BAB 2

LANDASAN TEORI

2.1. *Game*

Video game merupakan salah satu media hiburan yang sangat populer untuk semua orang, baik itu tua maupun muda. Teknologi *game* telah mengalami kemajuan yang sangat pesat. Terbukti hal ini ditandai dengan berkembangnya jenis, produk dan alat yang beragam untuk berinteraksi dengan *game*.

2.1.1. Pengertian *Game*

Game berasal dari kata bahasa inggris yang memiliki arti dasar Permainan. Permainan dalam hal ini merujuk pada pengertian "kelincahan intelektual" (*intellectual playability*). *Game* juga bisa diartikan sebagai arena keputusan dan aksi pemainnya. Ada target-target yang ingin dicapai pemainnya.

2.1.2. Sejarah Singkat Perkembangan *Game*

Asal usul *video game* dimulai pada awal tabung sinar katoda berbasis pertahanan peluru kendali sistem pada akhir 1940-an. Program-program ini kemudian diadaptasi ke dalam permainan sederhana lainnya di era tahun 1950-an. Pada akhir 1950-an dan melalui tahun 1960-an, lebih banyak permainan komputer yang dikembangkan (kebanyakan di komputer *mainframe*), secara bertahap tingkat kecanggihan dan kompleksitasnya pun turut bertambah. Setelah periode ini, *video game* mulai berpindah ke berbagai platform: *arcade, mainframe,* konsol, pribadi komputer dan kemudian permainan genggam.

Generasi pertama konsol permainan *video* berlangsung dari tahun 1972, dengan dirilisnya *Magnavox Odyssey* hingga 1977, ketika "pong"-*style* produsen konsol meninggalkan pasar secara massal karena pengenalan dan keberhasilan mikroprosesor berbasis konsol. Dalam sejarah komputer dan *video game,* generasi kedua (biasa disebut sebagai awal era 8 bit atau kurang lebih 4 bit) dimulai pada tahun 1976 dengan merilis *Fairchild Channel F* dan *Radofin* 1292 *Advanced*

Programmable Video system. Di era generasi kedua ini yang menjadi primadona konsol *game* adalah konsol *game* ATARI.

Generasi ketiga dimulai pada tahun 1983 dengan dipasarkannya *Japan Family Computer* atau lebih dikenal dengan nama FAMICOM (kemudian dikenal sebagai *Nintendo Entertainment System* di seluruh dunia). FAMICOM juga termasuk sistem 16-bit seperti *Mega Drive* / *Genesis*. Di era ini pulalah terjadi perang konsol *game* yang pertama antara perusahaan konsol Nintendo dengan SEGA.

Generasi keempat atau biasa disebut dengan era 16 bit, pada generasi ini NES mendapat sambutan hangat di seluruh dunia, dan sebuah perusahaan bernama Sega mencoba menyaingi Nintendo. Pada tahun 1988, Sega merilis konsol *Sega Mega Drive* (yang juga dikenal dengan *Sega Genesis*). Konsol ini menyajikan gambar yang lebih tajam dan animasi yang lebih halus dibanding NES. Konsol ini cukup berhasil memberi tekanan, tetapi NES tetap bertahan dengan angka penjualan tertinggi. Dua tahun berselang, pada 1990, Nintendo kembali menggebrak dengan konsol mereka, SNES (*Super Nintendo Entertainment System*). Selama 4 tahun, Nintendo dan Sega menjadi bebuyutan, meskipun ada beberapa produsen seperti SNK dengan NeoGeo-nya, NEC dengan *TurboGrafx*-16 dan *Phillips CD-i*, tetapi Nintendo dan Sega begitu handal dan populer. Generasi kelima atau disebut juga dengan era konsol 32 bit. dimana konsol *game* yang paling populer pada generasi ini adalah *Sony Playstation*.

Generasi keenam ini ditandai dengan munculnya konsol-konsol *game next-generation* dari masing-masing perusahaan seperti SONY, SEGA, Nintendo serta munculnya satu lagi konsol *game* baru yang diluncurkan oleh Microsoft yang diberi nama *Xbox*. Perang konsol *game* ini akhirnya mengakibatkan jatuhnya perusahaan konsol SEGA yang tidak dapat lagi meneruskan konsol generasi selanjutnya (*Dreamcast*) mereka dan lebih memilih untuk berkonsentrasi dibidang pembuatan *game* konsol.

Dikarenakan semakin canggihnya teknologi di bidang teknologi maka kemudian 3 perusahaan konsol terbesar (Sony, Nintendo, dan Microsoft) mengeluarkan kembali konsol *next-generation* mereka dengan keluaran terbaru. Sony mengeluarkan konsol selanjutnya yang diberi nama PS3 (Playstation 3), lalu Nintendo dengan Nintendo Wii kemudian Microsoft dengan Xbox 360. Pada generasi ini semakin dikembangkan sistem permainan *online* atau permainan yang melibatkan banyak pemain yang terhubung dengan antar konsol dan semakin ditinggalkannya permainan *single player*.

2.1.3. Jenis (*genre*) *Game*

Game dapat dibagi ke beberapa genre diantaranya [4]:

1. Sport

   Genre yang satu ini terdiri dari game yang menciptakan tiruan dari *sports* asli jadi pemain bisa merasakan seperti bermain *sports* asli hanya dengan memainkan *game*-nya Contohnya : *Pro Evolution Soccer, Fifa, Snow Board Hero.*



Gambar 2.1 Contoh dari permainan ber-*genre sports*

2. *Action*

   Genre ini menyediakan pemain dengan beberapa rintangan dan musuh yang harus dikalahkan dengan menggunakan reaksi cepat sang pemain dan biasanya tidak memberikan banyak teka-teki pada pemain.

   Contohnya : *Devil May Cry, Resident Evil.*



Gambar 2.2 Contoh permainan *Action.*

3. *Adventure*

   Genre game ini terdiri dari *game* yang ada beberapa tantangan untuk pemain selama pemain mengendalikan 1 atau lebih karakter yang berkelana di tempat yang disediakan.

   Contohnya : *Space Quest IV, Labyrinth of Word, Tintin.*



Gambar 2.3 Contoh dari *game Adventure.*

4. *Simulation*

   Genre game ini menyediakan simulasi dari aktifitas asli, jadi pemain diharapkan untuk mempunyai pengalaman yang sama tanpa melibatkan aktifitas asli.

   Contohnya : *Sim City, Roller Coster Tycoon, Caesar*.



Gambar 2.4 Contoh dari *Simulation Game*

5. *Strategy*

   Jenis permainan ini pemain diberi kesempatan untuk memikirkan strategi untuk menyelesaikan masalah atau mendapatkan gol, biasanya *setting*-nya tentang peperangan.

   Contohnya : *The Ancient Art of War*, *Command and Conquer*, dan *Warcraft*



Gambar 2.5 Contoh dari *game strategy*

6. *Puzzle*

   Game ini tentang pemain mencoba untuk menyelesaikan teka-teki yang diberikan oleh *game*.

   Contohnya : *Tetris.*



Gambar 2.6 Contoh *game puzzle.*

7. *Role Playing Game (RPG)*

Jenis permainan ini pemain diharapkan untuk mengikuti cerita dan berikan 1 karakter atau lebih untuk dikendalikan, karakter akan diberikan perjalanan dan akan mendapatkan pengalaman dengan mengalahkan musuh.

Contohnya : *Final Fantasy series*, *Fire Emblem.*



Gambar 2.7 Contoh dari *Role-Playing Game*

2.1.4. Sprite

Salah satu faktor yang membuat keberhasilan *game* adalah grafis yang unggul. Sebagian besar objek pada sebuah *game* dikategorikan sebagai grafis khusus yang disebut sprite. Sebuah sprite dapat berupa *bullet, monster,* karakter utama, musuh, kekuatan spesial, kunci dan pintu [18].



Gambar 2.8 contoh sebuah sprite

Pada umumnya *sprite adalah* grafis animasi. Grafis animasi dibuat dari *sprite* yang sama namun berbeda penampakannya. Kumpulan *sprite* biasanya mengacu sebagai sebuah kumpulan *frame*. *Frame* ini dapat dibuat secara terurut untuk memudahkan pengkodean.



Gambar 2.9 Contoh kumpulan beberapa sprite

Ada 3 konstruktor yang disediakan oleh kelas sprite [18]

1. Sprite (Image image) : Membuat frame sprite tunggal, tidak dianimasikan.
2. Sprite (Sprite sprite) : membuat sprite baru dari sprite lainnya.
3. Sprite (Image image, int frameWidth, int frameHeight) : membuat animasi sprite dengan lebih dari 2 frame, frameWidth adalah lebar dari sebuah sprite dan frameHeight adalah tinggi dari sebuah *sprite*.



Gambar 2.10 Contoh kumpulan sprite dengan bingkai di sekelilingnya

Sesuai dengan gambar , kita dapat memecah kumpulan sprite menjadi frame secara tersendiri. Pada contoh ini total lebar kumpulan sprite adalah 160 pixel, yang dibagi dengan 5 sehingga menjadi 5 buah frame dengan lebar masing-masing 32 pixel. Tinggi masing-masing frame adalah 32 pixel. Tinggi dan lebar tidaklah selalu sama, tetapi lebar dan tinggi harus konstan untuk semua sprite pada kumpulan sprite. Dengan kata lain, Anda tidak bisa memiliki sebuah frame dengan lebar 32 pixel dan sprite yang tersisa hanya memiliki lebar 16 pixel, semua fame harus memiliki lebar yang sama. Di dalam konstruktor Sprite (Image image, int frameWidth, int frameHeight) Anda akan diberitahu bahwa Anda tidak dapat menetapkan angka pada frame, hal ini secara otomatis dihitung oleh kelas Sprite [18].

2.1.5. TiledLayer

TiledLayer merupakan elemen visual yang berupa area yang tersusun atas sel-sel berukuran sama, yang ditempati oleh satu set gambar *tile* (*tile image*). Kelas TiledLayer memungkinkan Layer virtual yang besar dibuat tanpa memerlukan gambar yang besar. Teknik ini umum digunakan di dalam pembuatan platform *game* 2D untuk membuat *scrolling background* yang besar [20].

Tile-tile yang digunakan untuk mengisi sel-sel TiledLayer disediakan oleh sebuah obyek Image, yang dibagi atas sekumpulan *tile* dengan ukuran yang sama. Seperti yang terlihat pada gambar berikut, satu set *tile* yang sama dapat disimpan dalam beberapa pengaturan [20].

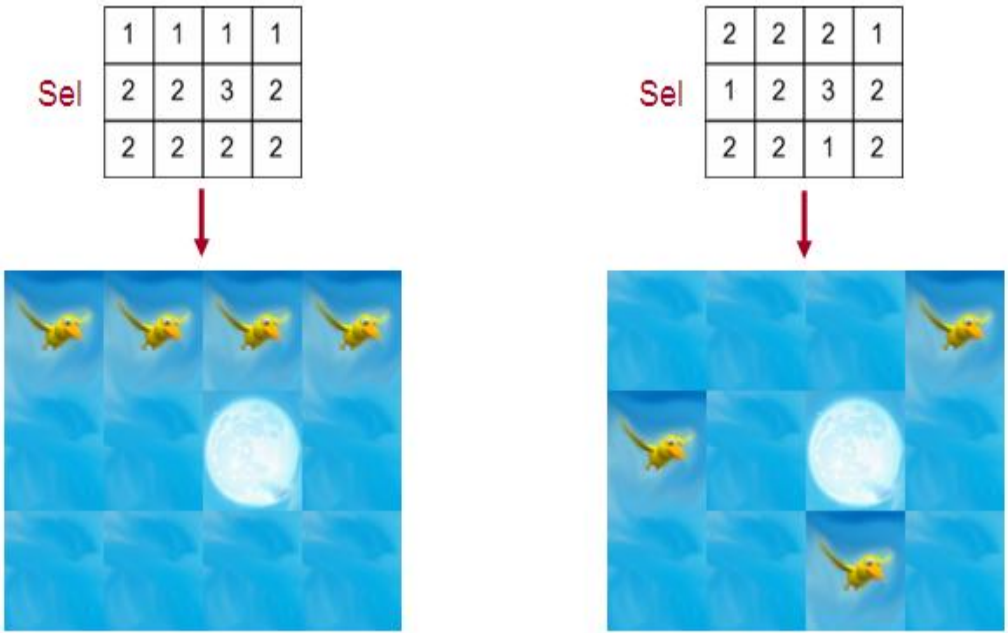


Gambar 2.11 *Set tile* yang sama dapat disimpan dalam beberapa pengaturan

Setiap *tile* diberi indeks yang berbeda. *Tile* yang berada pada sisi kiri atas sebuah obyek Image diberi indeks 1, selebihnya diberi nomor sesuai dengan urutan baris. *Tile-tile* yang dihasilkan ini dinamakan *static tile* karena terdapat link yang tetap antara *tile* dan data obyek Image yang bersesuaian.

TiledLayer akan berfungsi untuk mendefinisikan semua area unik pada background dan menggunakannya kembali sebanyak yang diperlukan untuk menciptakan sebuah gambar yang lengkap. Isi dari setiap sel penyusun *area grid* TiledLayer direferensi oleh nilai indeks *tile*. Nilai indeks positif merujuk ke *static tile*, sedangkan nilai indeks negatif merujuk ke *animated tile*. Nilai indeks 0

mengindikasikan sel yang bersangkutan kosong (transparan), tidak ditempati oleh tile [20].



Gambar 2.12 *Grid sel* merepresentasikan *tile-tile* penyusun gambar

2.1.6. *Collision* Detection

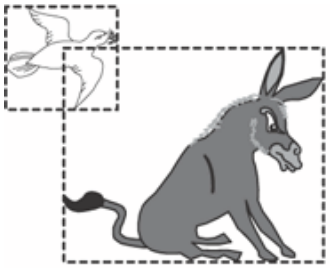
Fungsionalitas *collision detection* merupakan fitur penting bagi aplikasi *game*. Tabrakan atau *collision* yang dimaksud adalah antara dua atau lebih Sprite, TiledLayer atau Image dalam satu tampilan. Terkait dengan ini, *Game* API mendukung dua teknik untuk mendeteksi terjadinya *collision*/tabrakan yaitu [20]:

1. Implementasi dapat membandingkan area-area persegi empat yang merepresentasi-kan sebuah Sprite dengan Sprite, TiledLayer atau Image lainnya. *Collision* terjadi jika area-area persegi empat tersebut beririsan. Teknik ini merupakan cara yang cepat untuk men*collision detection*, tetapi memungkinkan terjadinya ketidaktepatan di dalam men*collision detection* untuk bentuk-bentuk yang tidak persegi empat.
2. Implementasi dapat membandingkan tiap piksel sebuah Sprite dengan Sprite lainnya. Jika sebuah piksel yang tidak transparan (*opaque*) dari sebuah Sprite beririsan dengan sebuah piksel yang tidak transparan (*opaque*) dari Sprite yang lain, *collision* terjadi. Teknik ini memerlukan

komputasi yang lebih banyak, tetapi akan memberikan hasil yang lebih akurat.

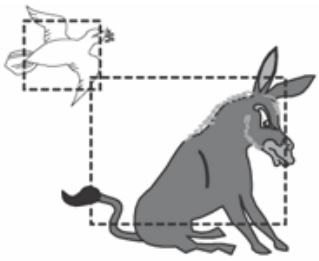
Sprite memiliki *collision rectangle*, yang secara default terletak pada titik (0, 0) dengan lebar dan tinggi yang sama dengan obyek Sprite. Untuk mengganti *collision rectangle*, *method* berikut digunakan fungsi defineCollisionRectangle. *Collision rectangle* memiliki dua peran sebagai berikut [20].

a. Jika *collision detection* berdasarkan piksel (*pixel-level collision detection*) tidak digunakan, *collision rectangle* digunakan sebagai area untuk mendeteksi *collision*.



Gambar 2.13 contoh *collision* detection tidak berdasarkan piksel.

b. Jika *collision detection* berdasarkan piksel (*pixel-level collision detection*) digunakan, maka hanya piksel di dalam *collision rectangle* yang diproses.



Gambar 2.14 contoh *collision* detection berdasarkan piksel.

2.2. Multimedia

Definisi multimedia secara umum adalah penggabungan berbagai infomasi dengan menggunakan fasilitas dari komputer. Multimedia yang berasal dari kata multi yang berarti banyak atau lebih dari satu dan media yang dapat diartikan penyajian suatu tempat. Multimedia adalah pemanfaatan komputer untuk membuat dan menggabungkan teks, *audio*, gambar, bergerak (video dan animasi

dengan menggunakan *link* dan *tool* yang memungkinkan pemakaian melakukan navigasi. berinteraksi, berkreasi dan berkomunikasi [16].

2.2.1. Objek Multimedia

Objek multimedia terbagi menjadi [17] :

1. Teks

   Teks merupakan dasar dari pengolahan kata dan informasi berbasis multimedia. Dalam kenyataannya multimedia menyajikan informasi kepada audiens dengan cepat, karena tidak diperlukan membaca secara rinci dan teliti. Sebagian besar sistem multimedia dirancang dengan menggunakan teks karena teks karena teks merupakan sarana yang efektif untuk mengemukakan ide-ide dan menyediakan instruksi-instruksi kepada *user* (pengguna).

2. Image

   Secara umum image atau grafik berarti still image seperti foto dan gambar. Manusia sangat berorientasi pada visual dan gambar merupakan sarana yang sangat baik untuk menyajikan informasi.

3. Animasi

   Animasi adalah pembentukan gerakan dari berbagai media atau objek yang divariasikan dengan gerakan transisi, efek-efek, juga suara yang selaras dengan gerakan animasi tersebut atau animasi merupakan penayangan frame-frame gambar secara cepat untuk menghasilkan gerakan.

4. Audio

   Penyajian audio atau suara merupakan cara lain untuk lebih memperjelas pengertian suatu informasi. Contohnya, narasi merupakan kelengkapan dari penjelasan yang dilihat melalui video. Suara dapat lebih menjelaskan karakteristik suatu gambar, misalnya musik dan suara efek (*sound effect*). Salah satu bentuk bunyi yang bisa digunakan dalam produksi multimedia adalah *Waveform Audio* yang merupakan format file audio yang berbentuk digital. Kualitas produknya bergantung pada *sampling rate* (banyaknya

sampel per detik). *Waveform* (wav) merupakan standar untuk Windows PC.

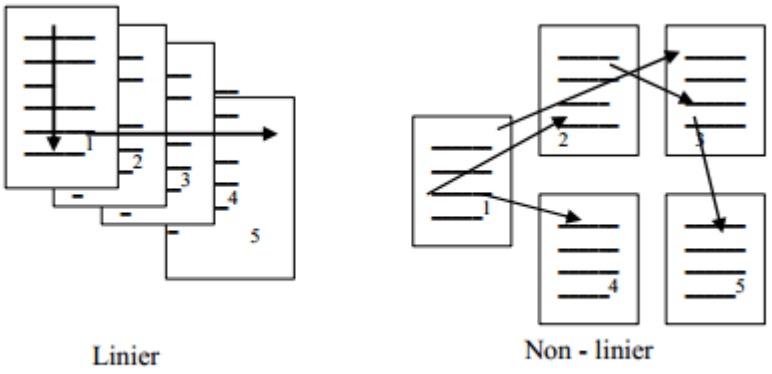
5. Video

   Video merupakan elemen multimedia paling kompleks karena penyampaian informasi yang lebih komunikatif dibandingkan gambar biasa. Walaupun terdiri dari elemen-elemen yang sama seperti grafik, suara, teks, namun bentuk video berbeda dengan animasi. Perbedaaan terletak pada penyajiannya. Dalam video, informasi disajikan dalam kesatuan utuh dari objek yang dimodifikasikan sehingga terlihat saling mendukung pengambaran yang seakan terlihat hidup.

6. *Interactive Link*

   Sebagian dari multimedia adalah interaktif, dimana pengguna dapat menekan mouse atau objek pada screen seperti button atau teks dan menyebabkan program melakukan perintah tertentu.

   Interactive link dengan informasi yang dihubungkannya sering kali dihubungkan secara keseluruhan sebagai *hypermedia*. Secara spesifik, dalam hal ini termasuk *hypertext (hotword)*, *hypergraphics* dan *hypersound* Menjelaskan jenis informasi yang dihubungkan. *Interactive link* diperlukan bila pengguna menunjuk pada suatu objek atau button agar dapat mengakses program tertentu. Interactive link diperlukan untuk menggabungkan beberapa elemen multimedia sehingga menjadi informasi yang terpadu. Cara peng-aksesan informasi pada multimedia terdapat dua macam, yaitu linier dan non-linier. Informasi linier adalah informasi yang ditampilkan secara sekuansial, yaitu dari atas ke bawah atau halaman demi halaman, sedangkan pada informasi non-linier (seperti pada Gambar dibawah) dapat ditampilkan langsung sesuai dengan kehendak pengguna.



Gambar 2.15 ***Interactive link.***

2.2.2. Kategori Multimedia

Multimedia dapat di definisikan menjadi 2 kategori, yaitu [17]**:**

1. Multimedia linier

   Multimedia linier adalah suatu multimedia yang tidak dilengkapi dengan alat pengontrol apapun yang dapat dioperasikan oleh penguna.

   Contohnya: TV dan film.

2. Multimedia Interaktif

   Multimedia interaktif adalah suatu multimedia yang dilengkapi dengan alat pengontrol yang dapat dioperasikan oleh pengguna, sehingga pengguna dapat memilih apa yang dikehendaki untuk proses selanjutnya.

   Contoh multimedia interaktif adalah: multimedia pembelajaran interaktif, aplikasi game, dan lain-lain.

2.3. AI (Artficial Intelligence)

AI atau kecerdasan buatan adalah salah satu bagian ilmu komputer yang membuat agar mesin (komputer) dapat melakukan pekerjaan seperti dan sebaik yang dilakukan oleh manusia. Lebih detailnya pengertian AI 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

   AI adalah suatu studi bagaimana membuat agar komputer dapat melakukan sesuatu sebaik yang dikerjakan oleh manusia.

   Domain yang sering dibahas :

   1. *Mundane task* adalah kegiatan duniawi yang biasa dilakukan manusia.
      - Persepsi
      - Bahasa alam
      - Robot control
   2. *Formal task* adalah meliputi permainan dan matematika yang dalam pelaksanaannya memiliki aturan-aturan yang harus dipatuhi.
      - Permainan
      - Matematika
   3. *Expert Task* adalah kegiatan-kegiatan yang meliputi bidang-bidang profesional.
      - Analisis finansial
      - Analisis mudikal
      - Analisis ilmu pengetahuan

3. Bisnis

   AI adalah kumpulan peralatan yang sangat powerful dan metodologis dalam menyelesaikan masalah-masalah bisnis.

4. Pemrograman

   Studi tentang pemrograman simbolik, penyelesaian masalah dan pencarian. Untuk melakukan aplikasi AI ada 2 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 [2].

Definisi diatas belum seluruhnya benar karena pada kenyataannya istilah kecerdasan sendiri belum bisa dipahami atau dideskripsikan dengan sangat baik. Kerap kali kita meyakini telah mengetahui tingkah laku cerdas ketika kita melihatnya sendiri. Akan tetapi keraguan juga akan muncul ketika orang lain mengatakan kecerdasan dalam cara khusus menurut cara mereka yang cukup membantu dalam evaluasi kecerdasan buatan pada program komputer.

2.3.1. Perbandingan Kecerdasan Buatan dengan Kecerdasan Alami

Ada beberapa keuntungan kecerdasan buatan dengan kecerdasan alamiah diantaranya ialah:

1. Kecerdasan buatan lebih bersifat permanen tidak seperti kecerdasan alami yang akan cepat mengalami perubahan karena sifat manusia yang pelupa
2. Kecerdasan buatan lebih mudah diduplikasi dan disebarkan tidak seperti manusia karena mentransfer pengetahuan manusia dari satu orang ke orang lain membutuhkan proses yang lama dan juga suatu keahlian itu tidak akan pernah dapat diduplikasi dengan lengkap.
3. Kecerdasan buatan lebih murah dibandingkan kecerdasan alami.
4. Kecerdasan buatan bersifat konsisten.
5. Kecerdasan buatan dapat didokumentasi.
6. Kecerdasan buatan akan lebih cepat mengerjakan pekerjaan dibanding dengan kecerdasan alami.
7. Kecerdasan buatan dapat mengerjakan pekerjaan lebih baik dibanding dengan kecerdasan alami [2].

2.3.2. Komputasi Kecerdasan Buatan dan Komputasi Pemrograman Konvensional

Berikut adalah tabel perbedaan kecerdasan buatan dengan pemrograman konvensional.

Tabel 2.1 Perbedaan Kecerdasan buatan dengan pemrograman konvensional

Dimensi	Kecerdasan Buatan	Pemrograman Konvensional
Sifat Input	Bisa tidak lengkap	Harus lengkap
Pencarian	Kebanyakan bersifat Heuristik	Biasanya didasarkan pada algoritma
Keterangan	Disediakan	Biasanya tidak disediakan
Fokus	Pengetahuan	Data dan Informasi
Struktur	Kontrol dipisahkan dengan pengetahuan	Kontrol terintegrasi dengan informasi
Sifat Output	Kuantitatif	Kualitatif
Pemeliharaan dan Update	Relatif mudah	Sulit
Kemampuan menalar	Ya	Tidak
Pemrosesan	Mengandung konsep-konsep simbolik	Algoritmik

2.3.3. Algoritma A*

Algoritma A* (*A Star*) adalah algoritma pencarian yang merupakan pengembangan dari algoritma *Best First Search* (BFS). Seperti halnya pada BFS, untuk menemukan solusi, A* juga ‘dituntun’ oleh fungsi heuristik, yang menentukan urutan titik mana yang akan dikunjungi terlebih dahulu. Heuristik merupakan penilai yang memberi harga pada tiap verteks yang memandu A* mendapatkan solusi yang diinginkan.

Algoritma ini pertama kali diperkenalkan pada 1968 oleh Peter Hart, Nils Nilsson, dan Bertram Raphael Dalam tulisan mereka, algoritma ini dinamakan algoritma A. Dengan penggunaan fungsi heuristik yang tepat pada algoritma ini yang dapat memberikan hasil yang optimal, maka algoritma inipun disebut A*.

Fungsi heuristik Algoritma ini membangkitkan verteks yang paling mendekati solusi. Verteks ini kemudian disimpan suksesornya ke dalam *list* sesuai dengan urutan yang paling mendekati solusi terbaik. Kemudian, verteks pertama

pada *list* diambil, dibangkitkan suksesornya dan kemudian suksesor ini disimpan ke dalam *list* sesuai dengan urutan yang terbaik untuk solusi. *List* verteks ini disebut dengan verteks terbuka *(open node)* [8].

Pencarian menggunakan algoritma A* mempunyai prinsip yang sama dengan algoritma BFS, hanya saja dengan dua faktor tambahan.

1. Setiap sisi mempunyai "*cost*" yang berbeda-beda, seberapa besar *cost* untuk pergi dari satu simpul ke simpul yang lain.

2. *Cost* dari setiap simpul ke simpul tujuan bisa diperkirakan. Ini membantu pencarian, sehingga lebih kecil kemungkinan kita mencari ke arah yang salah.

Cost untuk setiap simpul tidak harus berupa jarak. *Cost* bisa saja berupa waktu bila kita ingin mencari jalan dengan waktu tercepat untuk dilalui. Sebagai contoh, bila kita berkendaraan melewati jalan biasa bisa saja merupakan jarak terdekat, tetapi melewati jalan tol biasanya memakan waktu lebih sedikit.

Cost antara simpul adalah jaraknya, dan perkiraan *cost* dari suatu simpul ke simpul tujuan adalah penjumlahan jarak dari simpul tersebut ke simpul tujuan. Atau agar lebih mudahnya bisa ditunjukkan seperti berikut ini.

$$f(n) = g(n) + h(n) \qquad (2.1)$$

dengan :

$f(n)$ = fungsi evaluasi

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

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

Node dengan nilai terendah merupakan solusi terbaik untuk diperiksa pertama kali pada $g(n) + h(n)$. Dengan fungsi *heuristic* yang memenuhi kondisi tersebut, maka pencarian dengan algoritma A* dapat optimal [9].

2.3.3.1. Kompleksitas Algoritma A*

Kompleksitas waktu dari algoritma A* tergantung dari *heuristic*-nya. Dalam kasus terburuk (*worst case*), jumlah simpul yang diekspansi bisa eksponensial dalam solusinya (jalan tependek). Akan tetapi, kompleksitasnya bisa berupa polinomial bila fungsi heuristik *h* bertemu kondisi berikut:

$$| h(x) - h(x) | = O(\log h(x)) \tag{2.2}$$

Dimana *h** adalah heuristik optimal, atau *cost* pasti untuk menuju tujuan dari *x*. Dengan kata lain, kesalahan (*error*) dari *h* tidak boleh tumbuh lebih cepat dari algoritma "*perfect heuristic*" *h** yang mengembalikan jarak sebenarnya dari *x* menuju tujuan [10].

2.3.3.2. Efisiensi Waktu algoritma A*

Dengan digunakannya fungsi *heuristic* H(n), algoritma A* dapat memfokuskan pencarian pada *node-node* yang berada pada arah yang mendekati *node* tujuan. Kemudian pencarian diterminasikan pada waktu *node* tujuan diperiksa. Hal ini dapat meminimalisasikan jumlah *node* yang harus diperiksa dan arena waktu yang diperlukan untuk mendapatkan jalur berbanding lurus dengan jumlah *node* yang diperiksa, maka waktu pencarian dapat diminimalisasikan.

Walaupun jumlah *node* yang diperiksa dapat diminimalisasikan, algoritma A* mempunyai kasus terburuk. Pada kasus ini, sebagian besar ataupun keseluruhan *node* pada jalan diperiksa, sehingga algoritma A* bekerja seperti algoritma dijkstra atau BFS *(Best-First-Search).* Ada dua hal yang dapat menyebabkan keadaan terburuk ini, yaitu keadaan sepadan dan jika jalur yang dicari tidak ditemukan [14].

2.3.4. PathFinding

Pathfinding (pencarian jalan/*rute*) adalah salah satu bidang penerapan yang sering ditangani oleh kecerdasan buatan khususnya dengan menggunakan algoritma pencarian. Penerapan yang dapat dilakukan dengan *pathfinding* antara

lain adalah pencarian *rute* dalam suatu *game* dan pencarian jalan/*rute* pada suatu peta. Algoritma pencarian yang dipakai harus dapat mengenali jalan dan elemen peta yang tidak dapat dilewati.

Bagaimana *Pathfinding* ini bekerja? Pertama-tama peta permainan harus diolah atau diproses terlebih dahulu sebelum algoritma A * dapat bekerja. Hal ini akan melibatkan pemutusan peta ke titik yang berbeda atau lokasi, yang disebut *node*. *Node* ini digunakan untuk mencatat kemajuan pencarian. Selain memegang peta setiap *node* memiliki tiga atribut lainnya yaitu fungsi , tujuan dan heuristik yang umum dikenal sebagai f, g,dan h. Nilai yang berbeda dapat diberikan ke jalur antara *node*, biasanya nilai-nilai ini akan mewakili jarak antara *node*. Atribut g, h, dan f didefinisikan sebagai berikut:

1. g adalah biaya yang didapatkan dari *node* awal ke *node* saat ini yaitu jumlah dari semua nilai di jalan antara awal dan *node* saat ini.
2. h singkatan heuristik yang merupakan perkiraan biaya dari *node* saat ini ke tujuan *node* (biasanya jarak garis lurus dari *node* ini ke tujuan).
3. f adalah jumlah dari g dan h dan merupakan estimasi terbaik dari biaya jalan akan melalui *node* saat ini. Intinya semakin rendah nilai f yang akan lebih efisien.

Tujuan dari f, g, dan h adalah untuk menghitung berapa jarak menjanjikan melalui jalan yang sampai ke *node* ini. Selain itu A * memelihara dua yaitu daftar *Open* dan daftar Close. Daftar *open* berisi semua *node* dalam peta yang belum sepenuhnya dieksplorasi lagi, sedangkan daftar *close* terdiri dari semua *node* yang telah sepenuhnya dieksplorasi.

Masalah utama yang timbul dalam *pathfinding* adalah pada saat sebelum proses, yang membuat *pathfinding* kompleks secara real-time. Masalah-masalah ini meliputi ketidakmampuan *pathfinding* untuk menangani dunia yang paling dinamis dan menghasilkan pergerakan yang realistis. Namun jika suatu hambatan yang dinamis kemudian mencakup *node* sepanjang jalur yang telah ditentukan, agen masih akan dipercaya untuk berjalan dimana objek itu berada. Masalah lain

adalah gerakan realistis yang muncul ketika agen berjalan dalam garis lurus antara *node* pada *path*. Hal ini disebabkan oleh dilema yang muncul dalam *trade off* antara kecepatan dan gerakan realistis. Ini telah diperbaiki dalam beberapa permainan dengan menerapkan *splines* (kurva yang paling cocok) antara *node* yang berbeda untuk merapikan jalan.

Masalah yang tercantum di atas adalah karena pengenalan objek dinamis ke dalam peta statis, adalah salah satu fokus penelitian di industri *game* saat ini. Banyak upaya untuk meningkatkan kemampuan reaktif agen AI ketika objek dinamis melalui jalur kotor. Salah satu solusi yang berfokus pada agen memberikan metode dengan memperhatikan sekitarnya. Cara mudah untuk mencapai hal ini adalah dengan cara memberikan agen sebuah sensor sederhana sehingga dipandu oleh *pathfinder* tapi tidak sepenuhnya dikendalikan olehnya. Namun metode ini tidak akan efektif jika sensor yang digunakan tidak dapat menangani data yang kurang baik [11].

2.4. *Tools* yang Digunakan

Tools yang digunakan untuk merancang dan membangun *game* Crazy Becak ini, baik itu untuk pemrograman maupun analisis.

2.4.1. OOP (Object Oriented Programming)

Object Oriented Programming (OOP) menerapkan sifat yang lebih modular agar setiap program dapat lebih mudah dikembangkan. Dalam OOP dibutuhkan memori lebih besar dibandingkan dengan program procedural (tradisional)[5]. Dua objek yang identik akan memerlukan dua area memori berbeda walaupun dari sisi data dan proses keduanya memiliki jumlah dan jenis yang sama. Hal ini disebabkan karena data dan proses pada kedua objek tersebut dipisahkan oleh komputer.

Secara garis besar yang menjadi ciri dari OOP adalah adanya proses abstraksi (*abstraction*), pengkapsulan (*encapsulation),* penurunan sifat (*inheritance*), dan polimorfisme (*polymorphism*) pada objek-objek yang dibentuk.

Object Oriented Programming (OOP) dibagi menjadi beberapa cirri utama[5], yaitu :

A. **Kelas**

Kelas (*class*) merupakan contoh abstrak dari sebuah objek yang telah terbentuk dari proses penyederhanaan, dengan kata lain kelas (*class*) merupakan cikal bakal dari objek (*object*), kemudian contoh nyata atau perwujudan dari sebuah objek dinamakan *instance*. Sehingga apabila kita mempunyai sebuah kelas manusia, maka beberapa *instances* (wujud nyata) dari kelas manusia adalah Prima, Aulia, Dewi, dan masih banyak yang lainnya.

Perbedaan antara kelas (*class*) dengan objek (*object*) dalam OOP dibagi menjadi dua[1], yaitu :

1. *Class* merupakan rancangan (*design*) dan *object* merupakan perwujudan dari suatu *class.*
2. *Class* bersifat abstrak sementara *object* bersifat konkrit (atau nyata).

Di dalam kelas, terdapat *access modifier, class modifier,* dan *class body*, dibawah ini merupakan penjelasan tentang isi dari sebuah kelas.

1. *Access Modifier*

Access modifier merupakan *keyword* yang digunakan untuk menentukan spesifikasi tingkat akses suatu variable atau *method* (anggota kelas). Pengaksesan yang dimaksud bisa berupa pengaksesan dalam kelas yang sama, turunan, maupun di luar kelas di mana variable dan *method* dideklarasikan. Dalam *Java*, *access modifier* terdiri atas :

a. *Default*

   Access modifier ini hanya menspesifikasikan kelas-kelas (*classes*) di paket yang sama yang bisa mengakses variable dan *method* suatu kelas. Anggota kelas dengan *default access* bisa melihat kelas lain di paket yang sama. Apabila tidak ada *access modifier* pada suatu pendeklarasian kelas, berarti yang dimaksud adalah *default access*.

b. *Public*

Apabila suatu anggota kelas (*class member*) dideklarasikan sebagai *public*, maka anggota tersebut akan bisa digunakan oleh siapa saja, yaitu baik oleh kelas yang sama, kelas lain, maupun lingkungan luar kelas.

c. *Protected*

Access modifier protected menspesifikasikan atau menentukan anggota kelas (*class member*) yang hanya bisa diakses oleh *method-method* yang ada di dalam kelas yang sama dan subkelas-subkelas turunan dari kelas tersebut.

d. *Private*

Access modifier private merupakan tingkatan akses yang sangat terbatas. Pada keadaan ini tidak ada kelas lain yang dapat mengakses anggota kelas *private*, sekaligus termasuk subkelas-subkelasnya.

2. *Class Modifier*

a. *Public*

Apabila suatu kelas (*class*) dideklarasikan sebagai *public*, maka kelas tersebut akan bisa digunakan oleh kelas lain maupun (termasuk dari lingkungan luar) tanpa memperhatikan apakah kelas lain tersebut berasal dari paket (*package*) yang sama atau tidak.

b. Abstract

Apabila suatu kelas (*class*) dideklarasikan sebagai *abstract*, maka kelas tersebut tidak akan bisa diwujudkan menjadi suatu objek. Kelas ini disusun dari satu atau lebih metode abstrak, artinya *method-method* dalam kelas ini dideklarasikan akan tetapi tanpa badan (tidak d2mplementasikan).

c. Final

Apabila suatu kelas (*class*) dideklarasikan sebagai *final*, maka kelas tersebut tidak akan bisa diturunkan lagi menjadi kelas turunan yang lain. Dengan kata lain sudah selesai atau berakhir (*final*). Tujuannya adalah untuk mencegah perluasan yang tidak diinginkan.

d. Extends

Class modifier extends digunakan untuk mendeklarasikan kelas turunan dari kelas yang lainnya.

e. Implements

Class modifier implements digunakan untuk mendeklarasikan sebuah kelas yang akan mengimplementasikan satu atau lebih *interface Java*. Apabila *interface Java* yang akan d2mplementasikan lebih dari satu, maka pendeklarasiannya dipisahkan dengan tanda koma “,”.

B. Objek

Dalam kenyataannya, sebuah objek dalam OOP adalah sebuah persilangan yang berbagi-pakai (*share*) sejumlah ciri dari objek umum dengan fitur (*feature*) dari sebuah bentuk komputer[1].

Sebuah objek secara praktis pemrograman berorientasi objek bisa didefinisikan sebagai berikut :

1. Setiap objek dimiliki oleh kelas objek, sehingga sebuah objek tidak bisa hadir tanpa sebuah kelas yang mendefinisikannya. Dengan kata lain objek adalah wujud (*instance*) dari sebuah kelas.
2. Sebuah objek (dan kelas yang memuatnya) adalah sebuah pengkapsulan (*encapsulation*) yang memasukkan data dan operasi untuk pemrosesannya.
3. Atribut-atribut (*attributes*) objek membantu untuk menyimpan dan menjaga status objek. Atribut-atribut ini menentukan apa yang d2ngat mengenai objek. Methode objek adalah satu-satunya cara untuk mengakses data dan memodifikasi statusnya. Cara pengaksesan dan pemodifikasian data dilakukan dengan mengirimkan sebuah pesan ke objek tersebut.

C. Abstraksi

Abstraksi dapat didefinisikan sebagai suatu proses melakukan desain *class* dan menentukan data dan *method* yang akan dimiliki oleh sebuah *class*[1].

Sebuah *method* abstrak mendefinisikan sebuah antarmuka dalam kelas dasar dan meninggalkan implementasi pada kelas turunan. Kelas abstrak adalah sebuah kelas yang berisi satu atau beberapa *method* abstrak.

D. Pengkapsulan

Pengkapsulan (*encapsulation*) merupakan proses pembungkusan atau penyederhanaan dari beberapa data atau *method* menjadi sebuah objek (*object*) atau kelas (*class*)[1].

E. Pewarisan Sifat

Penurunan atau pewarisan sifat (*inheritance*) ini merupakan cirri utama dari OOP dimana sifat-sifat yang terdapat pada kelas induk (*base class*) akan dimiliki oleh kelas turunannya (*derived class*)[1].

2.4.2. UML (Unified Modeling Language)

Pada perkembangan teknik pemrograman berorientasi objek, muncul sebuah standarisasi bahasa pemodelan untuk pembangunan perangkat lunak yang dibangun dengan menggunakan teknik pemrograman berorientasi objek, yaitu *Unified Modelling* Language (UML). Adapun pengertian dari UML adalah salah satu standar bahasa yang banyak digunakan di dunia industri untuk mendefinisikan *requirement,* membuat analisis dan desain, serta menggambarkan arsitektur dalam pemrograman berorientasi objek [19].

UML muncul karena adanya kebutuhan pemodelan visual untuk menspesifikasikan, menggambarkan, membangun, dan dokumentasi dari sistem perangkat lunak. Dalam hal ini UML merupakan suatu bahasa visual untuk melakukan pemodelan dan komunikasi mengenai sebuah sistem dengan menggunakan diagram dan teks-teks pendukung.

2.4.2.1. Diagram UML

UML menggunakan berbagai macam diagram dengan fungsi masing-masing untuk menggambarkan setiap proses dari sistem berorientasi objek. Berikut merupakan beberapa diagram UML diantaranya[19] :

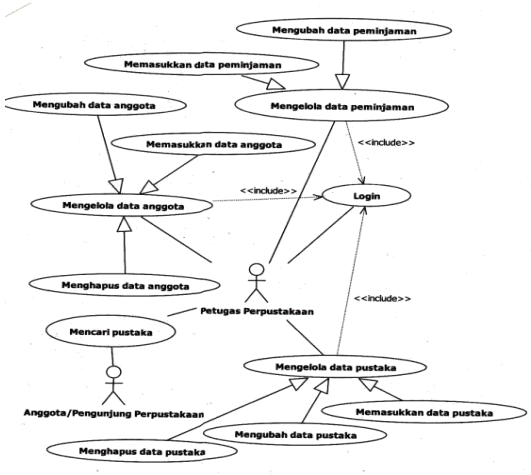
A. ***Use Case Diagram***

Use Case atau diagram *use case* merupakan pemodelan yang digunakan untuk menggambarkan kelakuan (*behavior)* dari sistem yang akan dibuat [19]. *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*[19].

1. Aktor merupakan orang, proses, atau sistem lain yang berinteraksi dengan sistem yang akan dibuat diluar sistem yang akan dibuat itu sendiri, jadi walaupun simbol dari aktor adalah gambar orang, tapi aktor belum tentu merupakan orang.
2. *Use case* merupakan fungsionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesar antarunit atau aktor.

Contoh dari *use case diagram* dapat dilihat pada gambar 2.13.



Gambar 2.16 Contoh dari *Use Case Diagram*

B. ***Activity Diagram***

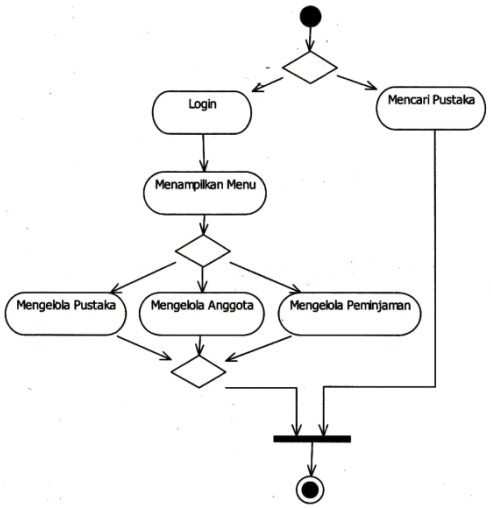
Diagram aktivitas atau *activity diagram* adalah sebuah diagram yang menggambarkan *workflow* (aliran kerja) atau aktivitas dari sebuah sistem atau proses bisnis [19]. Dalam diagram aktivitas yang perlu diperhatikan adalah bahwa

diagram aktivitas menggambarkan aktivitas sistem, bukan apa yang dilakukan aktor, jadi aktivitas yang dapat dilakukan oleh sistem.

Diagram aktivitas juga banyak digunakan untuk mendefinisikan hal-hal berikut [19] :

1. Rancangan proses bisnis di mana setiap urutan aktivitas yang digambarkan merupakan proses bisnis sistem yang didefinisikan.
2. Urutan aau pengelompokan tampilan dari sistem/*user interface* di mana setiap aktivitas dianggap memiliki sebuah rancangan antarmuka tampilan.
3. Rancangan pengunjian di mana setiap aktivitas dianggap memerlukan sebuah pengujian yang perlu didefinisikan kasus ujinya.

Contoh dari *activity diagram* dapat dilihat pada gambar 2.14.



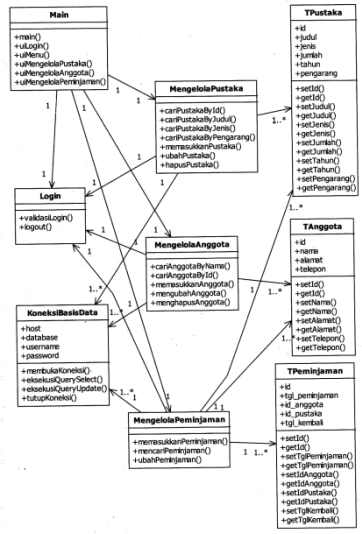
Gambar 2.17 Contoh dari *Activity Diagram*

C. ***Class Diagram***

Diagram kelas atau *class diagram* menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang akan dibuat untuk membangun sistem. Kelas memiliki apa yang disebut atribut dan metode atau operasi [19].

1. Atribut merupakan variabel-variabel yang dimiliki oleh suatu kelas
2. Operasi atau metode adalah fungsi-fungsi yang dimiliki oleh suatu kelas

Contoh dari *class diagram* dapat dilihat pada gambar 2.15.



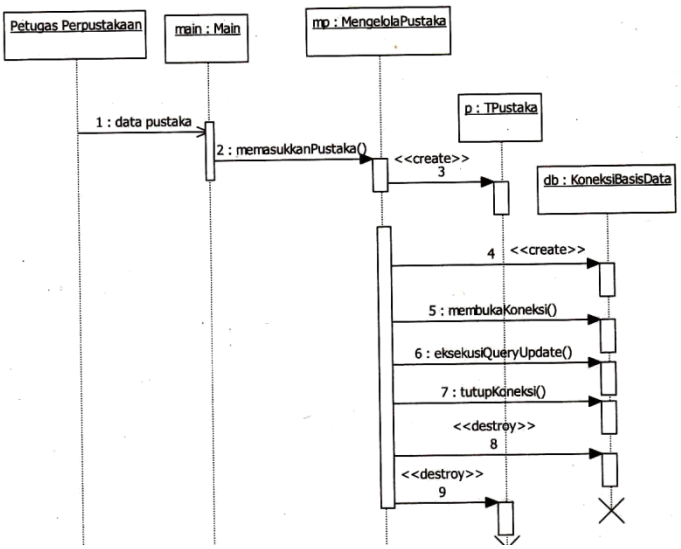
Gambar 2.18 Contoh dari *Class Diagram*

D. ***Sequence Diagram***

Diagram sekuen adalah diagram yang menggambarkan kelakuan objek pada *use case* dengan mendeskripsikan waktu hidup objek dan *message* yang dikirimkan dan diterima antarobjek [19]. Oleh karena itu untuk menggambarkan diagram sekuen maka harus diketahui objek-objek yang terlibat dalam sebuah *use case* beserta metode-metode yang dimiliki kelas yang d2nstansiasi menjadi objek itu.

Banyaknya diagram sekuen yang harus digambarkan adalah sebanyak pendefinisian *use case* yang memiliki prose situ sendiri atau yang penting semua *use case* yang telah didefinisikan interaksi jalannya pesan sudah dicakup pada diagram sekuen sehingga semakin banyak *use case* yang didefinisikan maka diagram sekuen yang harus dibuat juga semakin banyak.

Contoh dari *sequence diagram* dapat dilihat pada gambar 2.16.



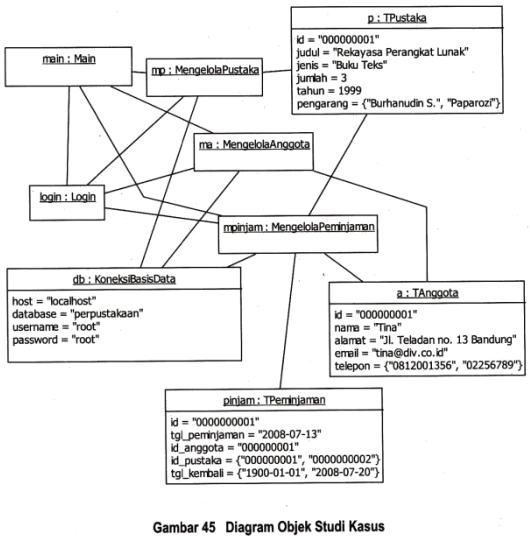
Gambar 2.19 Contoh dari *Sequence Diagram*

E. ***Object Diagram***

Diagram objek menggambarkan struktur system dari segi penamaan objek dan jalannya objek dalam sistem [19]. Pada diagram objek harus dipastikan semua kelas yang sudah didefinisikan pada diagram kelas harus dipakai objeknya, karena jika tidak, pendefinisian kelas itu tidak dapat dipertanggungjawabkan.

Untuk apa mendefinisikan sebuah kelas sedangkan pada jalannya sistem, objeknya tidak pernah dipakai. Hubungan *link* pada diagram objek merupakan hubungan memakai dan dipakai di mana dua buah objek akan dihubungkan oleh *link* jika ada objek yang dipakai oleh objek lainnya.

Contoh dari *object diagram* dapat dilihat pada gambar 2.17.



Gambar 45 Diagram Objek Studi Kasus

Gambar 2.20 Contoh dari *Object Diagram*

F. *Component Diagram*

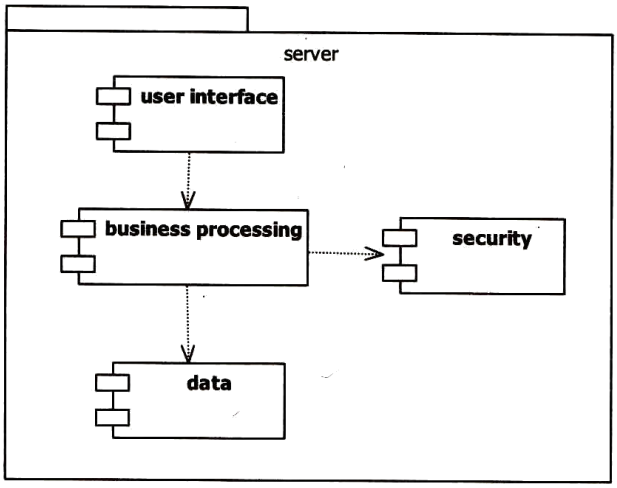
Diagram komponen dibuat untuk menunjukkan organisasi dan kebergantungan di antara kumpulan komponen dalam sebuah sistem. Diagram komponen focus pada komponen sistem yang dibutuhkan dan ada di dalam sistem [19]. Diagram komponen juga dapat digunakan untuk memodelkan hal-hal berikut [19] :

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

Adapun komponen-komponen dasar yang biasanya ada dalam suatu sistem adalah sebagai berikut[19] :

1. Komponen *user interface* yang menangani tampilan
2. Komponen *business processing* yang menangani fungsi-fungsi proses bisnis
3. Komponen *data* yang menangani manipulasi data
4. Komponen *security* yang menangani keamanan sistem

Contoh dari *component diagram* dapat dilihat pada gambar 2.18.



Gambar 2.21 Contoh dari *Component Diagram*

G. ***Composite Structure Diagram***

Diagram ini dapat digunakan untuk menggambarkan struktur dari bagian-bagian yang saling terhubung maupun mendeskripsikan struktur pada saat berjalan (*runtime*) dari *instance* yang saling terhubung [19]. Contoh penggunaan diagram ini misalnya untuk menggambarkan deskripsi dari setiap bagian mesin yang saling terkait *router* pada jaringan komputer, dll.

H. ***Package Diagram***

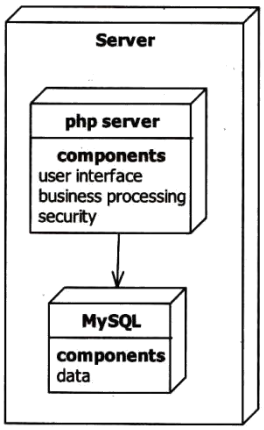
Package diagram menyediakan cara mengumpulkan elemen-elemen yang saling terkait dalam diagram UML [19].

I. ***Deployment Diagram***

Diagram deployment atau *deployment diagram* menunjukkan konfigurasi komponen dalam proses eksekusi aplikasi [19]. Diagram *deployment* juga dapat digunkan untuk memodelkan hal-hal berikut :

1. Sistem tambahan (*embedded system*) yang menggambarkan rancangan *device, node,* dan selanjutnya
2. Sistem *client/server*
3. Sistem terdistribusi murni
4. Rekayasa ulang aplikasi

Contoh dari *deployment diagram* dapat dilihat pada gambar 2.19.



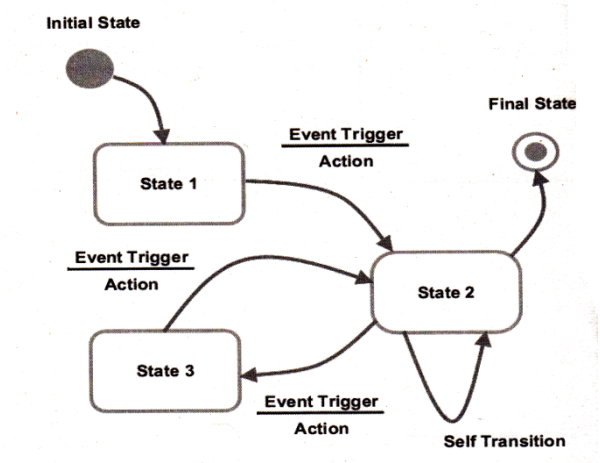
Gambar 2.22 Contoh dari *Deployment Diagram*

J. ***State Machine Diagram***

Diagram mesin status digunakan untuk menggambarkan perubahan status atau transisi status dari sebuah mesin atau sistem [19]. Perubahan tersebut digambarkan dalam suatu graf berarah. *State machine diagram* merupakan pengembangan dari diagram *Finite State Automata* dengan penambahan beberapa fitur dan konsep baru.

Diagram ini cocok digunakan untuk menggambarkan alur interaksi pengguna dengan sistem [19].

Contoh dari *state machine diagram* dapat dilihat pada gambar 2.20.



Gambar 2.23 Contoh dari *State Machine Diagram*

K. *Communication Diagram*

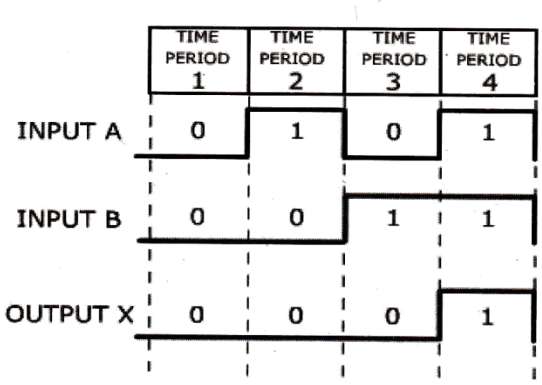
Diagram komunikasi merupakan penyederhanaan dari diagram kolaborasi (*collaboration diagram*) [19]. Diagram ini menggambarkan interaksi antarobjek/bagian dalam bentuk urutan pengiriman pesan. Diagram komunikasi merepresentasikan informasi yang diperoleh dari Diagram Kelas, Diagram Sekuen, dan Diagram *Use Case* untuk mendeskripsikan gabungan antara struktur statis dan tingkah laku dinamis dari suatu sistem.

Diagram komunikasi mengelompokkan *message* pada kumpulan diagram sekuen menjadi sebuah diagram. Dalam diagram komunikasi yang dituliskan adalah operasi/metode yang dijalankan antara objek yang satu dan objek yang lainnya secara keseluruhan, oleh karena itu dapat diambil dari jalannya interaksi pada semua diagram sekuen. Penomoran metode dapat dilakukan berdasarkan urutan dijalankannya metode/operasi di antara objek yang satu dengan objek yang lainnya atau objek itu sendiri.

L. *Timing Diagram*

Timing Diagram merupakan diagram yang focus pada penggambaran terkait batasan waktu. *Timing diagram* digunakan untuk menggambarkan tingkah laku sistem dalam periode waktu tertentu [19]. *Timing diagram* biasanya digunakan untuk mendeskripsikan operasi dari alat *digital* karena penggambaran secara visual akan lebih mudah dipahami daripada dengan kata-kata. Aliran waktu pada *timing diagram* dibaca dari kiri ke kanan.

Contoh dari *timing diagram* dapat dilihat pada gambar 2.15.



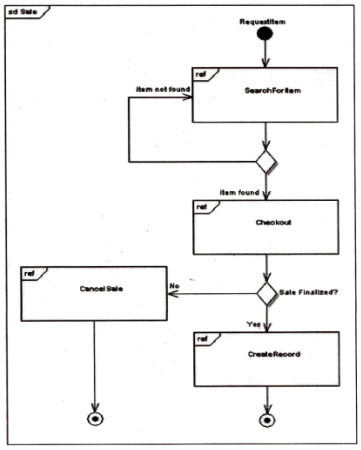
Gambar 2.24 Contoh dari *Timing Diagram*

M. ***Iteraction Overview Diagram***

Iteraction overview diagram mirip dengan diagram aktivitas yang berfungsi untuk menggambarkan sekumpulan urutan aktivitas. *Iterraction overview diagram* adalah bentuk aktivitas diagram yang setiap titik merepresentasikan diagram interaksi[19]. Interaksi diagram dapat meliputi diagram sekuen, diagram komunikasi, *interaction overview diagram*, dan *timing diagram*.

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

Contoh dari *overview diagram* dapat dilihat pada gambar 2.16.



Gambar 2.25 Contoh dari *Iteraction Overview Diagram*

2.4.3. Java

Java adalah bahasa pemrograman yang berorientasi objek dan dapat dijalankan di berbagai *platform* sistem operasi termasuk telepon genggam. Perkembangan java tidak hanya terfokus pada satu sistem operasi, tetapi dikembangkan untuk berbagai sistem operasi dan bersifat *open source* [12].

Bahasa ini awalnya dibuat oleh James Gosling saat masih bergabung di *Sun Microsystems* saat ini merupakan bagian dari Oracle dan dirilis tahun 1995.

Pada mulanya James Gosling ingin memberi nama OAK dari pohon yang terdapat pada kantor James Gosling, tetapi kata OAK telah ada pada *Sun Microsystem* maka diberi nama Java (dari inspirasi minum kopi). Versi pertama java, dirilis pada tahun 1996, dengan sebutan JDK 1.1 (Java Development Kit versi 1.1). Kemudian muncul Java 2 yag dilengkapi dengan *swing*, yaitu teknologi GUI yang dapat menghasilkan aplikasi *desktop* [13].

Sun Microsystems telah mendefinisikan tiga buah edisi dari Java 2, yaitu sebagai berikut [6]:

1. *Java 2 Standard Edition* (J2SE)
   Inti dari bahasa pemrograman Java. J2SE yang digunakan untuk mengembangkan aplikasi-aplikasi *desktop* dan *applet.* Java *applet* adalah sebuah program kecil yang ditulis dengan menggunakan bahasa pemrograman Java, yang diakses melalui halaman web dan dapat di-*download* ke dalam mesin *client* yang kemudian menjalankannya di dalam jendela *web-browser.*

2. *Java 2 Enterprise Edition* (J2EE)
   Merupakan superset dari J2SE yang memperbolehkan kita untuk mengembangkan aplikasi-aplikasi berskala besar (*enterprise*) karena dijalankan pada jaringan komputer.

3. *Java 2 Micro Edition* (J2ME)
   Merupakan subset dari J2SE yang yang ditujuka untuk implementasi pada peralatan *embedded system* dan *mobile devices* yang tidak mampu mendukung secara penuh implementasi menggunakan J2SE. *Embedded system* adalah produk-produk dengan komputer kecil berada di dalamnya, namun aplikasi yang bisa dimanfaatkan dari peralatan tersebut sangatlah spesifik.

Berdasarkan *white paper* resmi dari SUN, Java memiliki karakteristik berikut [12]:

1. Sederhana

   Bahasa pemrograman Java menggunakan sintaks mirip dengan C++ namun sintaks pada Java telah banyak diperbaiki terutama menghilangkan penggunaan *pointer* yang rumit dan *multiple inheritance*. Java juga menggunakan *automatic memory allocation* dan *memory garbage collection*.

2. Berorientasi Objek

   Java mengunakan pemrograman berorientasi objek yang membuat program dapat dibuat secara modular dan dapat dipergunakan kembali. Pemrograman berorientasi objek memodelkan dunia nyata kedalam objek dan melakukan interaksi antar objek-objek tersebut.

3. Dapat didistribusi dengan mudah Java dibuat untuk membuat aplikasi terdistribusi secara mudah dengan adanya *libraries* networking yang terintegrasi pada Java.
4. *Interpreter*

   Program Java dijalankan menggunakan interpreter yaitu *Java Virtual Machine* (JVM). Hal ini menyebabkan *source code* Java yang telah dikompilasi menjadi Java *bytecodes* dapat dijalankan pada platform yang berbeda-beda.

5. *Robust*

   Java mempuyai reliabilitas yang tinggi. *Compiler* pada Java mempunyai kemampuan mendeteksi *error* secara lebih teliti dibandingkan bahasa pemrograman lain. Java mempunyai *runtime-Exception handling* untuk membantu mengatasi *error* pada pemrograman.

6. Aman

   Sebagai bahasa pemrograman untuk aplikasi internet dan terdistribusi, Java memiliki beberapa mekanisme keamanan untuk menjaga aplikasi tidak digunakan untuk merusak sistem komputer yang menjalankan aplikasi tersebut.

7. *Architecture Neutral*

   Program Java merupakan *platform independent*. Program cukup mempunyai satu buah versi yang dapat dijalankan pada platform yang berbeda dengan *Java Virtual Machine*.

8. Portabel

   Source code maupun program Java dapat dengan mudah dibawa ke *platform* yang berbeda-beda tanpa harus dikompilasi ulang.

9. *Performance*

   Performance pada Java sering dikatakan kurang tinggi. Namun *performance* Java dapat ditingkatkan menggunakan kompilasi Java lain seperti buatan *Inprise*, *Microsoft* ataupun *Symantec* yang menggunakan *Just In Time Compilers* (JIT).

10. *Multithreaded*

    Java mempunyai kemampuan untuk membuat suatu program yang dapat melakukan beberapa pekerjaan secara sekaligus dan simultan.

11. Dinamis

    Java didesain untuk dapat dijalankan pada lingkungan yang dinamis. Perubahan pada suatu *class* dengan menambahkan *properties* ataupun *method* dapat dilakukan tanpa menggangu program yang menggunakan *class* tersebut.

2.4.3.1. J2ME (Java 2 Micro Edition)

Java 2 Micro Edition atau yang biasa disebut J2ME adalah satu set spesifikasi dan teknologi yang fokus kepada perangkat konsumen. Perangkat ini memiliki jumlah memori yang terbatas, menghabiskan sedikit daya dari baterei, layar yang kecil dan bandwith jaringan yang rendah. Program J2ME, seperti semua program JAVA adalah diterjemahkan oleh VM. Program-program tersebut dikompile ke dalam bytecode dan diterjemahkan dengan Java Virtual Machine (JVM). Ini berarti bahwa program-program tersebut tidak berhubungan langsung

dengan perangkat. Aplikasi-aplikasi tersebut tidak harus dikompile ulang supaya mampu dijalankan pada mesin yang berbeda [7].

Teknologi J2ME juga memiliki beberapa keterbatasan jika diaplikasikan pada ponsel. J2ME sangat bergantung pada *device* yang digunakan, bisa dari merk ponsel, maupun kemampuan ponsel, dan dukungan terhadap teknologi J2ME. Misalnya, jika sebuah ponsel tidak memiliki kamera maka jelas J2ME pada ponsel tersebut tidak dapat mengakses kamera [7].

2.4.3.1.1. *Connected Limeted Device Configuration (CLDC)*

CLDC atau *Connected Limited Device Cofiguration* adalah perangkat dasar dari J2ME, spesifikasi dasar yang berupa *library* atau API yang diimplementasikan pada J2ME, seperti yang digunakan pada telepon selular, *pager*, dan PDA. Perangkat tsb dibatasi dengan keterbatasan *memory*, sumber daya, dan kemampuan memproses. Spesifikasi CLDC pada J2ME adalah spesifikasi minimal pada *package*, *class*, dan sebagian fungsi *Java Virtual Machine* yang dikurangi agar dpat diimplementasikan dengan keterbatasan sumber daya pada alat tersebut, JVM yang digunakan adalah KVM (*Kilobyte Virtual Machine*) [7].

2.4.3.1.2. *Mobile Information Device Profile (MIDP)*

MIDP atau *Mobile Information Device Profile* adalah spesifikasi untuk sebuah profil J2ME. MIDP memilik lapisan diatas CLDC, API tambahan untuk daur hidup aplikasi, antar muka, jaringan, dan pemyimpanan persisten. Spesifikasi MIDP, kebanyakan seperti CLDC dan API lainnya sudah digambarkan melalui *Java Community Process* (JCP). MIDP terus berkembang, dengan versi-versi masa depan yang telah lulus dari proses ketat JCP [7].

2.4.3.1.3. MIDlet

MIDlet berupa sebuah class abstrak yang merupakan subclass dari bentuk dasar aplikasi sehingga antarmuka antara aplikasi J2ME dan aplikasi manajemen pada perangkat bisa terbentuk. Suatu aplikasi MIDP disebut MIDlet. Perangkat

Application Mangement Software (AMS) berinteraksi langsung dengan MIDlet dengan method MIDlet *creat*, *start*, *pause*, dan *destroy* [7].

2.4.4. Netbean IDE

NetBeans adalah adalah arsitektur *rich-client* dan menyediakan berbagai *tools* untuk mengembangkan aplikasi yang berbeda seperti WEB, Enterprise, *plug-in module*, dan aplikasi *mobile* dengan bahasa yang berbeda termasuk java, C/C++ dan Ruby. Netbeans IDE kompatibel terhadap Windows, Linux, UNIX, dan solaris [3].

Fitur-fitur yang terdapat dalam Netbeans IDE [3] :

a. *Additional Language*
b. *Improve Code Editor*
c. *The Netbeans Profiler*
d. *Swing Desktop Database*
e. *Version Control*