BAB 2
LANDASAN TEORI

2.1 *Game*

Game merupakan suatu bentuk hiburan yang seringkali dijadikan sebagai penyegar pikiran dari rasa penat yang disebabkan oleh aktivitas dan rutinitas kita. *Game* dapat dimainkan berbagai umur tua maupun muda. *Game* juga sudah berkembang dengan pesat sesuai dengan perkembangan teknologi, dari *game* sederhana sampai *game* modern saat ini. Hal ini terbukti dengan adanya perkembangan jenis, produk, serta alat yang digunakan. Selain itu *game* juga bisa dimainkan dari satu orang hingga beberapa orang sekaligus.

2.1.1 Pengertian *Game*

Game atau sering disebut juga *video game* adalah permainan yang menggunakan interaksi antarmuka pengguna melalui gambar yang dihasilkan oleh piranti *video*. *Game* umumnya menyediakan sistem penghargaan, misalnya *score* yang dihitung berdasarkan tingkat keberhasilan yang dicapai dalam menyelesaikan tugas-tugas yang ada di dalam *game* tersebut. Sistem elektronik yang digunakan untuk menjalankan *game* disebut *platform*, seperti komputer pribadi (*personal computer*) ataupun konsol permainan (*game console*) [2]. *Game* merupakan sebuah sistem di mana permainan terlibat dalam konflik buatan, didefinisikan oleh aturan, yang menghasilkan hasil yang terukur [10].

Tapi pertama-tama, kita pertimbangkan apakah *game* itu bukan : banyak fitur keren, banyak grafis mewah, serangkaian teka-teki menantang, suasana yang menarik dan cerita [7].

2.1.2 *Genre Game*

Game memiliki banyak tipe atau *genre.* Masing-masing genre memiliki karakteristik berbeda. *Genre game* diantaranya yaitu[2]:

1. *Action Game*

   Game ini dikenali dari aksi didalamnya yang berupa pergerakan, *attack*, reaksi dan pergerakan-pergerakan lainnya.
2. *Adventure GameGame* tipe ini menuntut *player* untuk menyelesaikan misi yang diberikan sepanjang petualangan di dalam *game.*

3. *Casual Game*

   Game tipe ini memiliki aturan yang mudah dimengerti dan tidak membutuhkan keahlian khusus untuk memainkannya

4. *Educational Game*

   Game edukasi memberikan penekanan pada unsur pendidikan dan pembelajaran

5. *Role-Playing Game*

   Pada *game* bertipe ini memiliki dunia yang luas untuk dieksplorasi oleh *player*, baik untuk mencari harta karun, objek dan status (*leveling*), sambil menghancurkan *monster* dan objek yang menghalaginya.

6. *Strategy Game*

   Dalam *game* tipe ini dibutuhkan pemikiran dan perencanaan yang benar untuk dapat memenangkan permainan dan biasanya menggunakan sudut pandang *overhead* sehingga *player* dapat melihat seluruh area permainan.

7. *Simulation Game*

   Simulation game atau *sims* adalah suatu bentuk permainan yang menggambarkan situasi dunia nyata, baik dalam rupa simulasi *driving* maupun dalam hubungannya dengan lingkungan interaksi antara manusia.

8. *Puzzle Game*

   Puzzle game memiliki tujuan yang sangat sederhana untuk menyelesaikan *puzzle* yang diberikan.

2.1.3 *Casual Game*

Casual game adalah *game* yang dikembangkan untuk konsumen massal, bahkan bagi orang yang memainkannnya tidak dianggap sebagai seorang *gamer*. *Casual game* mudah untuk diakses, mudah dipelajari, dan tidak memerlukan keterampilan khusus dari *game* sebelumnya, keahlian, atau waktu yang teratur untuk bermain. Hal ini menyenangkan untuk kesibukan orang dewasa yang memiliki sedikit waktu untuk bermain[1]. Yang termasuk dalam *casual game* adalah *board game*, *card game* dan sebagainya. *Game* ini bertujuan untuk

menghibur *player*, dan selain itu dapat juga digunakan untuk melatih, mempelajari dan meningkatkan pengetahuan dan *level skill gamer*[7].

Pada *casual game* terdapat empat elemen kunci yaitu[16]*:*

1. aturan dan tujuan harus jelas.
2. *player* harus dapat dengan cepat mahir bermain *game*.
3. *gameplay casual* bisa menyesuaikan waktu bermain dengan kehidupan pemain dan jadwalnya.
4. konsep permainan meminjam konten yang familiar dan tema dari kehidupan[16].



Gambar 2-1 Contoh *Casual Game* [18]

Casual game memiliki serangkaian mekanisme permainan yang terbagi beberapa jenis yang beragam. Mekanisme ini mencakup *gameplay* dan *subgenre* dari *casual game*. Mekanisme *casual game* diantara lain [1]:

1. *Matching*

   Matching game atau permainan mencocokan telah lama mendominasi *casual game*. Hal tersebut dikarenakan mekanisme permainannya sangat fleksibel dan dengan cepat *player* mendapatkan *feedback* yang biasanya berupa *score*. Mekanisme *matching game* yaitu *player* diminta untuk mencocokan objek yang sudah disediakan sesuai dengan pola yang diinginkan oleh *game* bisa berupa warna, gambar, simbol dan lain-lain.

2. *Sorting*

   Sorting atau pengurutan mempunyai sejumlah cara yang berbeda. pengurutan simbol-simbol abstrak yang relevan untuk permainan, untuk angka dan huruf. Beberapa *game* bahkan mendorong penggunaan pengetahuan kontekstual untuk mengurutkan pakaian ke dalam kategori gaya yang berbeda. *Sorting* membuat *gameplay casual* alami karena player tidak perlu untuk mengembangkan keterampilan baru.

3. *Seeking*

   Seeking atau sering disebut juga *seek and find* merupakan mekanisme permainan dimana *player* diminta untuk menemukan benda tersembunyi di area permainan. Dalam bentuk yang paling sederhana yang permainan memberikan gambaran dengan benda tersembunyi di suatu tempat, sambil memberikan *player* dengan daftar objek yang *player* harus temukan.

4. *Managing*

   Managing merupakan mekanisme permainan yang meminta *player* mengelola sebuah tempat. Pengelolaan bisa berupa manajemen waktu, sumber daya dan sebagainya. Mekanisme permainan ini menuntut *player* memantau beberapa elemen yang ada pada *game* dan menjaganya tetap pada *track.*

5. *Hitting*

   Hitting merupakan mekanisme permainan yang meminta *player* menjangkau dan bersentuhan dengan benda lain. mekanisme ini menggunakan banyak bentuk, *player* bisa menjangkau objek dengan tangan, dengan tongkat, dengan raket, dan lain-lain.

6. *Chaining*

   Chaining adalah rantai spesifik tindakan yang dapat dilakukan dalam sebuah permainan. *Chaining* merupakan cara untuk membentuk tindakan *player*. Ini adalah cara bagi seorang desainer *game* memberikan pilihan lain untuk melakukan sesuatu dalam permainan bahkan jika unsur lain dari permainan menceritakan pemain sebaliknya.

7. *Constructing*

Constructing atau konstruksi merupakan mekanisme permainan dimana pemain mengontruksi sebuah objek disusun secara rapih. Mekanisme kontruksi memungkinkan pemain untuk menjadi kreatif. Terutama dalam *sandbox game* di mana pemain dapat bebas bereksperimen. *Game* berbasis konstruksi sering memungkinkan beberapa opsi untuk mencapai tujuan, terkadang hal ini bisa saja tidak terstruktur.

8. *Physics*

   Physics atau mekanisme permainan fisika merupakan simulasi hukum dasar fisika, meniru hal-hal seperti gravitasi, gesekan dan gaya. Hukum fisika menjadi aturan yang mempengaruhi unsur-unsur dalam permainan dan berdampak bergerak. *Player* yang memiliki Pengalaman dengan hukum-hukum fisika alam, ini membuatnya lebih mudah untuk mengambil konsep dasar permainan. Meskipun kita memahami dasar hukum fisika tidak berarti kita bisa memprediksi bagaimana semuanya akan berjalan dengan tepat, ini diterjemahkan menjadi *game* dimana *player* bereksperimen.

9. *Leveling*

   Leveling merupakan mekanisme yang ditemukan dalam *game* untuk memberikan pemain rasa peningkatan. Seringkali tingkat terkait dengan membuka konten permainan yang berbeda.

10. *MultiPlayer*

    MultiPlayer game memungkinkan beberapa pemain bermain *game* bersama-sama, baik itu berdasarkan giliran seperti pada papan permainan, atau *real time*.[1]

2.2 *Little Farm*

Game little farm merupakan ber*genre casual game*. *Game little farm* ini dikembangkan pada tahun 2008 oleh MumboJumbo LLC™. MumboJumbo LLC™ merupakan pengembang dan pemasar *premium casual game* masal untuk PC dan konsol *game*. *Game* ini bisa di*download* melalui situs resmi MumboJumbo LLC™ atau bisa juga dimainkan secara *online* disitus resminya[18].

Game little farm ini bisa dimainkan dengan spesifikasi seperti dibawah ini:

Tabel 2-1 Spesifikasi Minimum *Game Little Farm*[18]

Minimum System Requirements
Processor Speed: 600MHz
256MB RAM
167 free HD *space*
Windows 2K/XP
Sound card
64 MB DirectX 8.1 *compatible graphics card or better*

2.2.1 Cara bermain *Game Little Farm*

Tujuan *game little farm* adalah untuk memuat truk dengan produk untuk mengirim ke pasar. Berikut ini cara bermain game *little farm* :

1. Mengidentifikasi kotak pencocokan tanaman di lahan yang dapat digunakan sebagai empat sudut persegi atau persegi panjang.
2. Klik kiri pada salah satu kotak sudut dan tarik kursor ke sudut yang berlawanan untuk menyorot persegi yang lebih besar dan menunjuk area penyiraman.
3. Lanjutkan untuk membuat *matching* dan menanam buah-buahan dan sayuran. Sebagai produk yang dipanen, maka secara otomatis akan ditempatkan di truk.
4. Setelah menghasilkan cukup stok, truk akan pergi ke pasar dan menghitung skor.
5. Setiap *level* memiliki *timer* disisi kiri layar untuk menunjukkan berapa banyak waktu yang pemain miliki untuk mencapai tujuan di setiap *level*. Jika pemain kehabisan waktu, maka pemain harus mengulang *level* tersebut.
6. Pada setiap *level,* pemain dapat membeli barang baru untuk peternakan pemain di toko umum dengan uang yang anda peroleh pada akhir setiap hari.
7. Pemain juga dapat memperoleh poin tambahan saat bermain dengan mencari *best match* di lahan setiap membuat persegi terbesar yang dapat disesuaikan pada satu waktu.[18]

2.2.2 *Scoring*

Dapatkan skor untuk setiap muatan truk yang pemain kirim ke pasar. Anda dapat memperoleh poin tambahan dengan mencari *best match,* dan *bonus timer* pada akhir setiap putaran.

Skor sesorang pemain dan kemajuan melalui permainan disimpan dalam profil. Ketika *game* dijalankan untuk pertama kalinya, pemain akan diminta untuk memasukkan nama mereka untuk membuat profil. Profil aktif akan ditampilkan di bagian atas layar *main menu*. Profil dapat ditambahkandan dihapus.[18]

2.3 *Artificial Intelligence*

Artificial intelligence atau kecerdasan buatan merupakan salah satu bagian ilmu komputer yang membuat agar mesin (komputer) dapat melakukan pekerjaan seperti dan sebaik yang dilakukan oleh manusia. Lebih detailnya pengertian kecerdasan buatan dapat dipandang sebagai berikut:

1. Sudut pandang kecerdasan

   Kecerdasan buatan akan membuat mesin menjadi cerdas (mampu berbuat seperti yang dilakukan manusia).
2. Sudut pandang penelitian

   Kecerdasan buatan adalah suatu studi bagaimana membuat agar komputer dapat melakukan sesuatu sebaik yang dikerjakan oleh manusia.
3. Bisnis

   Kecerdasan buatan adalah kumpulan peralatan yang sangat powerfull dan metodologis dalam menyelesaikan masalah-masalah bisnis.
4. Pemrograman

   Studi tentang pemrograman simbolik, penyelesaian masalah dan pencarian. Untuk melakukan aplikasi kecerdasan buatan ada dua bagian utama yang sangat dibutuhkan

   a. Basis Pengetahuan

      Fakta-fakta, teori, pemikiran dan hubungan antara satu dengan yang lainnya.

   b. Motor Inferensi

      Kemampuan menarik kesimpulan berdasarkan pengalaman [5].

2.4 *Forward chaining*

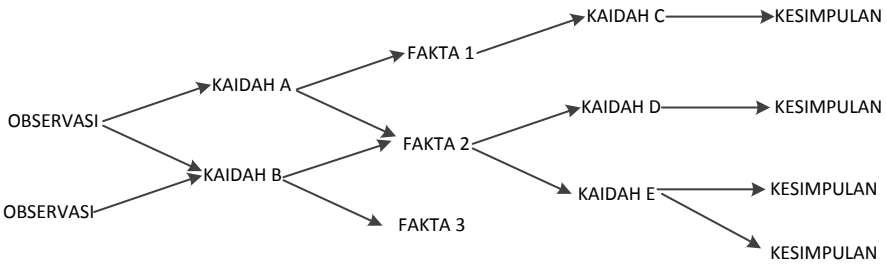
Forward chaining merupakan perunutan yang dimulai dengan menampilkan kumpulan data atau fakta yang menyakinkan menuju konklusi akhir. *Forward chaining* dimulai dari premis – premis atau informasi masukan (IF) dahulu kemudian menuju konklusi atau *derived information* (THEN) atau dapat dimodelkan sebagai berikut [4]:

IF (informasi masukan)

THEN (konklusi)

Informasi masukan dapat berupa data, bukti, temuan atau pengamatan. Sedangkan konklusi dapat berupa tujuan, hipotesa, penjelasan atau diagnosa. Sehingga jalannya *forward chaining* maju dapat dimulai dari data menuju tujuan dan bukti menuju hipotesa, dari temuan menuju penjelasan, atau dari pengamatan menuju diagnosa[4].

Strategi dari sistem ini adalah dimulai dari inputan beberapa fakta, kemudian menurunkan beberapa fakta dari aturan-aturan yang cocok pada *knowledge base* dan melanjutkan prosesnya sampai jawaban sesuai. *Forward chaining* dapat dikatakan sebagai penelusuran deduktif[2].



Gambar 2-2 Diagram *Forward Chaining*[4]

Forward chaining merupakan suatu rantai yang dicari atau dilewati/ dilintasi dari suatu permasalahan untuk memperoleh solusinya disebut dengan *forward chaining*. Cara lain menggambarkan *forward chaining* ini adalah dengan penalaran dari fakta menuju konklusi yang terdapat dari fakta [9].

Dalam implementasinya, *forward chaining* sangat membantu *developer* aplikasi dalam membangun sebuah sistem. Karena dengan penggunaan metode ini jika developer ingin menambah beberapa kondisi dan aturan, *developer* tidak

perlu membongkar lagi kode program dari awal. Berikut adalah karakteristik *forward chaining*:

Tabel 2-2 Karakteristik *Forward chaining* [15]

Forward chaining
Perencanaan, *Monitoring, Control.* Disajikan untuk masa depan. Data memandu, penalaran dari bawah ke atas. Bekerja kedepan untuk mendapatkan solusi apa yang mengikuti fakta. *Breadth first search* dimudahkan

Pada Metode *forward chaining* juga di artikan sebagai pendekatan yang dimotori data. Dalam pendekatan ini pelacakan dimulai dari informasi masukan, dan selanjutnya mencoba menggambarkan kesimpulan. Sehingga metode ini juga sering disebut "*Data driven*".

Operasi dari sistem *forward chaining* dimulai dengan memasukkan sekumpulan fakta yang diketahui ke dalam memori kerja (*working memory*), kemudian menurunkan fakta baru berdasarkan aturan yang premisnya cocok dengan fakta yang diketahui. Proses ini dilanjutkan sampai dengan mencapai goal atau tidak ada lagi aturan yang premisnya cocok dengan fakta yang diketahui. Operasi tersebut dapat digambarkan seperti gambar berikut:

Langkah-langkah yang harus dilakukan dalam membuat sistem *forward chaining* berbasis aturan, yaitu:

1. Pendefinisian Masalah.

   Tahap ini meliputi pemilihan domain masalah dan akuisisi pengetahuan.
2. Pendefinisian Data Input.

   Sistem *forward chaining* memerlukan data awal untuk memulai inferensi.
3. Pendefinisian Struktur Pengendalian Data.

   Aplikasi yang kompleks memerlukan premis tambahan untuk membantu mengendalikan[15].

2.5 Pemrograman Berorientasi Objek

Metodologi berorientasi objek adalah suatu strategi pembangunan perangkat lunak yang mengorganisasikan perangkat lunak sebagai kumpulan objek yang berisi data dan operasi yang berisi data dan operasi yang diberlakukan terhadapnya. Metodologi berorientasi objek merupakan suatu cara bagaimana sistem perangkat lunak dibangun melalui pendekatan objek secara sistematis. Metode berorientasi objek didasarkan pada penerapan prinsip-prinsip pengelolaan kompleksitas. Metode berorientasi objek meliputi rangkaian aktivitas analisis berorientasi objek, perancangan berorientasi objek, pemrograman berorientasi objek dan pengujian berorientasi objek[11].

Pendekatan berorientasi objek merupakan suatu teknik atau cara pendekatan dalam melihat permasalahan dan sistem (sistem perangkat lunak, sistem informasi, atau sistem lainnya). Pendekatan berorientasi objek akan memandang sistem yang akan dikembangkan sebagai suatu kumpulan objek yang berkorespondensi dengan objek-objek dunia nyata.

Sistem berorientasi objek merupakan sebuah sistem yang dibangun dengan berdasarkan metode berorientasi objek adalah sebuah sistem yang komponennya dibungkus menjadi kelompok data dan fungsi. Setiap komponen dalam sistem tersebut dapat mewarisi atribut, sifat dan komponen lainnya yang dapat berinteraksi satu sama lain[11].

Berikut ini adalah beberapa konsep dasar yang harus dipahami tentang metodologi berorientasi objek[11]:

1. Kelas

   Kelas adalah kumpulan objek-objek dengan karakteristik yang sama. Kelas merupakan definisi statik dan himpunan objek yang sama yang mungkin lahir atau diciptakan dengan kelas tersebut. Sebuah kelas akan mempunyai sifat (atribut), kelakuan (operasi/metode), hubungan (*relationship*) dan arti. Suatu kelas dapat diturunkan pada kelas yang lain, dimana atribut dan kelas semula dapat diwariskan ke kelas yang baru.

2. Objek (*Object*)

Objek adalah abstraksi dan sesuatu yang mewakili dunia nyata seperti benda, manusia, satuann organisasi, tempat, kejadian, struktur, status, atau hal-hal lain yang bersifat abstrak. Objek merupakan suatu entitas yang mampu menyimpan informasi (status) dan mempunyai operasi (kelakuan) yang dapat diterapkan atau dapat berpengaruh pada status objeknya. Objek mempunyai siklus hidup yaitu diciptakan, dimanipulasi dan dihancurkan.

3. Metode (*method*)

   Metode atau *method* pada sebuah pada sebuah kelas hampir sama dengan fungsi atau prosedur pada metode struktural. Sebuah kelas boleh memiliki lebih dari satu metode atau operasi. Metode atau operasi yang berfungsi untuk memanipulasi objek itu sendiri. Metode merupakan fungsi atau transformasi yang dapat dilakukan terhadap objek atau dilakukan oleh objek. Metode dapat berasal dari *event,* aktivitas atau aksi keadaan, fungsi, atau kelakuan dunia nyata. Contoh metode atau operasi misalnya Read, Write, Move, dan sebagainya. Kelas sebaiknya memiliki metode get dan set untuk setiap atribut agar konsep enkapsulasi tetap terjaga. Metode get digunakan untuk memberikan akses kelas lain dalam mengakses atribut, dan set adalah metode yang digunakan untuk mengisi atribut, agar kelas lain tidak mengakses atribut secara langsung

4. Atribut (*attribute*)

   Atribut dari sebuah kelas adalah variabel global yang dimiliki sebuah kelas. Atribut dapat berupa nilai atau elemen-elemen data yang dimiliki oleh objek dalam kelas objek. Atribut dimiliki secara individual oleh sebuah objek, misalnya berat, jenis, nama, dan sebagainya. Atribut sebaiknya bersifat privat untuk menjaga enkapsulasi

5. Abstraksi (*abstraction*)

   Prinsip untuk merepresentasikan dunia nyata yang kompleks menjadi satu bentuk model yang sederhana dengan mengabaikan aspek-aspek lain yang tidak sesuai dengan permasalahan.

6. Enkapsulasi (*encapsulation*)

Pembungkusan atribut data dan layanan (operasi-operasi) yang dimiliki objek untuk menyembunyikan implementasi dan objek sehingga objek lain tidak mengetahui cara kerjanya.

7. Pewarisan (*Inheritance*)

   Mekanisme yang memungkinkan satu objek mewarisi sebagian atau seluruh definisi dan objek lain sebagai bagian dirinya.

8. Antarmuka (*Interface*)

   Antarmuka atau *interface* sangat mirip dengan kelas, tapi tanpa atribut kelas dan memiliki metode yang dideklarasikan tanpa isi. Deklarasi metode pada sebuah *interface* dapat diimplementasikan oleh kelas lain. Sebuah kelas dapat mengimplementasikan lebih dari satu antarmuka dimana kelas ini akan mendeklarasikan metode pada antarmuka yang dibutuhkan oleh kelas itu sekaligus mendefinisikan isinya pada kode program kelas itu. Metode pada antarmuka yang diimplementasikan pada suatu kelas harus sama persis dengan yang ada pada antarmuka. Antarmuka atau interface biasanya digunakan agar kelas yang lain tidak mengakses langsung ke suatu kelas, mengakses antarmukanya.

9. *Reusability*

   Pemanfaatan kembali objek yang sudah didefinisikan untuk suatu permasalahan pada permasalahan lainnya yang melibatkan objek tersebut.

10. Generalisasi dan Spesialisasi

    Menunjukan hubungan antara kelas dan objek yang umum dengan kelas dan objek yang khusus. Misalnya kelas yang lebih umum (generalisasi) adalah kendaraan darat dan kelas khususnya (spesialisasi) adalah mobil, motor, dan kereta.

11. Komunikasi antar objek

    Komunikasi antar objek dilakukan lewat pesan (*message*) yang dikirim dan satu objek ke objek lainnya.

12. Polimorfisme (*polymorphism*)

    Kemampuan suatu objek untuk digunakan di banyak tujuan yang berbeda dengan nama yang sama sehingga menghemat baris program.

13. *Package*

Package adalah sebuah kontainer atau kemasan yang dapat digunakan untuk mengelompokkan kelas-kelas sehingga memungkinkan beberapa kelas yang bernama sama disimpan dalam *package* yang berbeda.[11]

2.6 *Unified Modeling Language* (UML)

UML adalah penerus gelombang berorientasi objek analisis dan desain (OOA & D) metode yang muncul di akhir ‘80-an dan ’90-an awal. Hal yang paling langsung menyatukan metode Booch, Rumbaugh (OMT), dan Jacobson, tapi jangkauan lebih luas dari itu. UML pergi melalui proses standarisasi dengan OMG (*Object Management Group*) dan sekarang menjadi standar OMG.

UML disebut bahasa pemodelan, bukan metode. Sebagian besar metode terdiri, setidaknya pada prinsipnya, kedua model sebuah bahasa dan proses. Bahasa pemodelan adalah notasi (terutama grafis) bahwa metode digunakan untuk mengekspresikan desain. Proses ini menyarankan mereka apa langkah yang harus diambil dalam melakukan desain.

Bagian proses dalam buku banyak metode yang agak samar. Selain itu, kebanyakan orang, ketika mereka mengatakan bahwa mereka menggunakan metode, menggunakan bahasa pemodelan, tapi jarang mengikuti proses. Jadi dalam banyak hal pemodelan bahasa adalah bagian paling penting dari meode ini. Hal ini tentu bagian penting untuk komunikasi. Jika anda ingin mendiskusikan desain anda dengan seseorang, itu adalah bahasa pemodelan yang anda berdua perlu pahami, bukan proses yang digunakan untuk sampai ke desain tersebut [17].

2.5.1 Diagram 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.

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[11].

Diagram aktivitas juga banyak digunakan untuk mendefinisikan hal-hal berikut [10]:

1. Rancangan proses bisnis di mana setiap urutan aktivitas yang digambarkan merupakan proses bisnis sistem yang didefinisikan.
2. Urutan atau pengelompokan tampilan dari sistem/user interface di mana setiap aktivitas dianggap memiliki sebuah rancangan antarmuka tampilan.
3. Rancangan pengujian di mana setiap aktivitas dianggap memerlukan sebuah pengujian yang perlu didefinisikan kasus ujinya.
4. Rancangan menu yang ditampilkan pada perangkat lunak.

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

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 antar objek. 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 diinstansiasi 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[11].

2.7 Pengujian Perangkat Lunak

Pengujian perangkat lunak adalah elemen kritis dari jaminan kualitas perangkat lunak dan merepresentasikan kajian pokok dari spesifikasi, desain, dan pengkodean[8].

2.6.1 Pengujian *Black box*

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* [8].

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

1. Fungsi-fungsi yang tidak benar atau hilang
2. Kesalahan dalam *interface*
3. Kesalahan dalam struktur data atau akses *database* eksternal
4. Kesalahan kinerja

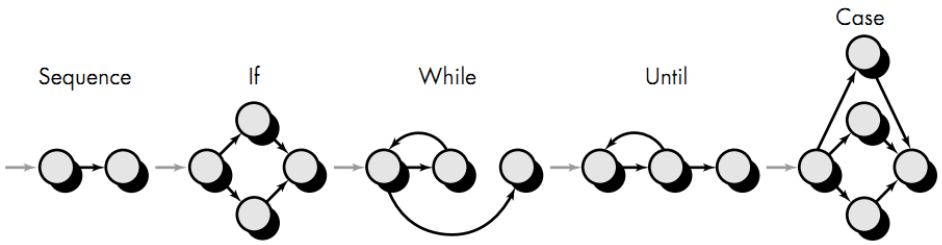
5. Inisialisasi dan kesalahan terminasi

2.6.2 Pengujian *White box*

Pengujian *white box*, yang kadang-kadang disebut pengujian glass box, adalah metode desain *test case* yang menggunakan struktur kontrol desain prosedural untuk memperoleh *test case* [8]. Dengan menggunakan metode pengujian *white box*, perekayasa sistem dapat melakukan *test case* sebagai berikut :

1. Memberikan jaminan bahwa semua jalur independen pada suatu modul telah digunakan paling tidak satu kali
2. Menggunakan semua keputusan logis pada sisi true dan false
3. Mengeksekusi semua *loop* pada batasan mereka dan pada batas operasional mereka
4. Menggunakan struktur data internal untuk menjamin validitasnya.

Pengujian *white box* yang berupa notasi diagram alir.



Gambar 2-3 Notasi Diagram Alir [8]

2.8 Java

Java adalah salah satu bahasa pemrograman komputer yang berorientasi objek, 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 *Hot*Java dengan simbol secangkir kopi panas.

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*.

1. 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*.
2. Java adalah bahsa yang *object-oriented* : Konsep objek diterapkan pada pembuatan program Java, bahkan Java menggunakan konsep objek secara

murni sehingga tidak mungkin seseorang membuat program Java tanpa mendefinisikan kelas dan menggunakan objek.

3. 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.
4. 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*.
5. Java adalah bahasa yang *secure* : Secara otomatis Java menerapkan pengamanan terhadap aplikasi sehingga mengurangi kemungkinan terjadinya serangan dari pengguna jaringan.
6. 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.
7. Java adalah bahasa yang *portable* : Program Java sangat mudah dipindahkan dari satu mesin ke mesin lain tanpa harus dikompilasi ulang.
8. 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.

9. 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.
10. 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 representasi *run-time* sehingga objek yang sedang digunakan oleh suatu aplikasi dapat diperiksa berasal dari kelas yang mana melalui informasi *run-time*.[14]

2.9 Greenfoot

Greenfoot merupakan sebuah IDE berbasis java yang dikhususkan untuk pembuatan game sederhana. Greenfoot diprakarsai oleh Michael Kolling 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 klik *new subclass* 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 klik *icon* aktor, kemudian pilih *new subclass*.

Pada dasarnya, Greenfoot dapat dipandang sebagai kombinasi dari :

1. Sebuah *framework* untuk membuat skema program Java yang dapat divisualisasikan dalam bentuk dua dimensi.
2. Sebuah IDE (*Integrated Development Environment*), termasuk *browser* kelas, *editor* program, *compiler*, antarmuka untuk eksekusi, dan lainlain.

Seperti yang telah diutarakan, *framework* Greenfoot dapat digunakan untuk program-program yang dapat divisualisasikan dalam *grid* dua dimensi.

Framework ini mempunyai dua fungsi utama :

1. Mempermudah representasi objek secara grafis.
2. Menyediakan fasilitas dan antarmuka untuk mengkontrol eksekusi program (*start*, *stop*, *step*).

Framework Greenfoot mempunyai sebuah kelas basis bernama GreenfootObject yang harus menjadi induk dari semua kelas yang ingin "menampakkan dirinya" secara visual. Kelas ini mempunyai dua metode penting yang harus diimplementasikan :

1. Metode getImage() yang harus memberikan nilai balik gambar/image dari sebuah objek; dan
2. Metode act() yang akan dipanggil dalam setiap langkah (*step*) dari proses iterasi (*simulation loop*).

Dalam Greenfoot terdapat beberapa istilah yang dipergunakan, diantaranya skenario atau projek. Skenario merupakan sekumpulan kelas mengimplementasikan kelas-kelas basis untuk aplikasi tertentu. Idenya adalah, supaya pengguna Greenfoot mempunyai sesuatu yang dapat digunakan sebagai pijakan awal untuk bereksperimen, untuk melakukan modifikasi atau untuk melakukan penambahan fungsionalitas, seiring dengan kemajuan pemakai dalam memahami bahasa Java.

Adapula IDE Greenfoot yang terdiri atas beberapa elemen yang biasa ditemui dalam sebuah IDE pada umumnya, misalnya :

1. *Editor* kode sumber
2. *Browser* kelas
3. Fasilitas kompilasi
4. Kontrol eksekusi program
5. *Debugger*

Greenfoot dapat menvisualisasikan objek-objek dari sebuah *scenario* dan menyediakan sebuah bentuk antarmuka bagi pengguna untuk berinteraksi dengan objek-objek tersebut. Selain itu, juga tersedia fasilitas untuk mengontrol eksekusi program dari *scenario* yang sedang berjalan.

Fitur penting dari Greenfoot yanag membedakan dengan IDE yang lain adalah interaksi langsung. Interaksi dapat dilakukan baik untuk kelas maupun untuk objek. Misalnya :

1. Untuk menciptakan sebuah objek, kita bisa memilih secara langsung *item menu* konteks (*context-menu* / *pop-up menu*) pada sebuah kelas untuk membentuk objek baru.
2. Setelah objek diinisiasi, ia dapat ditempatkan pada lingkungan dengan klik pada posisi yang diinginkan.

Interaksi objek juga dapat dilakukan dengan cara yang hampir serupa :

1. Objek yang telah ada bisa dipindahkan setiap saat dengan melakukan *drag* (menekan tombol kiri *mouse*, menggeser dan melepaskan tombol setelah sampai pada posisi yang diinginkan) pada objek tersebut.

Pilih/panggil metode yang diinginkan dari menu konteks, dengan klik kanan objek tersebut. Jika metode yang dipanggil mengubah keadaan (*state*) objek yang divisualisasikan, maka perubahan akan segera ditampilkan [3].