BAB 2

TINJAUAN PUSTAKA

2.1. Seni Pewayangan

Wayang bervariasi dengan kata bayang berarti bayang-bayang atau bayangan, yang memiliki nuansa menerawang, samar-samar, atau remang-remang. Dalam arti harfiah wayang merupakan bayang-bayang yang dihasilkan oleh boneka-boneka wayang di dalam teatrikalnya. Boneka-boneka wayang mendapat cahaya dari lampu minyak (blencong) kemudian menimbulkan bayangan, ditangkaplah bayangan itu pada layar dari balik layar tampaklah bayangan, bayangan ini disebut wayang.

Wayang merupakan bayangan, gambaran atau lukisan mengenai kehidupan alam semesta. Di dalam wayang digambarkan bukan hanya mengenai manusia, namun kehidupan manusia dalam kaitannya dengan manusia lain, alam, dan Tuhan. Alam semesta merupakan satu kesatuan yang serasi, tidak lepas satu dengan yang lain dan senantiasa berhubungan. Unsur yang satu dengan yang lain di dalam alam semesta berusaha keras ke arah keseimbangan. Kalau salah satu goncang maka goncanglah keseluruhan alam sebagai suatu keutuhan (sistem kesejagadan) [15].

2.1.1. Jabang Tetuka

Jabang Tetuka merupakan nama dari Gatotkaca sewaktu masih kecil. Sampai usia satu tahun, tali pusar Tetuka belum bisa dipotong walau menggunakan senjata apa pun. Arjuna (adik Bimasena) pergi bertapa untuk mendapatkan petunjuk dewa demi menolong keponakannya itu. Setelah bertapa Arjuna mendapat jawaban ada satu senjata yang dapat memotong tali pusar Tetuka, senjata itu adalah senjata konta yang ada di tangan Batara Narada.

2.1.2. Gatotkaca

Gatotkaca adalah seorang tokoh dalam wiracarita Mahabharata, putra Bimasena (Bima) atau Wrekodara dari keluarga Pandawa. Ibunya bernama

Hidimbi (Harimbi), berasal dari bangsa raksasa. Gatotkaca dikisahkan memiliki kekuatan luar biasa. Dalam perang besar di Kurukshetra, ia menewaskan banyak sekutu Korawa sebelum akhirnya gugur di tangan Karna.

Di Indonesia, Gatotkaca menjadi tokoh pewayangan yang sangat populer. Misalnya dalam pewayangan Jawa, ia dikenal dengan sebutan Gatutkaca (bahasa Jawa: Gathutkaca). Kesaktiannya dikisahkan luar biasa, antara lain mampu terbang di angkasa tanpa menggunakan sayap, serta terkenal dengan julukan "otot kawat tulang besi".

2.2. *Game*

2.2.1. Pengertian *Game*

Game merupakan kata dalam bahasa inggris yang berarti permainan. Permainan adalah sesuatu yang dapat dimainkan dengan aturan tertentu sehingga ada yang menang dan ada yang kalah, biasanya dalam konteks tidak serius atau dengan tujuan refreshing. Suatu cara belajar yang digunakan dalam menganalisa interaksi antara sejumlah pemain maupun perorangan yang menunjukkan strategi-strategi yang rasional. Teori permainan pertama kali ditemukan oleh sekelompok ahli Matematika pada tahun 1944. Teori itu dikemukakan oleh John von Neumann and Oskar Morgenstern yang berisi [4]:

"*Game* terdiri atas sekumpulan peraturan yang membangun situasi bersaing dari dua sampai beberapa orang atau kelompok dengan memilih strategi yang dibangun untuk memaksimalkan kemenangan sendiri atau pun untuk meminimalkan kemenangan lawan. Peraturan-peraturan menentukan kemungkinan tindakan untuk setiap pemain, sejumlah keterangan diterima setiap pemain sebagai kemajuan bermain, dan sejumlah kemenangan atau kekalahan dalam berbagai situasi. (*Theory of Games and Economic Behavior* (3d ed. 1953)).

Sedangkan Kartono menjelaskan bahwa teori permainan (*Game Theory*) merupakan teori yang menggunakan pendekatan matematis dalam merumuskan situasi persaingan dan konflik antara berbagai kepentingan. Teori ini dikembangkan untk menganalisa proses pengambilan keputusan yaitu strategi

optimum dari situasi-situasi persaingan yang berbeda-beda dan melibatkan dua atau lebih kepentingan.

Secara umum teori permainan dapat didefinisikan sebagai sebuah pendekatan terhadap kemungkinan strategi yang akan dipakai, yang disusun secara matematis agar bisa diterima secara logis dan rasional. Serta digunakan untuk mencari strategi terbaik dalam suatu aktivitas, dimana setiap pemain didalamnya sama - sama mencapai utilitas tertinggi.

2.2.2. Klasifikasi *Game*

2.2.2.1. Berdasarkan Jenis Alat Yang Digunakan

Ada beberapa jenis alat di dunia *game* yang selalu dipilih oleh pengguna *game*, yaitu [5]:

1. *Arcade games*, yaitu yang sering disebut ding-dong di Indonesia, biasanya berada di daerah / tempat khusus dan memiliki *box* atau mesin yang memang khusus di design untuk jenis *game* tertentu dan tidak jarang bahkan memiliki fitur yang dapat membuat pemainnya lebih merasa masuk dan menikmati, seperti pistol, kursi khusus, sensor gerakan, sensor injakkan dan stir mobil (beserta transmisinya tentunya).
2. PC *Games*, yaitu video *game* yang dimainkan menggunakan *Personal Computer*.
3. *Console games*, yaitu video *games* yang dimainkan menggunakan *console* tertentu, seperti Playstation 2, Playstation 3, XBOX 360, dan Nintendo Wii.
4. *Handheld games*, yaitu yang dimainkan di *console* khusus *game* yang dapat dibawa kemana-mana, contoh Nintendo DS dan Sony PSP.
5. *Mobile games*, yaitu yang dapat dimainkan atau khusus untuk *mobile phone* atau PDA.

2.2.2.2. Berdasarkan Sudut Pandang Permainan

Sebuah *game* biasanya mempunyai sudut pandang permainan tersendiri disesuaikan berdasarkan genre *game* yang diambil. Berikut beberapa macam sudut pandang permainan yang biasa digunakan:

1. Side Scrolling

Side Scrolling adalah sudut pandang permainan yang terlihat dari samping dan memungkinkan karakter utama untuk bergerak dari kiri ke kanan serta memungkinkan *background* pada *game* seolah-olah bergeser mengikuti pergerakan karakter utama.

2. Top Down

Top Down adalah sudut pandang permainan yang memungkinkan karakter utama bermanuver ke empat arah namun cara permainannya sendiri bergeser dari bawah ke atas, dan biasanya *game* yang menggunakan sudut pandang permainan jenis ini adalah *shooter game*.

3. Isometric

Isometric adalah sudut pandang permainan yang memungkinkan permainan terlihat diantara sisi *Side Scrolling* dan juga *Top Down*, dan biasanya diterapkan pada *game* dengan genre RTS (*Real Time Strategy*).

2.2.2.3. Berdasarkan Genre Permainannya

1. *Action Game*

Action Game adalah tipe *game* dengan fitur utama berupa banyaknya aksi di mana pemain harus memiliki keterampilan reaksi yang cepat untuk menghindari musuh atau menghindari rintangan.

Contohnya : Metal Gear Solid.

2. *Adventure Game*

Adventure Game adalah tipe *game* yang umumnya membuat pemain harus berjalan mengelilingi suatu tempat yang terkondisi, seperti sebuah istana, gua yang berkelok, dan planet yang jauh. Pemain melakukan navigasi suatu area mencari pesan-pesan rahasia, memperoleh obyek yang memiliki kemampuan yang bervariasi, bertempur dengan musuh, dan lain -lain.

Contohnya : Tomb Raider, Prince of Persia, Hitman, dan lain-lain.

3. *Fighting Game*

Fighting Game adalah tipe *game* yang intinya harus menjatuhkan lawan tandingnya, entah itu dengan pukulan, tendangan, combo, maupun dengan jurus special, Contohnya : Tekken, Mortal Kombat, Naruto, dan lain-lain.

4. *Shooting Game*

Shooting (tembak-tembakan, atau hajar-hajaran bisa juga tusuk-tusukan, 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, tembak dan tembak. Termasuk didalam-nya :

1. *First Person Shooting* (FPS) seperti Counter Strike dan Call Of Duty.
2. *Drive n' shoot*, menggunakan unsur simulasi kendaraan tetapi tetap dengan tujuan utama menembak dan menghancurkan lawan, contoh : Spy Hunter, Rock and Roll Racing, Road Rash.
3. *Shoot em' up*, seperti Raiden, 1942, dan Gradius.
4. *Beat em up* (tonjok hajar) seperti Double Dragon dan Final Fight, lalu hack and slash (tusuk tebas) seperti Shinobi dan Legend of Kage.
5. *Light gun shooting*, yang menggunakan alat yang umumnya berbentuk seperti senjata, seperti Virtua Cop dan Time Crisis.

5. RPG (*Role Playing Game*)

RPG (*Role Playing Game*) adalah tipe *game* yang seringkali berupa multi karakter *game* di mana setiap pemain memiliki karakter dengan kemampuan, kekuatan, dan kelemahan yang spesifik. Para pemain saling berkompetisi, berinteraksi, dan bertempur satu sama lain.

Contohnya : Final Fantasy, Warcraft, dan lain-lain.

6. *Puzzle Game*

Puzzle Game adalah tipe *game* yang umumnya membuat pemain menggunakan kemampuan berpikirnya sebagai pengganti keterampilan reaksi yang cepat karena terdapat rahasia yang perlu dipecahkan. *Game* ini lebih bersifat statis dibanding *action game*. Pembuatan *game* tipe ini seringkali ditunjang dengan kecerdasan buatan.

Contohnya : 7 Wonders of The Ancient World.

7. *Racing Game*

Racing Game adalah tipe *game* yang tujuannya adalah mencapai garis *finish* dari suatu *race*, dalam *game* ini biasanya pemain dapat memilih dan membeli kendaraan, memodifikasi, melakukan *upgrade* mesin.

Contohnya : Need For Speed.

8. *Simulation Game*

Simulation Game adalah tipe *game* yang memberikan pengalaman atau interaksi sedekat mungkin dengan keadaan yang aslinya, meskipun terkadang keadaan tersebut masih eksperimen atau bahkan fiktif, tapi ada penekanan khusus pada detail dan pengalaman *realistik* menggunakan keadaan tersebut.

Contohnya : Gran Turismo.

9. RTS (*Real Time Strategy*)

RTS (*Real Time Strategy*) adalah tipe *game* yang mengharuskan pemainnya menggunakan taktik dan strategi untuk jeli dalam melihat setiap peluang, kelemahan musuh dan bijaksana dalam menggunakan sumber daya yang ada.

Contohnya : Command and Conquer.

10. *Sport Game*

Sport Game adalah tipe *game* yang 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.

Contohnya : Pro Evolution Soccer.

2.2.2.4. Berdasarkan Batasan Usia

Berikut ini adalah klasifikasi permainan berdasarkan pembatasan usia dari pemainnya, menurut ESRB (*Entertainment Software Rating Board*) adalah sebagai berikut [6]:

1. *Early Childhood* (3+) memiliki isi yang mungkin cocok untuk usia 3 keatas. Tidak mengandung materi yang tidak pantas seperti adegan kekerasan.

2. *Everyone* (6+) memiliki konten yang mungkin cocok untuk usia 6 keatas. Judul-judul dalam kategori ini mungkin berisi sedikit kartun, fantasi atau kekerasan ringan dan tidak jarang menggunakan bahasa yang ringan.
3. *Everyone* (10+) memiliki konten yang mungkin cocok untuk usia 10 keatas. Judul-judul dalam kategori ini mungkin berisi lebih banyak kartun, fantasi atau kekerasan ringan, bahasa ringan dan juga minimal tema sugestif.
4. *Teen* (13+) memiliki konten yang mungkin cocok untuk usia 13 keatas. Judul-judul dalam kategori ini mungkin mengandung kekerasan, tema sugestif, humor kasar, sedikit darah, simulasi perjudian, dan tidak jarang menggunakan bahasa yang kuat.
5. *Mature* (17+) memiliki konten yang mungkin cocok untuk orang usia 17 dan lebih tua. Judul-judul dalam kategori ini mungkin berisi intens kekerasan, darah, konten seksual dan bahasa yang kuat.
6. *Adults Only* (18+) memiliki konten yang hanya boleh dimainkan oleh orang-orang 18 tahun keatas. Judul-judul dalam kategori ini mungkin termasuk adegan yang intens, kekerasan dan juga mengandung konten seksual.

2.3. Kecerdasan Buatan

Kecerdasan buatan adalah cabang ilmu komputer yang berfokus pada otomatisasi tingkah laku cerdas sebuah sistem. Kecerdasan buatan adalah bagian ilmu komputer yang harus didasarkan pada *sound theoretical* dan prinsip-prinsip aplikasi dibidangnya. Prinsip ini adalah struktur data untuk merepresentasikan pengetahuan, algoritma yang diperlukan untuk mengaplikasikan pengetahuan, dan bahasa serta teknik pemrograman yang digunakan untuk implementasi [7].

2.3.1. Implementasi Kecerdasan Buatan

Implementasi kecerdasan buatan dapat diterapkan pada bidang:

1. Visualisasi komputer

   Kecerdasan buatan pada bidang visualisasi komputer ini memungkinkan sebuah sistem komputer mengenali gambar sebagai input.

   Contohnya mengenali sebuah pola pada suatu gambar.

2. Pengenalan Suara

   Kecerdasan buatan pada pengenalan suara ini dapat mengenali suara manusia. Cara mengenali suara ini dengan mencocokannya pada acuan yang telah diprogramkan terlebih dahulu.

   Contohnya perintah komputer dengan menggunakan suara user.

3. Sistem Pakar

   Kecerdasan buatan pada Sistem Pakar ini memungkinkan sebuah sistem komputer memiliki cara berpikir dan penalaran seorang ahli dalam mengambil keputusan, untuk memecahkan masalah yang ada pada saat itu. Contohnya program komputer yang dapat mendiagnosa penyakit dengan memasukan gejala-gejala yang dialami pasien.

4. *Game*

   Kecerdasan buatan pada *game* ini memungkinkan sebuah sistem komputer untuk memiliki cara berpikir manusia dalam bermain.

   Contohnya *game* yang memiliki fasilitas orang melawan komputer. Komputer sudah di program sedemikian rupa agar memiliki cara bermain seperti seorang manusia bahkan bisa melebihi seorang manusia.

2.3.2. Kecerdasan Buatan Pada *Game*

2.3.2.1. Algoritma Pencarian

Terdapat banyak metode pencarian yang ada. Semua metode yang ada dapat dibedakan ke dalam dua jenis yaitu :

1. Pencarian Buta (*Blind/Un-informed Search*)

Blind/Un-informed Search adalah pencarian solusi tanpa adanya informasi yang dapat mengarahkan pencarian untuk mencapai *goal state* dari *current state* (keadaan sekarang). Informasi yang ada hanyalah definisi *goal state* itu sendiri, sehingga algoritma dapat mengenali *goal state* bila menjumpainya. Dengan ketiadaan informasi, maka *blind search* dalam kerjanya memeriksa atau mengembangkan *node-node* secara tidak terarah dan kurang efisien untuk kebanyakan kasus karena banyaknya node yang dikembangkan.

Beberapa contoh algoritma yang termasuk blind search antara lain adalah *Breadth First Search* (BFS), *Uniform Cost Search* (UCS), *Depth First Search* (DFS), *Depth-Limited Search* (DLS), *Iterative-Deepening Search* (IDS), dan *Bi-directional Search* (BDS).

2. Pencarian Terbimbing (*Heuristic/Informed Search*)

Berbeda dengan *blind search*, *heuristic search* mempunyai informasi tentang *cost*/biaya untuk mencapai *goal state* dari *current state*. Dengan informasi tersebut, *heuristic search* dapat melakukan pertimbangan untuk mengembangkan atau memeriksa *node-node* yang mengarah ke *goal state*. Misalnya pada pencarian rute pada suatu peta, bila berangkat dari kota A ke kota tujuan B yang letaknya di Utara kota A, dengan *heuristic search*, pencarian akan lebih difokuskan ke arah Utara (dengan informasi *cost* ke *goal*), sehingga secara umum, *heuristic search* lebih efisien daripada *blind search*.

Heuristic search untuk menghitung perkiraan *cost* ke *goal state*, digunakan fungsi heuristik. Fungsi heuristik berbeda daripada algoritma, dimana heuristik lebih merupakan perkiraan untuk membantu algoritma, dan tidak harus valid setiap waktu. Meskipun begitu, semakin bagus fungsi heuristik yang dipakai, semakin cepat dan akurat pula solusi yang didapat. Menentukan heuristik yang tepat untuk kasus dan implementasi yang ada juga sangat berpengaruh terhadap kinerja algoritma pencarian.

2.3.2.2. Fungsi Heuristik

Algoritma A* adalah algoritma pencarian yang menggunakan fungsi heuristik untuk ‘menuntun’ pencarian rute, khususnya dalam hal pengembangan dan pemeriksaan node-node pada peta. Fungsi-fungsi heuristik yang biasa digunakan dalam algoritma A*, antara lain [8]:

1. Manhattan

Fungsi ini merupaka fungsi heuristik yang paling umum digunakan. Fungsi ini hanya akan menjumlahkan selisih nilai *x* dan nilai *y* dari dua buah titik. Fungsi heuristik ini dinamakan Manhattan karena berasal dari Manhattan di Amerika,

jarak dari dua lokasi umumnya dihitung dari blok-blok yang harus dilalui saja dan tentunya tidak bisa dilintasi secara diagonal. Perhitungannya dapat ditulis sebagai berikut :

$$h(n) = abs(tujuan.x - n.x) + abs(tujuan.y - n.y) \quad \text{..... (2.1)}$$

Dimana *h(n)* merupakan perkiraan *cost node n* ke *node* tujuan yang dihitung dengan fungsi heuristik. Variabel *n.x* merupakan koordinat *x* dari *node n*, sedangkan *n.y* merupakan koordinat *y* dari *node n*. Variabel *tujuan.x* merupakan koordinat *x* dari node tujuan dan *tujuan.y* merupakan koordinat *y* dari *node* tujuan. Nilai *h(n)* akan selalu bernilai positif.

2. Euclidian

Heuristik ini akan menghitung jarak berdasarkan panjang garis yang dapat ditarik dari dua buah titik. Perhitungannya dapat ditulis sebagai berikut :

$$h(n) = \sqrt{((tujuan.x - n.x)^2 + (tujuan.y - n.y)^2)} \quad \text{..... (2.2)}$$

Dalam kasus ini, skala relatif nilai *g* mungkin akan tidak sesuai lagi dengan nilai fungsi heuristik *h*. Karena jarak Euclidian selalu lebih pendek dari jarak Manhattan, maka dapat dipastikan selalu akan didapatkan jalur terpendek, walaupun secara komputasi lebih berat.

3. Euclidian Kuadrat

Dalam beberapa literatur juga disebutkan jika nilai g adalah 0, maka lebih baik jika ongkos komputasi operasi pengakaran pada heuristik jarak Euclidian dihilangkan saja, menghasilkan rumus sebagai berikut :

$$h(n) = (tujuan.x - n.x)^2 + (tujuan.y - n.y)^2 \quad \text{..... (2.3)}$$

2.3.2.3. Algoritma A*

Algoritma A* 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. Dengan perhitungan biaya seperti ini algoritma A* adalah *complete* dan *optimal* [7].

Beberapa terminologi dasar yang terdapat pada algoritma ini adalah *starting point, current node, simpul, open list, closed list, harga (cost), walkability, target point.*

1. *Starting point* adalah sebuah terminologi untuk posisi awal sebuah benda.
2. *Current node* adalah node yang sedang dijalankan dalam algortima pencarian jalan terpendek.
3. *Node* adalah petak-petak kecil sebagai representasi dari area *pathfinding*. Bentuknya dapat berupa persegi, lingkaran, maupun segitiga.
4. *Open list* adalah adalah list yang menyimpan kemungkinan path yang akan diperiksa. *Open list* dibuat terurut berdasarkan nilai f. *Open list* digunakan untuk menentukan secara selektif (berdasarkan nilai f) jalan yang dikira lebih dekat menuju pada path tujuan. *Open list* berisi simpul-simpul yang masih memiliki peluang untuk terpilih sebagai simpul terbaik (*best node*).
5. *Closed list* adalah tempat untuk menyimpan simpul-simpul yang sudah pernah dibangkitkan dan sudah pernah terpilih sebagai simpul terbaik (*best node*) atau *list* yang menyimpan jalan yang sudah diperiksa dari *open list*. Artinya, *Closed list* berisi simpul-simpul yang tidak mungkin terpilih sebagai simpul terbaik (peluang untuk terpilih sudah tertutup). Kedua list (*Open list* dan *Closed list*) ini bertujuan juga untuk menghindari penelusuran jalan (*rute*) berkali-kali yang memang sudah diidentifikasi agar tidak masuk kembali ke dalam *Open list*.
6. Nilai f adalah harga perkiraan suatu path yang teridentifikasi. Nilai f merupakan hasil dari f(n).
7. Nilai g hasil dari fungsi g(n), adalah banyaknya langkah yang diperlukan untuk menuju ke path sekarang.
8. Nilai h hasil dari fungsi h(n), adalah estimasi harga simpul yang dihitung dari simpul tujuan.
9. *Target point* yaitu node yang dituju.
10. *Walkability* adalah sebuah atribut yang menyatakan apakah sebuah node dapat atau tidak dapat dilalui oleh *current node*.

Fungsi f sebagai estimasi fungsi evaluasi terhadap node n, dapat dituliskan :

$$f(n) = g(n) + h(n) \quad \text{... (2.4)}$$

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.

Prinsip algoritma ini adalah mencari jalur terpendek dari sebuah simpul awal (*starting point*) menuju simpul tujuan dengan memperhatikan harga (f) terkecil. Diawali dengan menempatkan A pada *starting point*, kemudian memasukkan seluruh simpul yang bertetangga dan tidak memilik atribut rintangan dengan A ke dalam *open list*. Kemudian mencari nilai h terkecil dari simpul-simpul dalam *open list* tersebut. Kemudian memindahkan A ke simpul yang memiliki nilai h terkecil. Simpul sebelum A disimpan sebagai *parent* dari A dan dimasukkan ke dalam *closed list*. Jika terdapat simpul lain yang bertetangga dengan A (yang sudah berpindah) namun belum termasuk kedalam anggota *open list*, maka masukkan simpul-simpul tersebut ke dalam *open list*. Setelah itu, bandingkan nilai g yang ada dengan nilai g sebelumnya (pada langkah awal, tidak perlu dilakukan perbandingan nilai g). Jika nilai g sebelumnya lebih kecil maka A kembali ke posisi awal. Simpul yang pernah dicoba dimasukkan ke dalam *closed list*. Hal terebut dilakukan berulang-ulang hingga terdapat solusi atau tidak ada lagi simpul lain yang berada pada *open list*.

2.4. Pemrograman Berbasis Objek

Pemrograman Berorientasi Objek atau *Object Oriented Programming* (OOP) merupakan suatu pendekatan kepada pengembangan perangkat lunak dimana struktur perangkat lunaknya berdasarkan interaksi objek satu sama lain untuk menyelesaikan suatu pekerjaan. Interaksi ini digambarkan dengan pengiriman dan penerimaan pesan antara dua objek. Dalam merespon sebuah pesan, sebuah objek dapat melakukan suatu aksi [9].

Pemrograman Berorientasi Objek atau OOP memiliki karakteristik sebagai berikut:

1. Analisis dan Desain Berorientasi Objek (*Object-Oriented Analysis and Design Process*)

Pemrograman berorientasi objek bekerja dengan baik ketika dibarengi dengan *Object-Oriented Analysis and Design Process* (OOAD). Wrampler mengatakan jika membuat program berorientasi objek tanpa OOAD, ibarat membangun rumah tanpa terlebih dahulu menganalisa apa saja yang dibutuhkan oleh rumah itu, tanpa perencanaan, tanpa blueprint, tanpa menganalisis ruangan apa saja yang diperlukan, berapa besar rumah yang akan dibangun dan sebagainya.

2. Objek

Menurut Douglas objek adalah entitas yang memiliki atribut, karakter (*behavior*) dan kadangkala disertai kondisi (*state*). Objek merepresentasikan sesuatu sistem real seperti siswa, sistem kontrol permukaan sayap pesawat, sensor atau mesin. Objek juga merepresentasikan sesuatu dalam bentuk konsep seperti nasabah bank, merek dagang, pernikahan atau sekedar *listing*. Bahkan bisa juga menyatakan visualisasi seperti, bentuk huruf (*font*), histogram, poligon, garis atau lingkaran.

3. Kelas

Kelas adalah penggambaran satu set objek yang memiliki atribut dan behavior yang sama. Kelas mirip tipe data pada pemrograman non objek, tapi lebih komprehensif karena terdapat struktur sekaligus karakteristiknya.

4. Abstraksi

Ketika berinteraksi dengan suatu objek, kita biasanya hanya terpaku pada bagian dari propertinya. Tanpa kemampuan abstraksi atau menyaring properti dari suatu objek, kita pasti kesulitan dalam memproses informasi atau mengerjakan suatu tugas.

Sebagai hasil dari abstraksi, ketika dua orang berinteraksi dengan objek yang sama, mereka biasanya berurusan dengan sebagian atribut yang berbeda. Sebagai contoh, ketika kita mengendarai mobil, kita hanya perlu mengetahui kecepatan

dan arah yang dituju oleh mobil. Dikarenakan mobil yang kita pakai menggunakan transmisi otomatis, kita tidak perlu mengetahui RPM (*revolution per minute*) dari mesin, jadi informasi ini dibuang. Di lain sisi, informasi ini boleh jadi merupakan informasi penting bagi seorang pembalap, jadi tidak akan dibuang informasi tersebut. Ketika membangun objek dalam aplikasi OOP, penting untuk menghubungkan konsep abstraksi ini. Objek hanya mengandung informasi yang berhubungan dengan konteks aplikasi.

5. Enkapsulasi

Fitur penting lain dari OOP adalah enkapsulasi. Enkapsulasi adalah proses dimana akses langsung terhadap data tidak diperbolehkan. Dengan enkapsulasi, data yang ada di sitem menjadi lebih aman dan dapat diandalkan.

6. Polimorfisme

Polimorfisme adalah kemampuan dari dua objek berbeda untuk merespon pesan yang sama dalam cara unik mereka sendiri. Sebagai contoh, kita dapat melatih anjing untuk merespon perintah menggonggong dan burung untuk merespon perintah berkicau. Di lain sisi, kita dapat melatih keduanya untuk merespon perintah berbicara. Melalui polimorfisme, anjing akan merespon dengan menggonggong dan burung akan merespon dengan berkicau.

Dalam OOP, polimorfisme dapat diimplementasikan melalui sebuah proses yang disebut *overloading*. Dengan *overloading*, metode atau prosedur yang berbeda dapat diimplementasikan untuk objek yang sama.

7. Pewarisan

Pewarisan atau *inheritance* dalam OOP digunakan untuk mengelompokkan objek dalam program sesuai karakteristik umum dan fungsinya. Hal ini membuat pekerjaan dengan objek menjadi lebih mudah dan lebih intuitif. Pewarisan juga membuat pemrograman lebih mudah karena karakteristik umum dapat disimpan ke dalam objek induk dan dapat diwariskan ke semua objek anak.

8. Agregasi

Agregasi adalah ketika sebuah objek terdiri dari gabungan objek lain yang bekerja bersama. Sebagai contoh, objek mesin pemotong rumput merupakan

gabungan dari objek roda, objek mesin, objek besi, dan seterusnya. Agregasi dalam OOP merupakan fitur yang memungkinkan kita secara akurat memodelkan dan mengimplementasikan proses bisnis dalam suatu program [10].

2.5. UML *(Unified Modeling Language)*

Unified Modeling Language merupakan salah satu alat bantu yang dapat digunakan dalam bahasa pemograman yang berorientasi objek, saat ini UML akan mulai menjadi standar masa depan bagi industri pengembangan sistem/perangkat lunak yang berorientasi objek sebab pada dasarnya UML digunakan oleh banyak perusahaan raksasa seperti IBM, Microsoft, dan sebagainya.

Definisi UML :

1. *Unified Modeling Language* merupakan metode pengembangan perangkat lunak dengan menggunakan metode grafis serta merupakan bahasa untuk visualisasi, spesifikasi, konstruksi serta dokumentasi.
2. *Unified Modeling Language* adalah bahasa yang telah menjadi standard untuk visualisasi, menetapkan, membangun dan mendokumentasikan arti suatu sistem perangkat lunak.
3. *Unified Modeling Language* merupakan standard modeling language yang terdiri dari kumpulan-kumpulan diagram, dikembangkan untuk membantu para pengembang sistem dan software agar bisa menyelesaikan tugas-tugas seperti :
    1. Spesifikasi
    2. Visualisasi
    3. Desain arsitektur
    4. Konstruksi
    5. Simulasi dan testing
    6. Dokumentasi

Berdasarkan beberapa pendapat yang dikemukakan diatas dapat ditarik kesimpulan bahwa "*Unified Modeling Language* (UML) adalah sebuah bahasa yang berdasarkan grafik atau gambar untuk menvisualisasikan, menspesifikasikan, membangun dan pendokumentasian dari sebuah sistem

pengembangan perangkat lunak berbasis Objek (*Object Oriented programming*)" [9].

2.5.1. Diagram UML

UML menyediakan 9 macam diagram untuk memodelkan aplikasi berorientasi objek. berikut akan dijelaskan 4 macam diagram yang paling sering digunakan dalam pembangunan aplikasi berorientasi object, yaitu *use case diagram, sequence diagram, activity diagram, dan class diagram*.

1. *Use Case Diagram*

Use case diagram digunakan untuk memodelkan bisnis proses berdasarkan perspektif pengguna sistem. *Use case diagram* terdiri atas diagram untuk *use case* dan *actor*. *Actor* merepresentasikan orang yang akan mengoperasikan atau orang yang berinteraksi dengan sistem aplikasi. Selain itu, *Use case* merepresentasikan operasi-operasi yang dilakukan oleh actor. *Use case* digambarkan berbentuk elips dengan nama operasi dituliskan di dalamnya. *Actor* yang melakukan operasi dihubungkan dengan garis lurus ke *use case*.

2. *Sequence Diagram*

Sequence diagram menjelaskan secara detail urutan proses yang dilakukan dalam sistem untuk mencapai tujuan dari *use case* yaitu interaksi yang terjadi antar *class*, operasi apa saja yang terlibat, urutan antar operasi, dan informasi yang diperlukan oleh masing-masing operasi.

3. *Activity Diagram*

Activity diagram digunakan untuk menggambarkan aliran aktivitas, langkah-langkah penyelesaian dan interaksi yang terjadi antara pengguna dengan sistem dalam suatu proses.

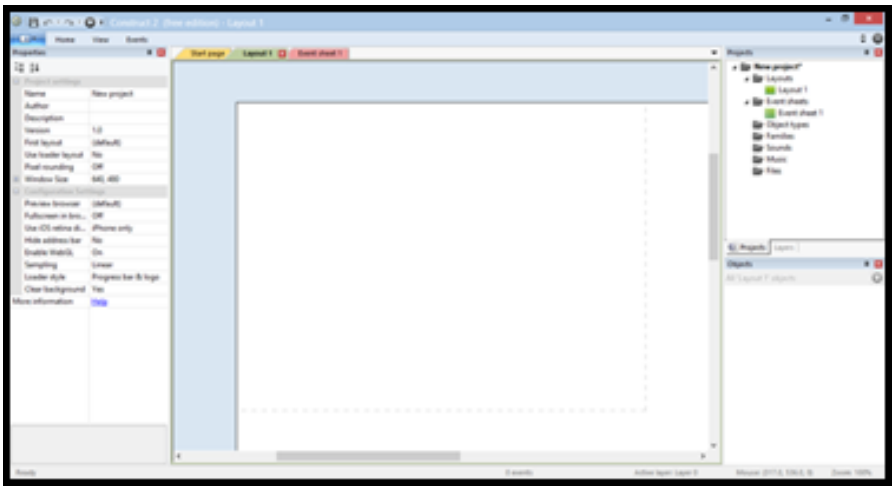
4. *Class Diagram*

Class diagram merupakan diagram yang selalu ada di permodelan sistem berorientasi objek. *Class diagram* menunjukkan hubungan antar *class* dalam sistem yang sedang dibangun dan bagaimana mereka saling berkolaborasi untuk mencapai suatu tujuan.

2.6. Construct 2

Construct adalah *Game Engine* yang dikembangkan oleh Scirra yang salah satu *platform* didukungnya adalah membangun apalikasi *game* untuk Windows 8. Construct 2 sangat mudah untuk di pelajari berbeda dengan pengembangan *game* pada umumnya Construct 2 memudahakan pengembang *game* untuk membuat aplikasi dengan metode *visual programming*, yaitu *drag & drop* modul yang telah disediakan dengan kebutuhan coding yang minimal. Membuat *Game* untuk Windows 8 dengan Construct 2 tidaklah susah, disini seorang desain sangat diperlukan. Construct 2 selain bisa digunakan di Windows, juga dapat dijalankan di banyak platform, seperti Mac, Linux, dan iPad. Construct 2 juga dapat diinstal pada user account yang terbatas.

Penjelasan tentang tools dari Construct 2



Gambar 2.1 Tampilan Construct 2

1. *Properties*, bagian untuk menambahkan data value dari *game* yang akan dibangun, seperti nama project, deskripsi project, author, dan ukuran dari layar aplikasi.
2. *Layout*, bagian untuk membuat design dari *game* yang akan dibangun seperti penempatan *actor game*, dan *object-object* didalam *game*. Pada bagian ini peran designer *game* akan sangat diperlukan untuk memberikan *game* dengan nuansa yang menarik.

3. *Event sheet*, bagian untuk menempatkan proses logic dari aplikasi *game* yang akan dibangun. Proses logic akan menggunakan *visual programming* sehingga lebih mudah dipahami untuk para pemula maupun *developer* yang backgroundnya Non-IT.
4. *Project*, bagian yang menampilkan keseluruhan data dari project yang sedang dikerjakan. Folder *Music*, *Sound*, *Family*, *Layout*, *Event*, seluruhnya dapat dilihat melalui project explorer.
5. *Objects*, merupakan pallet yang menampilkan daftar dari object-oject yang digunakan dalam project.

2.7. Corel Draw

Corel Draw adalah sebuah program komputer yang melakukan editing pada garis vektor. Program ini dibuat oleh Corel, sebuah perusahaan software yang berkantor pusat di Ottawa, Kanada. Corel draw memiliki kegunaan untuk mengolah gambar, oleh karena itu banyak digunakan pada pekerjaan dalam bidang publikasi atau percetakan ataupun pekerjaan di bidang lain yang membutuhkan proses visualisasi.

2.7.1. Kegunaan Corel Draw

Program Corel Draw seringkali dimanfaatkan untuk melakukan pekerjaan-pekerjaan dibawah ini, yaitu [11]:

1. Menciptakan desain logo atau simbol, yang mana ini adalah kegunaan Corel Draw yang paling banyak dimanfaatkan oleh penggunanya, terutama pembuatan logo dua dimensi karena kemudahannya dalam mengolah garis dan warna.
2. Membuat desain undangan, brosur dan lain-lain juga menjadi suatu kegunaan dari program Corel Draw. Media publikasi offline lainnya juga menggunakan Corel Draw sebagai alat untuk mendesain. Corel Draw memiliki banyak jenis font yang dapat memudahkan desainer untuk mengeksplorasi imajinasi desain dan tulisan yang akan dibuat.

3. Membuat cover buku juga dapat dilakukan di Corel Draw. Dengan Corel Draw maka tugas desain akan menjadi mudah karena dapat memanfaatkan desain sampul dan teknik pewarnaan yang lebih sempurna oleh Corel Draw. Detail gambar pun akan terlihat lebih jelas.
4. Pembuatan gambar ilustrasi juga dapat dilakukan dengan Corel Draw. Gambar yang dihasilkan lebih berkualitas, terutama ketika berhubungan dengan lengkungan, garis atau sudut. Ukuran yang diperoleh dijamin sangat akurat.

2.7.2. Keunggulan Corel Draw

Beberapa keunggulan program Corel Draw antara lain adalah:

1. Gambar yang dihasilkan dengan vektor atau berbasis vektor bisa ditekan pada tingkat yang paling rendah namun hasilnya tidak kalah dengan gambar yang berbasis bitmap atau raster.
2. Penggunaan Corel Draw, terutama pada tool-tool yang ada di dalamnya sangat mudah dipahami oleh penggunanya, bahkan oleh orang yang baru pertama menggunakannya.
3. Corel Draw sangat baik untuk mengkolaborasikan antara tulisan dengan gambar, seperti layaknya Adobe Potoshop.
4. Banyaknya pengguna Corel Draw, membuat adanya komunitas dengan jumlah anggota yang besar. Hal ini akan membuat kita tidak kesulitan jika ingin mempelajari Corel Draw lebih mendalam karena banyak ditemukan tutorial, tips dan trik yang diproduksi oleh pengguna lain.

2.8. Teknik Pengumpulan Data

Teknik pengumpulan data merupakan faktor penting demi keberhasilan penelitian. Hal ini berkaitan dengan bagaimana cara mengumpulkan data, siapa sumbernya, dan apa alat yang digunakan. Jenis sumber data adalah mengenai dari mana data diperoleh. Apakah data diperoleh dari sumber langsung (data *primer*) atau data diperoleh dari sumber tidak langsung (data *sekunder*).

Metode Pengumpulan Data merupakan teknik atau cara yang dilakukan untuk mengumpulkan data. Metode menunjuk suatu cara sehingga dapat diperlihatkan

penggunaannya melalui angket, wawancara, pengamatan, tes, dokumentasi dan sebagainya. Sedangkan Instrumen Pengumpul Data merupakan alat yang digunakan untuk mengumpulkan data. Karena berupa alat, maka instrumen dapat berupa lembar cek list, kuesioner (angket terbuka/tertutup), pedoman wawancara, camera photo dan lainnya.

1. Angket

Angket / kuesioner adalah teknik pengumpulan data yang dilakukan dengan cara memberikan seperangkat pertanyaan atau pernyataan kepada orang lain yang dijadikan responden untuk dijawabnya.

Beberapa hal yang perlu diperhatikan dalam penyusunan angket menurut Uma Sekaran terkait dengan prinsip penulisan angket, prinsip pengukuran dan penampilan fisik.

Prinsip Penulisan angket menyangkut beberapa faktor antara lain :

1. Isi dan tujuan pertanyaan artinya jika isi pertanyaan ditujukan untuk mengukur maka harus ada skala yang jelas dalam pilihan jawaban.
2. Bahasa yang digunakan harus disesuaikan dengan kemampuan responden. Tidak mungkin menggunakan bahasa yang penuh istilah-istilah bahasa Inggris pada responden yang tidak mengerti bahasa Inggris, dsb.
3. Tipe dan bentuk pertanyaan apakah terbuka atau terturup. Jika terbuka artinya jawaban yang diberikan adalah bebas, sedangkan jika pernyataan tertutup maka responden hanya diminta untuk memilih jawaban yang disediakan.

2. Observasi

Obrservasi merupakan salah satu teknik pengumpulan data yang tidak hanya mengukur sikap dari responden (wawancara dan angket) namun juga dapat digunakan untuk merekam berbagai fenomena yang terjadi (situasi, kondisi). Teknik ini digunakan bila penelitian ditujukan untuk mempelajari perilaku manusia, proses kerja, gejala-gejala alam dan dilakukan pada responden yang tidak terlalu besar.

2.9. Metode Pengujian Perangkat Lunak

2.9.1. Metode Pengujian *Black Box*

Balck box adalah cara pengujian yang dilakukan dengan hanya menjalankan atau mengeksekusi unit atau modul kemudian diamati apakah hasil dari unit itu sesuai dengan proses yang dinginkan. Dengan kata lain, *black box* merupakan *user testing*, biasanya pengujian perangkat lunak dengan metode *black box* melibatkan *client* atau pelanggan yang memesan produk perangkat lunak tersebut. Dalam pengujian *black box* pemrogram berusaha menemukan kesalahan seperti [13]:

a. Fungsi – fungsi yang tidak benar atau hilang.
b. Kesalahan *Interface*.
c. Kesalahan dalam struktur data atau akses *database internal*.
d. Kesalahan kinerja.
e. *Inisialisasi* dan kesalahan terminasi.

2.9.2. Metode Pengujian *White Box*

Pengujian *white box* merupakan metode desain uji kasus yang menggunakan *struktur control* dari *desain procedural* untuk menghasilkan kasus-kasus uji. Dengan menggunakan metode uji coba *white box*, para pengembang *software* dapat menghasilkan kasus – kasus uji yang [13]:

a. Menjamin bahwa *independent paths* dalam modul telah dilakukan sedikitnya satu kali.
b. Melakukan seluruh keputusan logikal dari sisi benar maupun salah.
c. Melakukan seluruh perulangan sesuai batasannya dan dalam batasan operasionalnya.
d. Menguji struktur data *internal* untuk memastikan validitasnya.

Menguji *logikal software* membutuhkan banyak waktu dan usaha, hal ini dikarenakan sifat kerusakan alami dari *software* itu sendiri, yaitu :

a. Kesalahan logika dan kesalahan asumsi secara proposional terbalik dengan kemungkinan bahwa program akan dieksekusi. Kesalahan akan selalu ada

ketika mendesain dan implementasi fungsi, kondisi dan kontrol yang keluar dari alur utama. Setiap harinya pemrosesan selalu berjalan dengan baik dan dimengerti sampai bertemu “kasus sepesial” yang mengarahkan kepada kehancuran.

b. Alur logika program biasanya berkebalikan dari intuisi, yaitu tanpa disadari asumsi dari alur kontrol dan dapat mengarah kepada pada kesalahan desain yang tidak dapat terlihat hanya dengan satu kali uji coba.
c. Kesalahan *typographical* (cetak) bersifat *random*, ketika program diterjemahkan kedalam kode sumber bahasa pemrograman, maka akan terjadi kesalahan pengetikan. Banyak yang terdeteksi dengan mekanisme pemeriksaan *sintaks*, tetapi banyak juga yang tidak terdeteksi sampai dengan dimulainya ujicoba.
d. Karena alasan tersebut diatas, maka uji coba *white box testing* perlu dilakukan selain *balck box testing*.