagram memfasilitasi komunikasi diantara analis dan pengguna serta antara analis dan client. *Usecase* dapat dilihat pada Gambar 2.2

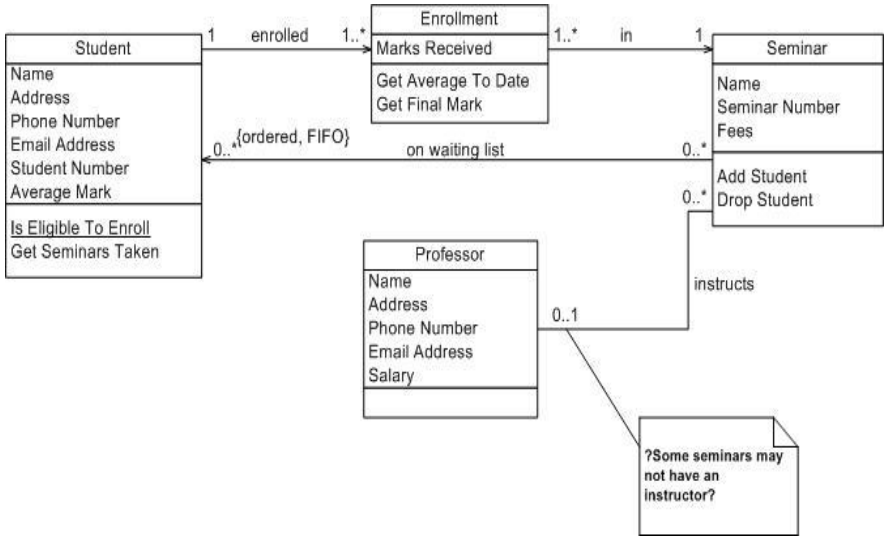


Gambar 2.2 *Usecase*

2.4.3.2 *Class Diagram*

Class adalah dekripsi kelompok obyek-obyek dengan *property*, perilaku (operasi) dan relasi yang sama. Sehingga dengan adanya *class diagram* dapat

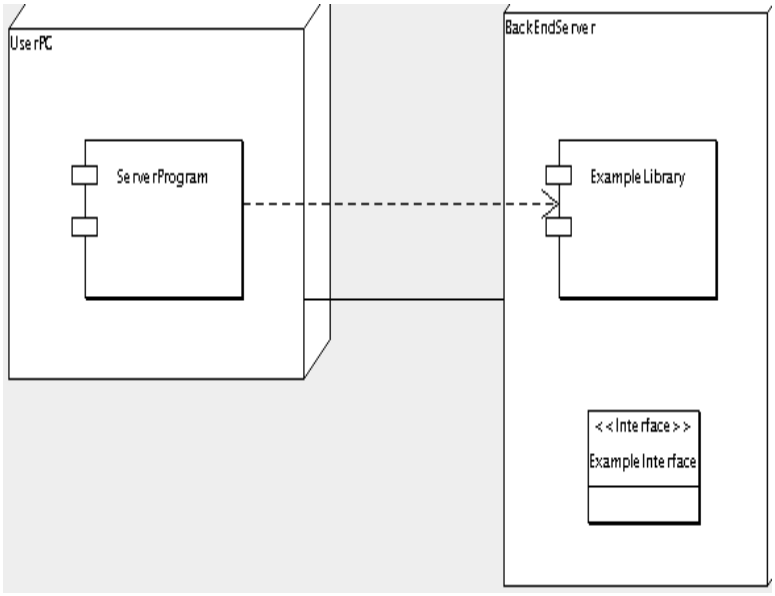
memberikan pandangan global atas sebuah *system*. Hal tersebut tercermin dari *class- class* yang ada dan relasinya satu dengan yang lainnya. Sebuah sistem biasanya mempunyai beberapa *class diagram*. *Class diagram* sangat membantu dalam visualisasi struktur kelas dari suatu system. *Class diagram* dapat dilihat pada Gambar 2.3



Gambar 2.3 *Class Diagram*

2.4.3.3 *Component Diagram*

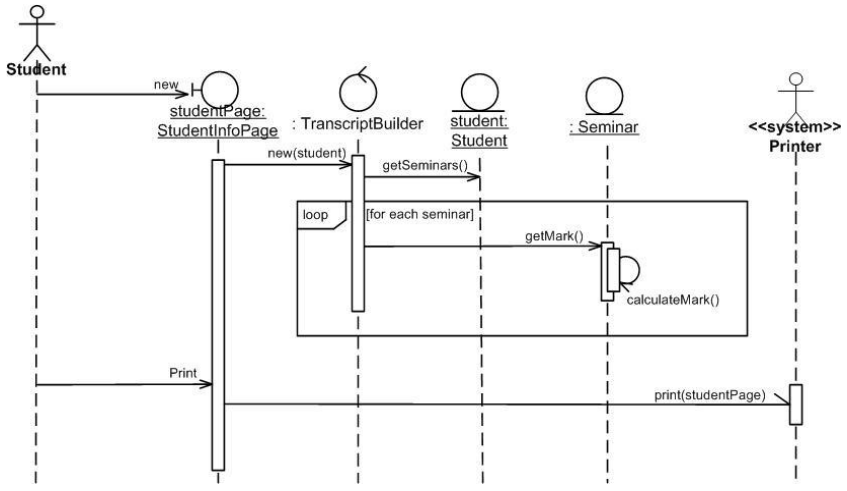
Component software merupakan bagian fisik dari sebuah system, karena menetap di komputer tidak berada di benak para analis. Komponent merupakan implementasi software dari sebuah atau lebih class. Komponent dapat berupa source code, komponent biner, atau executable component. Sebuah komponent berisi informasi tentang logic class atau class yang diimplementasikan sehingga membuat pemetaan dari logical view ke component view. Sehingga component diagram merepresentasikan dunia riil yaitu component software yang mengandung component, interface dan relationship.Component software dapat dilihat pada Gambar 2.4



Gambar 2.4 *Component Diagram*

2.4.3.4 *Sequence Diagram*

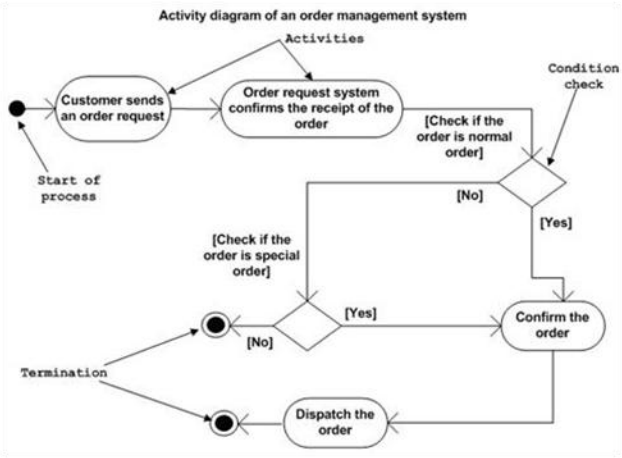
Sequence Diagram digunakan untuk menggambarkan perilaku pada sebuah scenario. Kegunaannya untuk menunjukkan rangkaian pesan yang dikirim antara object juga interaksi antara object, sesuatu yang terjadi pada titik tertentu dalam eksekusi sistem. Sequence Diagram diagram dapat dilihat pada Gambar 2.5



Gambar 2.5 *Sequence Diagram*

2.4.3.5 *Activity Diagram*

Activity Diagram menggambarkan rangkaian aliran dari aktivitas, digunakan untuk mendeskripsikan aktifitas yang dibentuk dalam suatu operasi sehingga dapat juga digunakan untuk aktifitas lainnya seperti *Usecase* atau interaksi, sebagai contoh dapat dilihat pada gambar 2.6



Gambar 2.6 *Activity Diagram*

2.4.4 Greenfoot

Greenfoot merupakan sebuah IDE berbasis java yang dikhususkan untuk pembuatan *game* sederhana. Greenfoot diprakarsai oleh Michael Kolliung pada tahun 2003. Prototype yang pertama dibangun oleh Poul Henriksson dan Michael Kolling pada tahun 2003/2004. semenjak tahun 2005 pembuatan greenfoot ini dilanjutkan dengan melibatkan anggota kelompok dari University of Kent dan Deakin University.

Pada awalnya, greenfoot bertujuan untuk menarik minat anak-anak untuk belajar pemrograman, greenfoot ini cocok untuk anak usia 13 tahun ke atas. Cara penggunaannya pun cukup mudah. Selain berbentuk teks seperti editor lainnya, pada greenfoot ini terdapat perangkat alat bantu seperti class browser, editor dan compiler. Greenfoot juga mendukung bahasa Java secara utuh. Dengan adanya bantuan perangkat-perangkat tadi, konsep OOP pada greenfoot lebih mudah dimengerti.

Didalam greenfoot terdapat project yang dinamakan scenario. Di dalam project tersebut, kita bisa membuat dua macam tipe class dari library greenfoot. Yaitu class world dan actor. World merupakan latar dari *game* yang akan dibuat. Untuk membuat class world, caranya adalah dengan meng-klik new sub class di bagian world. Maka akan mucul sebuah kotak dialog. Kemudian kita diminta untuk memberi nama untuk kelas world tersebut. Setelah itu kita dapat memilih gambar sebagai latar belakang class world yang akan kita buat. Begitu pula pada Class actor, membuat actor caranya adalah dengan meng-klik icon aktor, kemudian pilih new subclass, logo greenfoot dapat dilihat pada gambar 2.7.



Gambar 2.7 Logo Greenfoot

2.4.5 Corel Draw

Corel Draw merupakan salah satu aplikasi pengolah gambar berbasis *vector* yang banyak dipakai oleh pengguna PC. Karena berbagai kemudahan dan keunggulan yang dimiliki oleh corel draw, maka *corel draw* sering dimanfaatkan untuk *desktop publishing*, percetakan, dan bidang lain yang memerlukan pemrosesan visual.

Keunggulan mengolah gambar berbasis *vector* adalah ukuran hasil akhir yang dapat ditekan seminimal mungkin namun dengan kualitas yang tidak kalah dengan gambar berbasis *raster* atau *bitmap*.

Sebenarnya ada banyak sekali *software* desain grafis dan pengolah gambar berbasis *vector* yang bisa kita gunakan,misalnya saja diantaranya adalah corel draw, *photoshop*, *freehand*, *adobe ilustrator* dll. Namun corel draw merupakan program paling populer yang sering dimanfaatkan. Selain karena tampilannya yang *user friendly* dan mudah dipelajari, corel draw juga mempunyai berbagai keunggulan - keunggulan lain yang sangat bermanfaat.