BAB 2
LANDASAN TEORI

2.1 Game

Game merupakan salah satu media hiburan yang paling popular untuk semua kalangan usia. Sejak pertama kali ditemukan sampai saat sekarang, teknologi *game* telah mengalami kemajuan yang terbilang pesat. Hal ini ditandai dengan berkembangnya jenis *game*, produk, alat dan jenis interaksi *game* dengan penggunaan yang semakin beragam bentuknya [1].

2.1.1 Pengertian *Game*

Game merupakan sebuah bentuk seni dimana penggunanya disebut dengan pemain (*player*), diharuskan membuat keputusan-keputusan dengan tujuan mengelola sumber daya yang diperoleh dari kesempatan-kesempatan bermain (*token*) miliknya untuk mencapai tujuan tertentu. Video *game* adalah bentuk *game* yang interaksi umunya melibatkan media video dan audio.

Menurut Andang Ismail terdapat dua pengertian *game* (permainan). Pertama, *game* (permainan) adalah sebuah aktifitas bermain yang murni mencari kesenangan tanpa mencari menang atau kalah. Kedua, permainan diartikan sebagai aktifitas bermain yang dilakukan dalam rangka mencari kesenangan dan kepuasan, namun ditandai pencarian menang – kalah.

Berdasarkan representasinya, *game* dapat dibedakan menjadi 2 jenis yaitu *game* 2 dimensi (2D) dan 3 dimensi (3D). *Game* 2D adalah *game* yang secara matamatis hanya melibatkan 2 elemen koordinat kartesius yaitu x dan y, sehingga konsep kamera pada *game* 2D hanya menentukan gambar pada *game* yang dapat dilihat oleh pemain. Sedangkan *game* 3D adalah *game* yang selain melibatkan elemen x dan y juga melibatkan elemen z pada perhitungannya sehingga konsep kamera pada *game* 3D benar-benar menyerupai konsep kamera pada kehidupan nyata [1].

2.1.2 Sejarah *Game*

Sejak pertama kali ditemukan sampai saat ini, *game* telah melewati beberapa fase evolusi penting dimana yang berperan besar terhadap perkembangan *game* sudah semakin pesat sekarang ini.

Game pertama kali dibuat pada tahun 1966 oleh Ralph Baer bersama dengan timnya yang berjumlah 500 orang yang terdiri dari para insinyur dan teknisi. Setelah sebulan bekerja keras, Baer berhasil menampilkan dua titik putih yang bekejar-kejaran dilayar. Pada tahun 1972 muncul nama baru dalam dunia *game*. Nolan Bushel mendirikan perusahaan Atari dan membuat *game* Arcade Pong. Pada era baru dalam perkembagan dunia *game* terjadi pada tahun 1988 yang didominasi oleh perusahaan Jepang. Nintendo yang awalnya hanya memproduksi mesin fotocopy, dengan video *game* sistemnya telah mencapai omset miliaran dolar. Setalah masa awal perkembagan *game* ini, dunia *game* telah melalui beberapa fase yaitu fase *game* PC, fase *game console* dan fase *game online*. Sekarang ketiga fase *game* tersebut sudah semakin banyak variannya dan masing-masing memiliki kelompok penggemarnya sendiri [1].

2.1.3 Sejarah *Game* Pertama di Indonesia

Divinekids adalah *game* yang diakui dan mendapat award resmi dari MURI (Museum Rekor Indonesia) sebagai *game* pertama Indonesia pada tahun 2004 [1]. Akan tetapi sebenarnya sudah banyak *game-game* buatan orang-orang Indonesia yang lain tetapi karena kekurangan syarat-syarat minimal seperti karakter yang menggunakan karakter asing, penyebaran yang kurang luas, bahasa yang digunakan bahasa asing dan lain sebagainya. Oleh karena itu tidak terpilih sebagai *game* pertama Indonesia secara resmi.

Menurut sebagian besar pengamat *game* di Indonesia, ini adalah kurangnya investigasi MURI dalam melihat adanya *game-game* tersebut sebelum memberikan gelar *game* pertama Indonesia. Pengamat-pengamat yang berpendapat berbeda dengan MURI memposisikan bahwa sejarah *game* pertama Indonesia adalah *Game* devid. Pada tahun 2006, *game-game* Divinekids berjumlah 38 lebih dan terus bertambah yang permbuatnya adalah David

Setiabudi. Pada tahun 1992, David Setiabudi juga membuat *game* namun tidak diakui sebagai *game* pertama Indonesia karena kurang memenuhi syarat. Ada juga *game-game* buatan Korea yang terkenal diterjemahkan ke bahasa Indonesia dan *game-game* yang penyebarannya kurang luas serta berkarakter asing atau berbahasa asing secara resmi tidak terpilih menjadi *game* pertama Indonesia [1].

2.1.4 Jenis-jenis *Game*

Berikut ini beberapa jenis *game* yang dapat dimainkan, hal ini dibedakan berdasarkan cara bermain yang dapat dilakukan oleh para pemain.

A. *Action*

Permainan aksi (*Action games*) memerlukan pemain untuk menggunakan refleks, akurasi, dan waktu yang tepat untuk menyelesaikan sebuah tantangan. Ini mungkin merupakan genre dasar dari sebuah permainan, dan yang memiliki permainan terbanyak. Dalam permainan aksi, biasanya terdapat pertempuran.[6] Contoh *Game Action* seperti pada gambar 2.1.



Gambar 2.1 Contoh Permainan *Action [6]*

B. *Platformer*

Permainan platform asli terdiri dari karakter yang berlari dan melompat pada arena bermain dengan mode side-scrolling[6]. Beberapa contoh dari *Platformer*s yang terkenal diantaranya Super Mario Bros, contoh lain *game platformers* seperti pada gambar 2.2.



Gambar 2.2 Contoh Permainan *Platformers [6]*

C. *Adventure*

Permainan *Adventure* adalah permainan yang menawarkan eksplorasi dan memecahkan teka-teki sebagai daya tarik utama[6]. Permainan ini menawarkan histori cerita yang paling mengasyikkan, meskipun popularitas mereka telah menurun dalam 2 dekade terakhir. Penalaran, kreativitas, dan rasa ingin tahu adalah keterampilan yang paling umum yang diperlukan dari pemain *game* petualangan yang baik. Contoh dari *game Adventure* seperti pada gambar 2.3.



Gambar 2.3 Contoh Permainan *Adventure [6]*

Permainan petualangan aksi *(Action-adventure games)*, yaitu ragam permainan yang menempatkan pemain sebagai tokoh dalam permainan tersebut, mirip dengan permainan peran, tetapi ditambah dengan unsur-unsur aksi, misalnya perkelahian dan tembak-menembak [6]. Contoh permainan dalam ragam ini adalah serial *Prince of Persia* dan *Devil May Cry*. seperti pada gambar 2.4.



Gambar 2.4 Contoh permainan *Action-Adventure [6]*

D. *Fighting*

Permainan pertarungan (*Fighting games*) adalah ragam permainan dimana terjadi pertarungan satu lawan satu antara dua karakter, salah satunya dapat merupakan komputer.[6] Ragam permainan ini biasanya dimainkan dengan menghubungkan kombinasi-kombinasi tombol untuk bertarung. Rata-rata gerakan yang dihasilkan oleh karakter dramatis dan tidak mungkin dilakukan di kehidupan nyata. Tidak semua permainan pertarungan merupakan satu lawan satu, ada juga yang, seperti serial *Super Smash Bros.* dan *Guilty Gear Isuka*, menempatkan empat karakter dalam satu kali pertarungan. Ragam ini pertama kali muncul pada permainan *Heavyweight Boxing* buatan SEGA pada tahun 1976. Contoh permainan ragam ini adalah serial *Street Fighter* dan *Tekken*, seperti pada gambar 2.5.



Gambar 2.5 Contoh permainan *Fighting [6]*

E. *Shooter Games*

Permainan tembak-menembak (*Shooter games*), adalah sub-genre dari permainan aksi, meskipun banyak pemain yang menganggap bahwa ragam ini merupakan ragam yang berdiri sendiri [6]. Ragam ini menguji kecepatan dan reaksi dari pemainnya. Tembak-menembak memiliki sub-genre tersendiri, yang semuanya terfokus pada "mengontrol seseorang yang menggunakan sebuah senjata, biasanya pistol, senapan, atau senjata jarak jauh lainnya". Hal yang biasanya ada pada permainan ragam ini adalah amunisi. Biasanya tujuan dari permainan ini adalah menembak lawan dan melanjutkan misi tanpa karakter yang dipakai mati.

1. *First-person shooter* (FPS), menekankan ke menembak dengan perspektif orang-pertama yang dikontrol oleh pemain. Perspektif ini bertujuan untuk memberikan pemain rasa seakan benar-benar berada disana, dan dapat membuat pemain fokus membidik sasaran. Kebanyakan FPS memerlukan refleks yang cepat dalam tingkat kesulitan yang tinggi. Unsur 3D diperlukan untuk membuat FPS yang baik. *Wolfenstein 3D* merupakan FPS terkenal pertama yang diliris pada 1992, dan *Doom* merupakan terobosan besar dalam grafik. *Doom* berhasil membuat teknologi yang membuat permainan dapat berjalan dengan mulus di PC tingkat konsumer. Sejak kemunculannya, FPS sekarang memiliki fitur *multiplayer* (multipemain) yang memungkinkan berkompetisi antar pemain. *Call of*

Duty, *Counter-Strike*, *Half-Life*, *Halo*, *Battlefield* dan *Max Payne* adalah contoh FPS yang terus berkembang hingga sekarang.

2. *Third-person shooter* (TPS/3PS) menekankan ke menembak dari persperktif orang-ketiga atau karakter pemainnya dapat terlihat sepenuhnya. Persperktif ini memberikan pemain pandangan yang lebih luas akan lingkungan sekitar dari FPS. Juga, TPS memungkinkan pergerakan yang lebih rumit seperti bergulir atau menyelam, dibandingkan dengan melompat dan merunduk sederhana yang biasa terlhat di FPS. Meskipun begitu, kebanyakan jenis permainan ini memiliki elemen puzzle dari FPS. Ada beberapa TPS, seperti serial *Star Wars: Battlefront* dan *Rachet and Clank*, yang memungkinkan untuk merubah sudut pandang menjadi orang-pertama. Para pembuat permainan TPS sekarang telah mulai menggabungkan elemen dari TPS dengan RPG atau ragam lainnya, seperti yang terlihat pada trilogy RPG terkenal *Mass Effect*.
3. *Massively multiplayer online first-person shooter* (MMOFPS), merupakan permainan online yang menggabungkan pemain FPS pada sebuah dunia virtual dimana para pemain dapat berinteraksi satu dengan lainnya. Contoh dari MMOFPS ini adalah *Point Blank* dan *Cross Fire*.

Salah satu contoh *Shooter Games* seperti pada gambar 2.6.



Gambar 2.6 Contoh dari permainan *Shooter Games [6]*

F. *Strategy*

1. *Real-time strategy* (RTS), yaitu ragam permainan yang mengandung unsur strategi. Permainan RTS memerlukan kecakapan pemain untuk memimpin suatu bangsa, mengelola sumber daya, dan melancarkan perang untuk memperluas wilayah kekuasaan. Contoh permainan dalam ragam ini adalah serial *Age of Empires*, *Civilization*, *Rise of Nations*, dan *Warcraft*.[6]
2. *Turn-based strategy* (TBS), yaitu ragam permainan video strategi yang pemainnya saling menunggu giliran untuk bergerak., seperti pada gambar 2.7.



Gambar 2.7 Contoh permainan *Real-Time Strategy* (RTS) [6]

G. *Role-Playing Game* (RPG)

Permainan peran (*Role-playing games*), yaitu ragam permainan video yang menempatkan pemain sebagai tokoh dalam permainan tersebut untuk memecahkan suatu misteri dengan menyelesaikan berbagai macam *puzzle* dan *quest* [6]. Ragam ini biasanya memiliki alur cerita yang kompleks. Contoh permainan video dalam ragam ini adalah serial *Final Fantasy*, *Star Ocean*, dan *Suikoden*. seperti pada gambar 2.8.



Gambar 2.8 Contoh dari Role-Playing Game [6]

H. *Simulation*

Permainan simulasi diadaptasikan dari sistem simulasi. Elemen utama dari permainan simulasi adalah kemampuannya untuk mencocokkan situasi seperti di dunia nyata[6]. Simulasi berusaha untuk memberikan kenikmatan melalui pemeragaan. Simulasi peperangan dan simulasi balapan mobil relatif populer dalam jenis permainan. Simulasi juga dapat mencakup simulasi situasi sosial seperti Sims. Salah satu contoh dari permainan ini adalah The Sims, seperti pada gambar 2.9.



Gambar 2.9 Contoh permainan *Simulation [6]*

I. *Racing*

Racing game juga termasuk dalam kategori ini,dimana ragam permainan yang menuntut keterampilan pemain untuk mengemudi dalam sebuah kompetisi balap-membalap [6]. Contoh permainan dalam ragam ini adalah serial *Need for Speed* dan *Juiced*. seperti pada gambar 2.10.



Gambar 2.10 Contoh permainan *Racing [6]*

J. *Sports*

Permainan olahraga (*Sports games*), yaitu ragam permainan yang menuntut keterampilan pemain untuk melakukan pertandingan olahraga secara virtual, seperti pertandingan sepak bola, basket, dan sebagainya. Contoh permainan dalam ragam ini adalah serial *Pro Evolution Soccer* dan *Madden NFL*, seperti pada gambar 2.11.



Gambar 2.11 Contoh permainan *SPORT [6]*

K. *Puzzle*

Permainan *Puzzle* merupakan permainan yang menawarkan teka-teki sebagai daya tarik utama untuk *game*[6]. Permainan ini paling sering dirilis pada anggaran yang rendah melalui web. Orang-orang yang bermain *game-game* ini cenderung menjadi populasi tertua dari permainan bermain masyarakat. Salah satu permainan *Puzzle* paling sukses dan terkenal adalah Tetris, Lemmings dan Minesweeper. I.Q. Cube cerdas adalah lain *Puzzle*. Contoh *game Puzzle* seperti pada gambar 2.12.



Gambar 2.12 Contoh dari permainan *PUZZLE [6]*

2.1.5 Unsur *Game*

Dalam sebuah *game* terdapat unsur-unsur yang melengkapinya. Berikut beberapa unsur dalam suatu *game* yaitu [7]:

1. Warna

   Warna mempunyai kemampuan untuk membuat orang tanggap terhadap semua yang dilihat karena tidak ada seseuat hal bermakna tanpa warna. Warna terlihat sebelum penampakan gambar. Mata manusia tertarik oleh warna pada suatu *level* karena warna dari objek diterima sebelum detail-detail dipisahkan oleh bentuk-bentuk dan garisnya. Warna merah memiliki panjang gelombang yang terpanjang, biru memiliki panjang gelombang yang pendek sedangkan hijau memiliki panjang gelombang menengah. Pada anak-anak cenderung tertarik pada warna-warna yang cerah dan mencolok. Warna-warna yang cerah

terutama warna *primer* (merah, kuning, biru) dan warna *sekunder* (orange, ungu, hijau).



Gambar 2.13 Warna *Primer* dan *Sekunder* *[7]*

2. Komposisi

Komposisi adalah pengaturan segala elemen didalam sebuah karya desain yang sedemikian rupa dengan tujuan tertentu.Komposisi yang baik adalah komposisi yang mampu memenuhi kebutuhan dan tujuan desain, mudah dipahami dan membentuk kesatuan yang serasi dan harmonis. Kemudian *layout* yaitu perencanaan, penempatan semua unsur mulai dari tulisan, gambar, ilustrasi, teks, nama dan sebagainya dengan pengkuran secara seksama. Komposisi yang sesuai dengan anak-anak adalah komposisi yang sederhana dan tidak menggunakan petunjuk terlalu rumit. Kesederhanaan diwujudkan dengan penggunaan visual 2D dan penerapan warna-warna dalam seluruh aspek desain.

3. Bentuk dasar

Bentuk dasar adalah bentuk-bentuk yang mudah ditemui dalam kehidupan sehari-hari seperti kotak, lingkaran, segitiga dan lain sebagainya.



Gambar 2.14 Bentuk Dasar [7]

4. Tipografi

Tipografi merupakan representasi visual dari sebuah bentuk komunikasi *verbal* dan merupakan *property visual* yang pokok dan efektif. Huruf memainkan peranan penting dalam keberhasilan suatu bentuk komunikasi grafis. Dalam media pembelajaran untuk anak-anak, sebuah huruf harus *legible* yaitu jelas dan memiliki tingkat kemudahan untuk dibaca.

5. Audio

Audio adalah sinyal elektrik yang digunakan untuk membawa suara dalam batas pendengaran manusia. Audio merupakan komponen sistem yang sudah termasuk didalamnya atau dapat ditambahkan pada komputer.

2.1.6 Gameplay *Action-Adventure*

Action Game (Permainan Aksi) adalah jenis permainan yang menekankan pada tantangan fisik, yang membutuhkan reaksi cepat dengan koordinasi antara mata dan tangan. *Adventure game* adalah salah satu *genre game* yang berfokus pada eksplorasi dan memecahkan teka-teki. Permainan *action-adventure* merupakan gabungan dari keduanya, dimana permainan ini menekankan pada tantangan fisik, mempunyai unsur *item gathering*, penyelesaiaan *puzzle simple* serta pertarungan dan dapat pula berupa petualangan seorang karakter yg penuh dengan aksi yg akan terus ada hingga game tersebut selesai [1].

2.1.7 Klasifikasi *Game* Berdasarkan Umur

The Entertainment Software Rating Board (ESRB) adalah sebuah organisasi regulator mandiri yang menilai permainan video, panduan pengiklanan, prinsip privasi online permainan video dan software hiburan lainnya di Kanada dan Amerika Serikat. ESRB dibentuk pada tahun 1994 oleh *Entertainment Software Association* (sebelumnya *Interactive Digital Software Association*). Sejak tahun 2003, ESRB telah menilai 8,000 permainan yang di produksi oleh 350 penerbit permainan *video*.[8]

Berikut ini adalah klasifikasi *game* berdasrkan pembatas usia, menurut *Entertainment Software Rating Board* (ESRB), yaitu adalah :

1. *Early Childhood* (3+) memiliki isi yang mungkin cocok untuk usia 3 dan lebih tua. Tidak mengandung materi yang tidak pantas orang tua akan menemukan.[8]

2. *Everyone* (6+) memiliki konten yang mungkin cocok untuk usia 6 dan lebih tua. Judul-judul dalam kategori ini mungkin berisi sedikit kartun, fantasi atau kekerasan ringan dan jarang menggunakan bahasa ringan.[8]

3. *Everyone* (10+) memiliki konten yang mungkin cocok untuk usia 10 dan lebih tua. Judul-judul dalam kategori ini mungkin berisi lebih banyak kartun, fantasi atau kekerasan ringan, bahasa ringan dan minimal tema sugestif.[8]

4. *Teen* (13+) memiliki konten yang mungkin cocok untuk usia 13 dan lebih tuaJudul-judul dalam kategori ini mungkin mengandung kekerasan, tema sugestif, humor kasar, sedikit darah, simulasi perjudian, dan jarang menggunakan bahasa yang kuat. [8]

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 dan darah kental, konten seksual dan bahasa yang kuat.[8]

6. *Adults Only* (18+) memiliki konten yang hanya boleh dimainkan oleh orang-orang 18 tahun dan lebih tua. Judul-judul dalam kategori ini mungkin termasuk adegan yang intens lama kekerasan atau grafik konten seksual dan ketelanjangan.[8]

7. RP (*Rating Pending*) telah disampaikan kepada ESRB dan sedang menunggu penilaian akhir. (Simbol ini hanya muncul dalam iklan sebelum permainan dibebaskan.[8]

2.2 AI (Artificial Intelligence)

2.2.1 Definisi AI

Sebagian kalangan menerjemahkan *Artificial Intelligence* sebagai kecerdasan buatan, kecerdasan artifisial, intelijensia artifisial, atau intelejensia buatan. Pada bukunya[5], Suyanto sengaja tidak menerjemahkan istilah *Artificial Intelligence* ke bahasa Indonesia karena istilah tersebut sudah sangat akrab bagi orang Indonesia. Begitu juga dengan singkatan istilah tersebut, yaitu AI, sudah sangat melekat di berbagai media ilmiah maupun non ilmiah.

Para ahli mendefinisikan AI secara berbeda-beda tergantung pada sudut pandang mereka masing-masing. Ada yang focus pada logika berpikir manusia saja, tetapi ada juga yang mendefinisikan AI secara lebih luas pada tingkah laku manusia. Stuart Russel dan Peter Norvig mengelompokkan definisi AI, yang diperoleh dari beberapa textbook berbeda, ke dalam empat kategori[5], yaitu :

A. Thinking humanly : the cognitive modeling approach

Pendekatan ini dilakukan dengan dua cara sebagai berikut[5] :

1. Melalui introspeksi : mencoba menangkap pemikiran-pemikiran kita sendiri pada saat kita berpikir. Tetapi, seorang psikolog Barat mengatakan “how do you know that you understand?” Bagaimana anda tahu bahwa anda mengerti? Karena pada saat anda menyadari pemikiran anda. Sehingga definisi ini terkesan mengada-ngada dan tidak mungkin dilakukan.
2. Melalui eksperimen-eksperimen psikologi.

B. Acting humanly : the Turing test approach

Pada tahun 1950, Alan Turing merancang suatu ujian bagi komputer berintelijensia untuk menguji apakah komputer tersebut mampu mengelabui seorang manusia yang menginterogasinya melalui teletype (komunikasi berbasis teks jarak jauh). Jika interrogator tidak dapat membedakan yang d2nterogasi adalah manusia atau komputer, maka komputer berintelijensia tersebut lolos dari

Turning test. Komputer tersebut perlu memiliki kemampuan : Natural Language Processing, Knowledge Representation, Automated Reasoning, Machine Learning, Computer Vision, Robotics. Turing test sengaja menghindari interaksi fisik antara interrogator dan komputer karena simulasi fisik manusia tidak memerlukan intelijensia.

C. Thinking rationally : the laws of thought approach

Terdapat dua masalah dalam pendekatan ini, yaitu [5]:

1. Tidak mudah untuk membuat pengetahuan informal dan menyatakan pengetahuan tersebut ke dalam formal term yang diperlukan oleh notasi logika, khususnya ketika pengetahuan tersebut memiliki kepastian kurang dari 100%.
2. Terdapat perbedaan besar antara dapat memecahkan masalah "dalam prinsip" dan memecahkannya "dalam dunia nyata".

D. Acting rationally : the rational agent approach

Membuat inferensi yang logis merupakan bagian dari suatu rational agent. Hal ini disebabkan satu-satunya cara untuk melakukan aksi secara rasional adalah dengan menalar secara logis. Dengan menalar secara logis, maka bisa didapatkan kesimpulan bahwa aksi yang diberikan akan mencapai tujuan atau tidak. Jika mencapai tujuan, maka agent dapat melakukan aksi berdasarkan kesimpulan tersebut.

Thinking humanly dan acting humanly adalah dua definisi dalam arti yang sangat luas. Sampai saat ini, pemikiran manusia yang diluar rasio, yakni reflex dan intuitif (berhubungan dengan perasaan), belum dapat ditirukan sepenuhnya oleh komputer. Dengan demikian, kedua definisi ini dirasa kurang tepat untuk saat ini. Jika kita menggunakan definisi ini, maka banyak produk komputasi cerdas saat ini yang tidak layak disebut sebagai produk AI.

Definisi thinking rationally terasa lebih sempit daripada acting rationally. Oleh karena itu, definisi AI yang paling tepat untuk saat ini adalah acting rationally dengan pendekatan rational agent. Hal ini berdasarkan pemikiran bahwa

komputer bisa melakukan penalaran secara logis dan juga bisa melakukan aksi secara rasional berdasarkan hasil penalaran tersebut.

2.2.2 Teknik Pemecahan Masalah AI

Terdapat empat teknik dasar pemecahan masalah yang terdapat pada bidang academic/traditional *Artificial Intelligence* (AI)[5], diantaranya adalah sebagai berikut :

A. Searching

Pada teknik searching atau pencarian ini terdiri dari beberapa langkah untuk merealisasikannya. Langkah pertama adalah mendefinisikan ruang masalah untuk suatu masalah yang dihadapi. Langkah kedua adalah mendefinisikan aturan produksi yang digunakan untuk mengubah suatu keadaan ke keadaan lainnya. Langkah terakhir adalah memilih metode pencarian yang tepat sehingga dapat menemukan solusi terbaik.

Metode-metode pencarian pada teknik searching diantaranya[5] :

1. Blind/Un-informed Search
   a. Breadth-First Search (BFS)
   b. Depth-First Search (DFS)
   c. Depth-Limited Search (DLS)
   d. Uniform Cost Search (USC)
   e. Iterative-Deepening Search (IDS)
   f. Bi-Directional Search (BDS)
2. Metode Pencarian Heuristik
   a. Generate-and-Test
   b. Hill Climbing
   c. Simulated Annealing
   d. Best-First Search (BFS)
   e. Greedy Best-First Search
   f. A* (*A star*)

B. Reasoning

Teknik reasoning atau penalaran merupakan teknik penyelesaian masalah dengan cara merepresentasikan masalah ke dalam basis pengetahuan (knowledge base) menggunakan logic atau bahasa formal (bahasa yang dipahami komputer). Teknik ini melakukan proses penalaran berdasarkan basis pengetahuannya untuk menemukan solusi.

C. Planning

Planning adalah suatu metode penyelesaian masalah dengan cara memecah masalah ke dalam sub-sub masalah yang lebih kecil, menyelesaikan sub-sub masalah satu demi satu kemudian menggabungkan solusi-solusi dari sub-sub masalah tersebut menjadi sebuah solusi lengkap dengan tetap mengingat dan menangani interaksi yang terdapat pada sub-sub masalah tersebut.

D. Learning

Pada ketiga teknik sebelumnya, seseorang harus mengetahui aturan yang berlaku untuk sistem yang akan dibangunnya. Tetapi, pada masalah tertentu terkadang suatu aturan tidak bisa didefinisikan secara benar ataupun lengkap. Hal tersebut mungkin dikarenakan data-data yang didapat tidak lengkap. Melalui teknik yang disebut learning ini, secara otomatis aturan yang diharapkan bisa berlaku umum untuk data-data yang belum pernah diketahui dapat ditemukan.

2.2.3 Algoritma A* *(A Star)*

Algoritma A* *(A Star)* merupakan perbaikan dari metode BFS dengan memodifikasi fungsi *heuristic*nya. A* *(A Star)* akan meminimumkan total biaya lintasan. Pada kondisi yang tepat, A* akan memberikan solusi yang terbaik dalam waktu yang optimal. Pencarian menggunakan algoritma A* mempunyai prinsip yang sama dengan algoritma BFS hanya saja dengan dua faktor tambahan yaitu :

1. Setiap sisi mempunyai *cost* yang berbeda-beda, seberapa *cost* untuk pergi dari satu simpul ke simpul 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 ingin mencari jalan dengan waktu tercepat untuk dilalui. Algoritma A* bekerja dengan prinsip yang hampir sama kecuali dengan dua perbedaan yaitu :

1. Simpul-simpul di *list* terbuka diurutkan oleh *cost* keseluruhan dari simpul awal ke simpul tujuan, dari *cost* terkecil sampai *cost* terbesar. *Cost* keseluruhan dihitung dari *cost* dari simpul awal ke simpul sekarang (*current node*) ditambah *cost* perkiraan menuju simpul tujuan.
2. Simpul di *list* tertutup bisa dimasukkan ke *list* terbuka bila jalan terpendek menuju simpul tersebut ditemukan.

Cost antara simpul adalah jaraknya dan perkiraan *cost* dari suatu simpul ke simpul tujuan adalah penjumlahan jarak dari simpul tersebut ke simpul tujuan. Untuk lebih mudah dimengerti, berikut rumusnya.

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

keterangan :

f(n) = fungsi evaluasi

g(n) = biaya yang sudah dikeluarkan dari keadaan awal 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.Keoptimalan dari A* cukup langsung dinilai optimal jika h(n) adalah *admissible heuristic* yaitu nilai h(n) tidak akan memberikan penilaian lebih pada *cost* untuk mencapai tujuan. Salah satu contoh dari *admissible heuristic* adalah jarak dengan menarik garis lurus karena jarak terdekat dari dua titik adalah dengan menarik garis lurus [9].

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

1. **Simpul awal** adalah sebuah terminologi untuk posisi awal sebuah benda.
2. ***Current node*** adalah simpul yang sedang dijalankan dalam algoritma pencarian jalan terpendek.
3. **Simpul** adalah petak-petak kecil sebagai representasi dari area *pathfinding*. Bentuknya dapat berupa persegi, lingkaran, maupun segitiga.
4. ***Neighbor node*** adalah simpul-simpul yang bertetangga dengan *current node*.
5. ***Open List*** adalah tempat menyimpan data simpul yang mungkin diakses dari *starting point* maupun simpul yang sedang dijalankan.
6. ***Closed List*** adalah tempat menyimpan data simpul sebelum *current node* yang juga merupakan bagian dari jalur terpendek yang telah berhasil didapatkan.
7. ***Parent*** adalah tempat menyimpan data ketetanggaan dari suatu simpul, misalnya y *parent* x artinya *neighbor node* y dari *current node* x.
8. **Harga (F)** adalah nilai yang diperoleh dari penjumlahan nilai G, jumlah nilai tiap simpul dalam jalur terpendek dari *starting point* ke *current node*, dan H, jumlah nilai perkiraan dari sebuah simpul ke *target point*.
9. ***Target point*** yaitu simpul yang dituju.
10. ***Walkability*** adalah sebuah atribut yang menyatakan apakah sebuah simpul dapat atau tidak dapat dilalui oleh *current node*.

Algoritma A* secara ringkas langkah demi langkahnya adalah sebagai berikut:

1. Tambahkan simpul awal ke dalam *open list*.
2. Ulangi langkah berikut sampai pencarian berakhir:
   a. Carilah simpul n dengan biaya F(n) paling rendah, dalam *open* list. Simpul dengan biaya F terendah kemudian disebut *current node*.
   b. Keluarkan *current node* dari *open list* dan masukkan ke dalam *closed list*.

c. Untuk setiap 8 simpul (*neighbor node*) dari *current node* lakukan langkah berikut:
    1). Jika sudah terdapat dalam *closed list* atau tidak *walkable*, maka abaikan, jika tidak lanjutkan.
    2). Jika belum ada pada *open list*, tambahkan ke *open list*. Simpan *current node* sebagai *parent* dari *neighbor node* ini. Simpan harga F masing-masing simpul.
    3). Jika sudah ada dalam *open list*, periksa apakah ini jalan dari simpul ini ke *current node* yang lebih baik dengan menggunakan biaya G sebagai ukurannya. Simpul dengan biaya G yang lebih rendah berarti bahwa ini adalah jalan yang lebih baik. Jika demikian, buatlah simpul ini (*neighbor node*) sebagai *parent* dari *current node*, dan menghitung ulang nilai G dan F dari simpul ini.

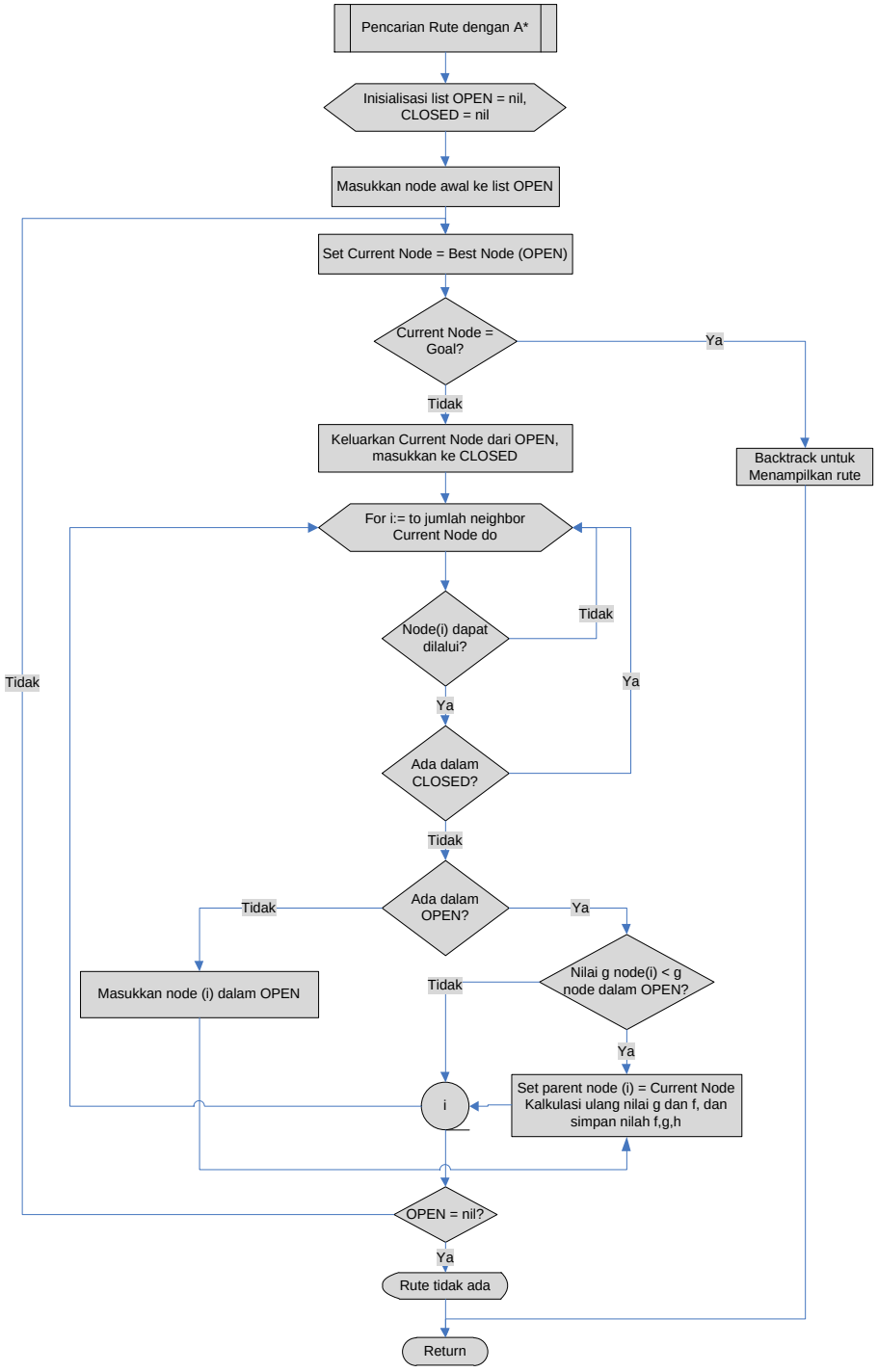
d. Berhenti ketika:
    1. Menambahkan *target point* ke dalam *closed list*, dalam hal ini jalan telah ditemukan, atau,
    2. Gagal untuk menemukan *target point*, dan *open list* kosong. Dalam kasus ini, tidak ada jalan.

e. Walaupun telah mencapai *target point*, jika masih ada *neighbor node* yang memiliki nilai yang lebih kecil, maka simpul tersebut akan terus dipilih sampai bobotnya jauh lebih besar atau mencapai *target point* dengan bobot yang lebih kecil dibanding dengan simpul sebelumnya yang telah mencapai *target point*.

f. Pada saat pemilihan simpul berikutnya, nilai F(n) akan dievaluasi, dan jika terdapat nilai F(n) yang sama maka akan dipilih berdasarkan nilai G(n) terbesar.

3. Simpan jalan. Bekerja mundur dari *target point*, pergi dari masing-masing simpul ke simpul *parent* sampai mencapai *starting point*.

Secara umum dapat digambarkan dengan *flowchart* sebagai berikut:



Gambar 2.15 *Flowchart* Algoritma A* [9]

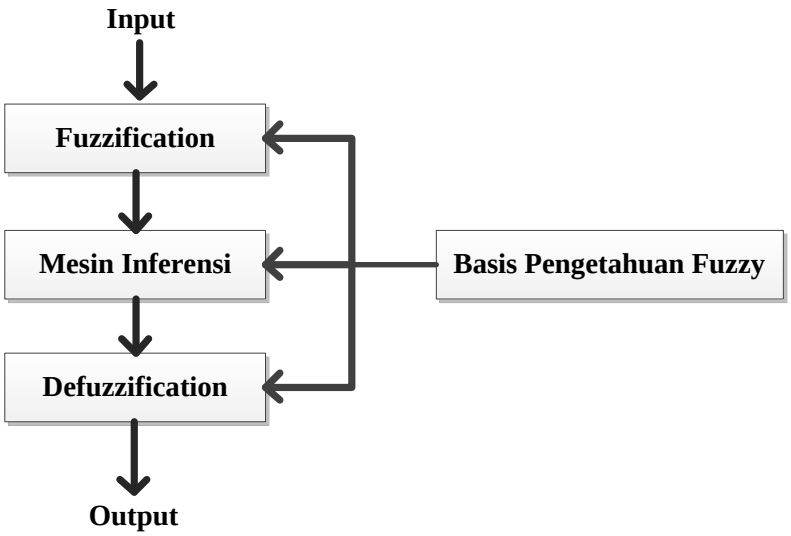
2.2.4 *Fuzzy Logic*

Konsep tentang logika *fuzzy* diperkenalkan oleh Prof. Lotfi Astor Zadeh pada 1962. Logika fuzzy adalah metodologi sistem kontrol pemecahan masalah, yang cocol diimplementasikan pada sistem, mulai dari sistem yang sederahana, sistem kecil, *embedded system*, jaringan PC, *multi-channel* atau *workstation* berbasis akurasi data, dan sistem kontrol. Metodologi ini dapat diterapkan pada perangkat keras, perangkat lunak, atau kombinasi keduanya. Dalam logika klasik dinyatakan bahwa segala sesuatu bersifat biner, yang artinya adalah hanya mempunyai dua kemungkinan, "Ya atau Tidak", "Benar atau Salah", "Baik atau Buruk" dan lain-lain. Oleh karena itu, semua ini dapat mempunyai nilai keanggotaan 0 dan 1. Akan tetapi, dalam logika *fuzzy* memungkinakan nilai keanggotaan berada diantara 0 dan 1. Artinya, bisa saja suatu keadaan mempunyai dua nilai "Ya dan Tidak", "Benar dan Salah", "Baik dan Buruk" secara bersamaan, namun besar nilainya tergantung pada bobot keanggotaan yang dimilikinya. Logika *fuzzy* dapat digunakan diberbagai bidang, seperti pada sistem diagnosis penyakit (dalam bidang kedokteran), pemodelan sistem pemasaran, riset operasi (dalam bidang ekonomi), kendali kualitas air, prediksi adanya gempa bumi, klasifikasi dan pencocokan pola (dalam bidang teknik) [10].

Konsep himpunan fuzzy memiliki 2 atribut, yaitu [10] :

1. *Linguistik*, yaitu nama suatu kelompok yang mewakili suatu keadaan tertentu dengan menggunakan bahasa alami, misalnya dingin, sejuk, panas mewakili variable temperatur.
2. *Numeris*, yaitu suatu nilai yang menunjukan ukuran dari suatu variabel, misalnya 10, 35, 40 dan sebagainya.

Cara kerja logika fuzzy dapat digambarkan sebagai struktur elemen dasar sistem inferensi *fuzzy* sebagai berikut:



Gambar 2.16 Struktur sistem inferensi fuzzy [10].

Keterangan :

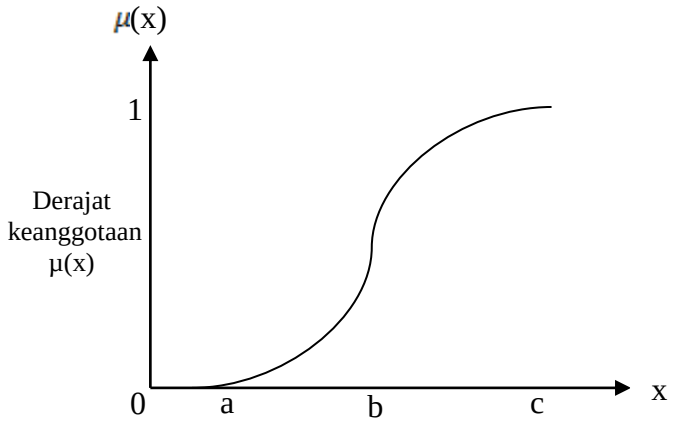
1. Basis Pengetahuan *Fuzzy* merupakan kumpulan *rule-rule fuzzy* dalam bentuk pernyataan IF…THEN.
2. *Fuzzyfikasi* adalah proses untuk mengubah input sistem yang mempunyai nilai tegas menjadi variabel linguistic menggunakan fungsi keanggotaan yang disimpan dalam basis pengetahuan *fuzzy.*
3. Mesin Inferensi merupakan proses untuk mengubah input *fuzzy* dengan cara mengikuti aturan-aturan (*IF-THEN Rules*) yang telah ditetapkan pada basis pengetahuan *fuzzy.*
4. *DeFuzzyfikasi* merupakan proses mengubah output *fuzzy* yang diperoleh dari mesin inferensi menjadi nilai tegas menggunakan fungsi keanggotaan yang sesuai dengan saat dilakukan *fuzzyfikasi*.

2.2.4.1 Fungsi Keanggotaan (Member function)

Pada fuzzy system, fungsi keanggotaan memainkan peran yang sangat penting untuk merepresentasikan masalah dan menghasilkan keputusan yang akurat. Terdapat banyak sekali fungsi keanggotaan yang dapat digunakan, diantaranya adalah [10] :

1. Fungsi sigmoid

Sesuai dengan namanya, fungsi ini berbentuk kurva sigmoidala seperti huruf S. Setiap nilai x (anggota crips set) dipetakan ke dalam interval [0,1]. Grafik dan notasi matematika untuk fungsi ini adalah sebagai berikut :



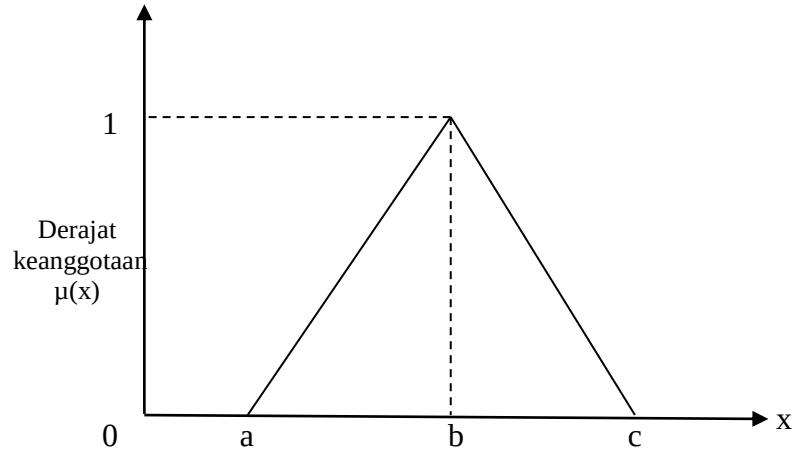
Gambar 2.17 Grafik fungsi sigmoid [10]

Fungsi keanggotaan :

$$Sigmoid(x,a,b,c) = \begin{cases} 0; x \leq a \\ 2(x-a)/(c-a)^2; a \leq x \leq b \\ 1-2((c-x)/(c-a))^2; b \leq x \leq c \\ 1; x \geq c \end{cases}$$

2. Fungsi Segitiga

Fungsi segitiga hanya terdapat satu nilai x yang memiliki derajat kenanggotaan sama dengan 1, yaitu ketika x = b. Tetapi, nilai-nilai disekitar b memiliki derajat keanggotaan yang turun cukup tajam (menjauhi 1) [10]. Grafik dan notasi matematika dari fungsi segitiga adalah sebagai berikut : $\mu(x)$



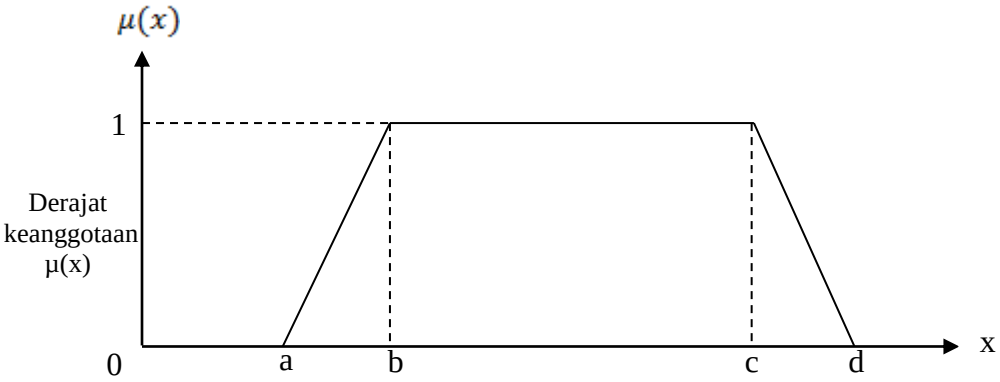
Gambar 2.18 Graik fungsi segitiga [10]

Fungsi keanggotaan :

$$\mu(x) = \begin{cases} 0;\ x \leq a\ atau\ x \geq c \\ (x-a)/(b-a);\ a \leq x \leq b \\ (b-x)/(c-b);\ b \leq x \leq c \end{cases}$$

3. Fungsi Trapesium

Berbeda dengan fungsi segitiga, pada fungsi ini terdapat beberapa nilai x yang memiliki derajat keanggotaan sama dengan 1, yaitu ketika $b \leq x \leq c$. Tetapi, derajat keanggotaan untuk $c \leq x \leq d$ memiliki karakteristik yang sama dengan fungsi segitiga. Grafik dan notasi matematika dari fungsi Trapesium adalah sebagai berikut :



Gambar 2.19 Graik fungsi trapesium [10]

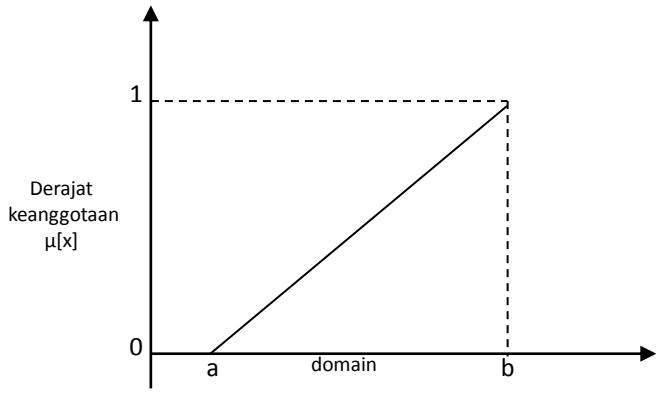
Fungsi Keanggotaan :

$$\mu[x] = \begin{cases} 0; & x \leq a \text{ atau } x \geq d \\ (x-a)/(b-a); & a \leq x \leq b \\ 1; & b \leq x \leq c \\ (d-x)/(d-c); & c \leq x \leq d \end{cases}$$

4. Fungsi Linear

Pada fungsi linear, pemetaan input ke derajat keanggotaannya digambarkan sebagai suatu garis lurus. Bentuk kurva ini paling sederhana dan menjadi pilihan yang baik untuk mendekati suatu konsep yang kurang jelas. Ada 2 keadaan himpunan fuzzy yang linear. Pertama, kenaikan himpunan dimulai pada

nilai domain yang memiliki derajat keanggotaan nol [0] bergerak ke kanan menuju ke nilai domain yang memiliki derajat keanggotaan lebih tinggi

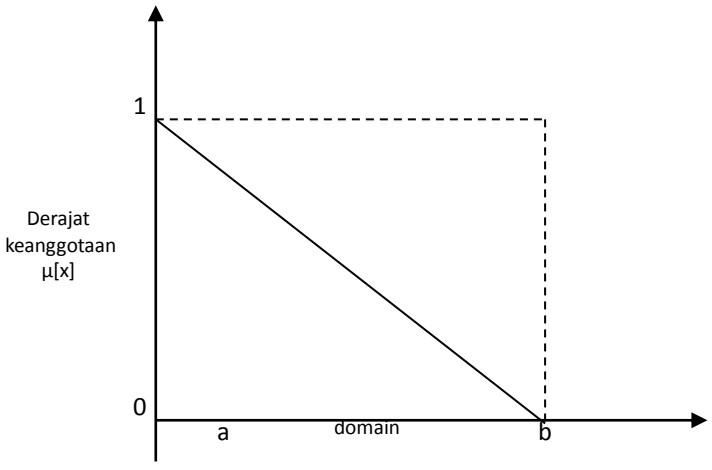


Gambar 2.20 Representasi Linear Naik [10]

Fungsi keanggotaan :

$$\mu[x] = \begin{cases} 0; & x < a \\ (x-a)/(b-a); & a < x \le b \\ 1; & x \ge b \end{cases}$$

Kedua, merupakan kebalikan dari yang pertama. Garis lurus dimulai dari nilai domain dengan derajat keanggotaan tertinggi pada sisi kiri, kemudian bergerak menurun ke nilai domain yang memiliki derajat keanggotaan lebih rendah .



Gambar 2.21 Representasi Representasi Linear Turun [10]

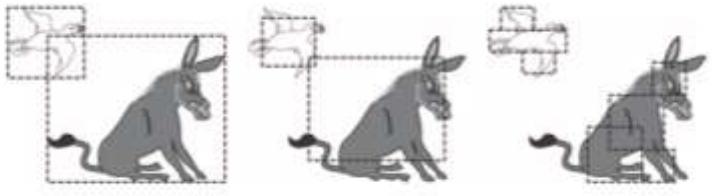
Fungsi Keanggotaan :

$$\mu[x] = \begin{cases} (b-x)/(b-a); & a \le x < b \\ 0; & x \ge b \end{cases} \quad \text{(II.6)}$$

2.2.5 Metode Collision detection

Hampir setiap permainan video yang ada sudah menerapkan *collision detection* (deteksi tabrakan), baik itu dalam hal tabrakan antara sprite dengan sprite maupun antara sprite dengan peluru dan lain-lain. Dalam aplikasi permainan ada beberapa reaksi yang ditunjukkan dari *collision detection*, yaitu reaksi dari spirte yang berupa berkurangnya nyawa, berkurangnya darah, meledak, dan lain-lain.

Proses *collision* dapat dibagi menjadi dua kategori dasar, yaitu *collision detection* dan *collision* response[12], dengan jarak respon yang telah diaplikasikan secara spesifik. Dalam *collision detection*, terdapat banyak sekali jenis dari *collision detection* itu sendiri, pada gambar 2.22 merupakan tiga tipe dari *collision detection*.



Gambar 2.22 Tiga jenis collision detection [19]

Setiap objek diberi sebuah kotak sebagai acuan jika *collision* itu terjadi. Untuk gambar yang pertama, *collision detection* terjadi ketika objek buruh bertabrakan dengan kotak yang memuat objek keledai, pada jenis *collision detection* ini rentan terhadap ketidaktelitian[11], untuk gambar yang kedua, *collision detection* terjadi ketika kotak dari kedua objek tersebut bersinggungan, pada jenis *collision detection* ini sudah lebih akurat dibanding jenis yang pertama, namun untuk memilih factor reduksi yang cocok itu sulit[11], dan untuk gambar yang ketiga, *collision detection* akan terjadi ketika objek saling bersinggungan

tanpa memperdulikan kotak secara keseluruhan, pada jenis *collision detection* ini adalah yang paling akurat dibandingkan dengan jenis yang lainnya[11].

Selain itu pada sumber lain[12] disebutkan bahwa *collision detection* terbagi menjadi dua jenis, yaitu *collision detection* object to object dan *collision detection* object to world. Pada object to object yang terjadi adalah adanya tubrukan antara satu objek dengan objek lainnya yang berada di dalam satu frame, baik itu objek tersebut bertubrukan secara langsung ataupun bertubrukan antara kotak maupun lingkaran yang mengandung objek tersebut, sedangkan pada object to world yang terjadi adalah adanya tubrukan antara objek dengan frame itu sendiri, contohnya adalah tubrukan antara sprite dengan dinding pada aplikasi permainan itu sendiri sehingga apabila terjadi, maka sprite tersebut tidak dapat melewati dinding tersebut.

2.3 Object Oriented Programming (OOP)

Object Oriented Programming (OOP) menerapkan sifat yang lebih modular agar setiap program dapat lebih mudah dikembangkan. Dalam OOP dibutuhkan memori lebih besar dibandingkan dengan program procedural (tradisional)[13]. 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 (abstr*ACTION*), pengkapsulan (encapsulation), penurunan sifat (inheritance), dan polimorfisme (polymorphism) pada objek-objek yang dibentuk.

Object Oriented Programming (OOP) dibagi menjadi beberapa ciri utama[13], 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[13], 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 (*Class*es) 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 dimplementasikan 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[13].

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*[13].

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*)[13].

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*)[13]. Akan tetapi hal itu tentunya bergantung juga pada access specifier (yaitu, public dan private) yang diberikan dalam proses penurunan kelas.

2.4 Unified Modelling Language (UML)

Pada perkembangan teknik pemrograman berorientasi objek, muncul sebuah standarisasi bahasa pemodelan untuk pembangunan perangkat lunak yang dibangun dengan menggunakan teknik pemrograman berorientasi objek, yaitu *Unified Modelling Language* (UML). Adapun pengertian dari UML adalah salah satu standar bahasa yang banyak digunakan di dunia industri untuk mendefinisikan requirement, membuat analisis dan desain, serta menggambarkan arsitektur dalam pemrograman berorientasi objek[4].

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.

2.4.1 Sejarah UML

Bahasa pemrograman berorientasi objek yang pertama dikembangkan dikenal dengan nama Simula-67 yang dikembangkan pada tahun 1967. Bahasa pemrograman ini kurang berkembang dan dikembangkan lebih lanjut, namun dengan kemunculannya telah memberikan sumbangan yang besar pada developer pengembang bahasa pemrograman berorientasi objek selanjutnya.

Perkembangan aktif dari pemrograman berorientasi objek mulai menggeliat ketika berkembangnya bahasa pemrograman Smalltalk pada awal 1980-an yang kemudian dikuti dengan perkembangan bahasa pemrograman berorientasi objek yang lainnya seperti C objek, C++, Eiffel, dan CLOS. Secara actual, penggunaan bahasa pemrograman berorientasi objek pada saat itu masih

terbatas, namun telah banyak menarik perhatian di saat itu. Sekitar lima tahun setelah Smalltalk berkembang, maka berkembang puka metode pengembangan berorientasi objek. Metode yang pertama diperkenalkan oleh Sally Shalaer dan Edward Yourdon, diikuti oleh Grady Booch, James R. Rumbaugh, Michael R. Blaha, William Lorensen, Frederick Eddy, William Premerlani, dan masih banyak lagi.

Karena banyaknya metodologi-metodologi yang berkembang pesat saat itu, maka muncullah ide untuk membuat sebuah bahasa yang dapat dimengerti semua orang. Usaha penyatuan ini banyak mengambil dari metodologi-metodologi yang berkembang pada saat itu. Maka dibuat bahasa yang merupakan gabungan dari beberapa konsep seperti konsep Object Modelling Technique (OMT) dari Rumbaugh dan Booch, konsep The *Class*es, Responsibilities, Collaborators (CRC) dari Rebecca Wirfs-Brock, konsep pemikiran Ivar Jacobson, dan beberapa konsep lainnya dimana James R. Rumbaugh, Grady Booch, dan Ivan Jacobson bergabung dalam sebuah perusahaan yang bernama Rational *Software* Corporation menghasilkan bahasa yang disebut dengan Unified Modeling Language (UML).

Pada 1996, Object Management Group (OMG) mengajukan proposal agar adanya standarisasi pemodelan berorientasi objek dan pada bulan September 1997 UML diakomodasi oleh OMG sehingga sampai saat ini UML telah memberikan kontribusinya yang cukup besar di dalam metodologi berorientasi objek dan hal-hal hanya terkait di dalamnya.

2.4.2 Diagram UML

UML menggunakan berbagai macam diagram dengan fungsi masing-masing untuk menggambarkan setiap proses dari sistem berorientasi objek. Berikut merupakan beberapa diagram UML diantaranya[14] :

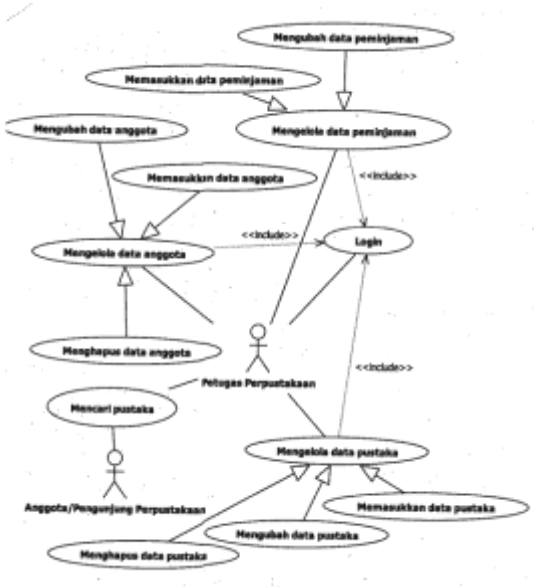
1. *Use Case Diagram*

Use Case atau diagram *Use Case* merupakan pemodelan yang digunakan untuk menggambarkan kelakuan (behavior) dari sistem yang akan dibuat[14]. *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*[14].

1. Aktor merupakan orang, proses, atau sistem lain yang berinteraksi dengan sistem yang akan dibuat di luar 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 2.23.



Gambar 2.23 Contoh dari *Use Case Diagram [14]*

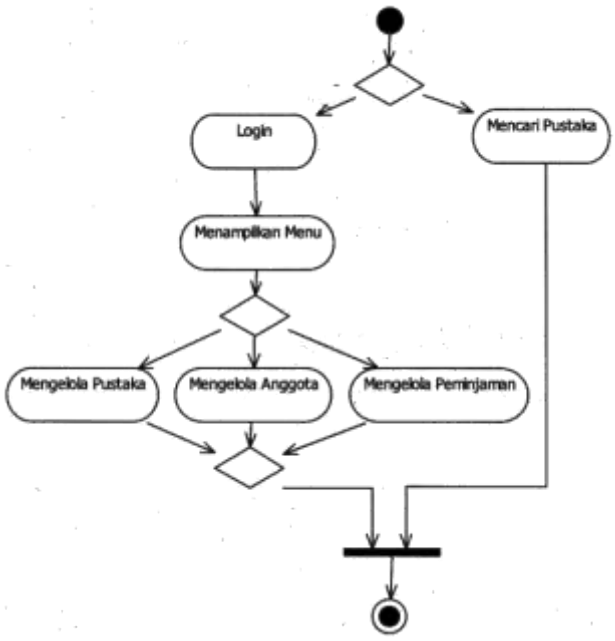
B. *Activity Diagram*

Diagram aktivitas atau *Activity Diagram* adalah sebuah diagram yang menggambarkan workflow (aliran kerja) atau aktivitas dari sebuah sistem atau proses bisnis. Dalam diagram aktivitas yang perlu diperhatikan adalah bahwa diagram aktivitas menggambarkan aktivitas sistem, bukan apa yang dilakukan aktor, jadi aktivitas yang dapat dilakukan oleh sistem.

Diagram aktivitas juga banyak digunakan untuk mendefinisikan hal-hal berikut[14] :

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 2.24.



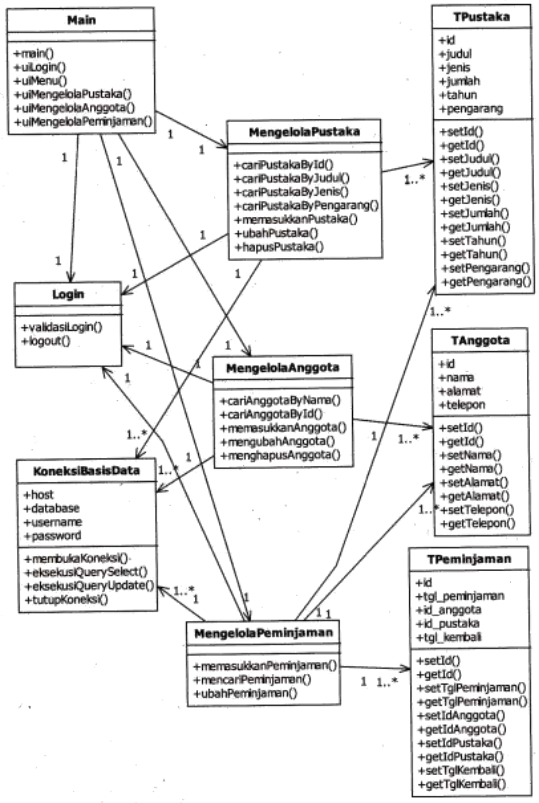
Gambar 2.24 Contoh dari *Activity Diagram [14]*

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[14].

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 2.25.



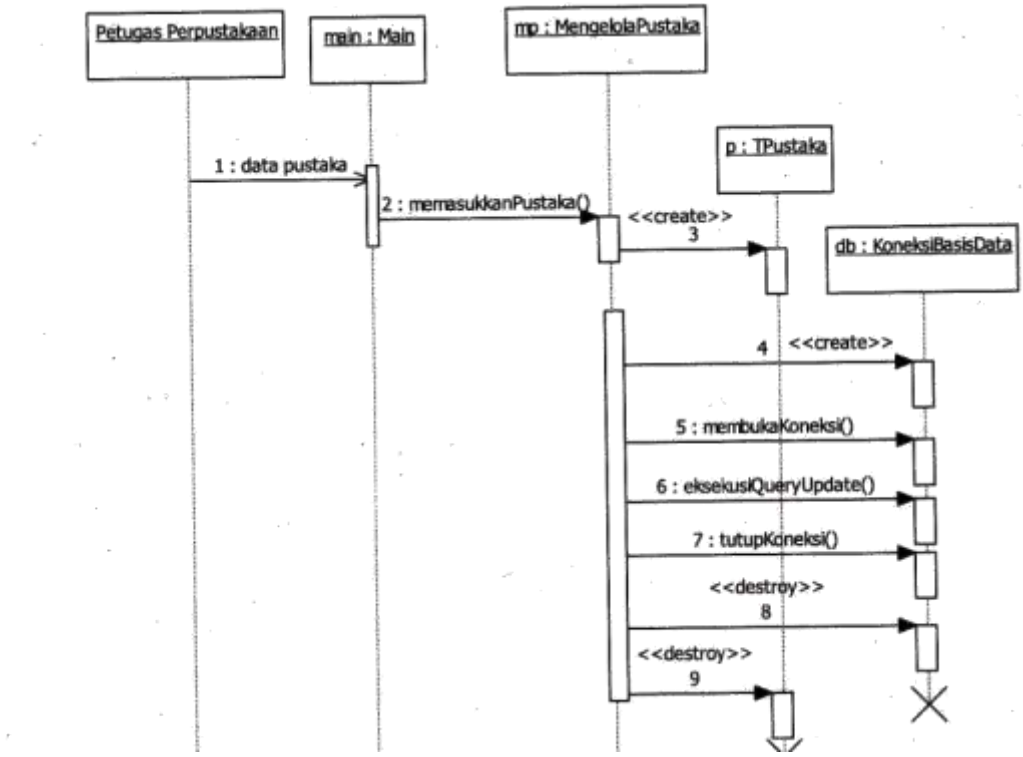
Gambar 2.25 Contoh dari *Class Diagram [14]*

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[14]. 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 2.26.



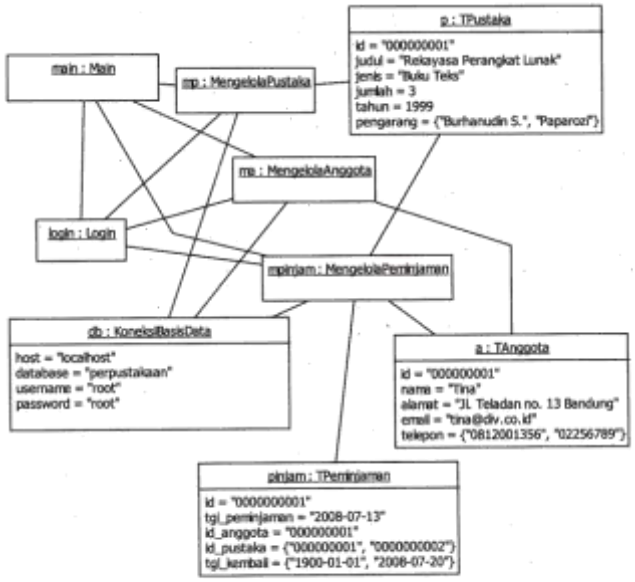
Gambar 2.26 Contoh dari *Sequence Diagram [14]*

E. *Object Diagram*

Diagram objek menggambarkan struktur system dari segi penamaan objek dan jalannya objek dalam sistem[14]. 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 2.27.



Gambar 45 Diagram Objek Studi Kasus

Gambar 2.27 Contoh dari *Object Diagram [14]*

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[14]. Diagram komponen juga dapat digunakan untuk memodelkan hal-hal berikut[14] :

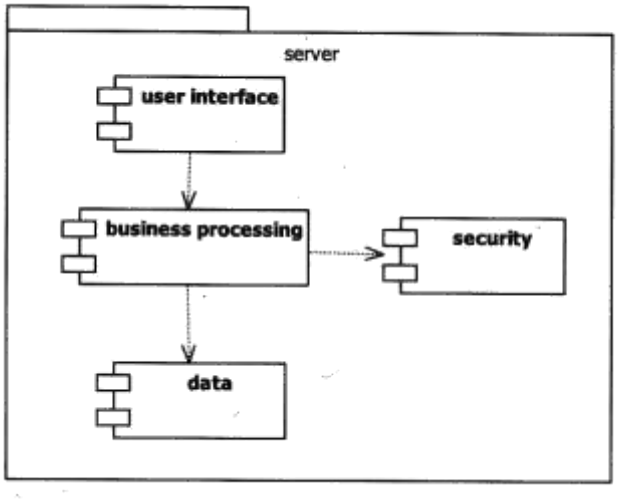
1. Source code program perangkat lunak
2. Komponen executable yang dilepas ke user
3. Basis data secara fisik
4. Sistem yang harus beradaptasi dengan sistem lain
5. Framework sistem

Adapun komponen-komponen dasar yang biasanya ada dalam suatu sistem adalah sebagai berikut[14] :

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 2.28.



Gambar 2.28 Contoh dari *Component Diagram [14]*

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[14]. 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[14].

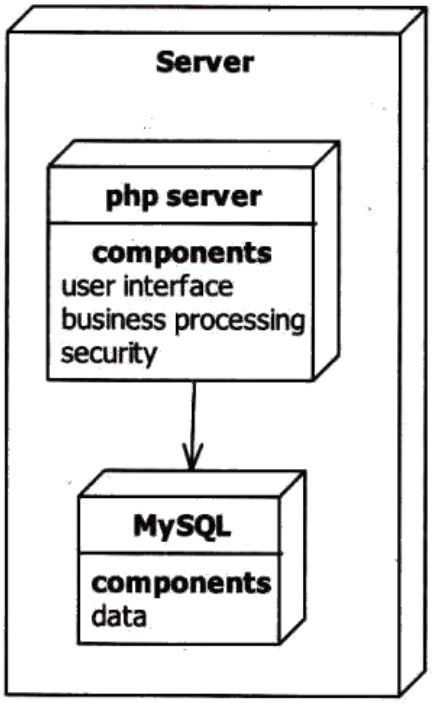
I. *Deployment Diagram*

Diagram deployment atau *Deployment Diagram* menunjukkan konfigurasi komponen dalam proses eksekusi aplikasi[14]. 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 2.29.



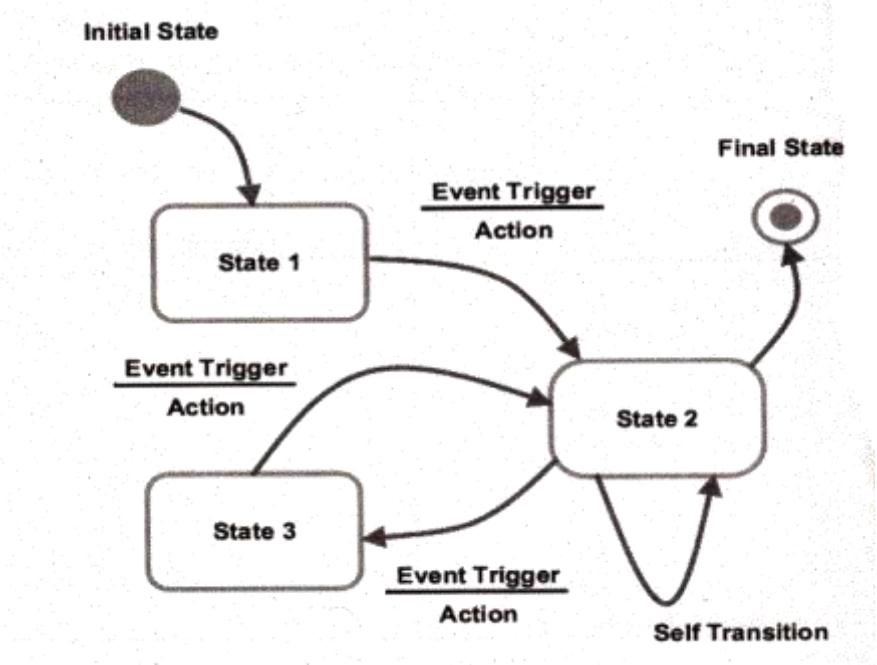
Gambar 2.29 Contoh dari *Deployment Diagram [14]*

J. *State Machine Diagram*

Diagram mesin status digunakan untuk menggambarkan perubahan status atau transisi status dari sebuah mesin atau sistem[14]. 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[14].

Contoh dari *State Machine Diagram* dapat dilihat pada gambar 2.30.



Gambar 2.30 Contoh dari *State Machine Diagram [14]*

K. *Communication Diagram*

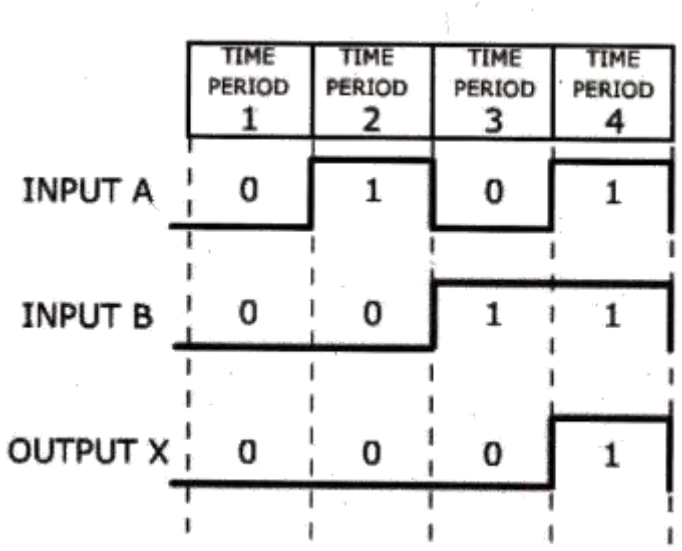
Diagram komunikasi merupakan penyederhanaan dari diagram kolaborasi (collaboration diagram)[14]. Diagram ini menggambarkan interaksi antarobjek/bagian dalam bentuk urutan pengiriman pesan. Diagram komunikasi merepresentasikan informasi yang diperoleh dari Diagram Kelas, Diagram Sekuen, dan Diagram *Use 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[14]. *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 2.31.



Gambar 2.31 Contoh dari *Timing Diagram [14]*

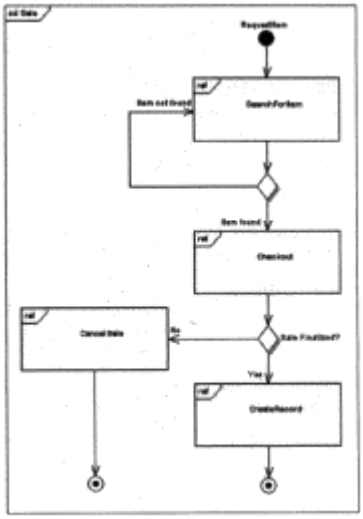
M. *Iteraction Overview Diagram*

Iteraction Overview Diagram mirip dengan diagram aktivitas yang berfungsi untuk menggambarkan sekumpulan urutan aktivitas. Iterr*action Over*view diagram adalah bentuk aktivitas diagram yang setiap titik merepresentasikan diagram interaksi[14]. Interaksi diagram dapat meliputi diagram sekuen, diagram komunikasi, Inter*action Over*view diagram, dan *Timing Diagram*.

Hampir semua notasi pada Inter*action Over*view diagram sama dengan notasi pada diagram aktivitas. Sebagai contoh initial, final, decision, merge, fork,

dan join nodes sama seperti pada diagram aktivitas. Tambahan pada Inter*action Over*view diagram adalah *Interaction Accurrence* dan *Interaction Element*.

Contoh dari *Overview Diagram* dapat dilihat pada gambar 2.32.



Gambar 2.32 Contoh dari *Iteraction Overview Diagram [14]*

2.5 Teknik Pengujian Perangkat Lunak

Pengujian perangkat lunak adalah elemen kritis dari jaminan kualitas perangkat lunak dan merepresentasikan kajian pokok dari spesifikasi, desain, dan pengkodean[15].

2.5.1 Pengujian *Black box*

Menurut Roger S. Pressman[15], pengujian *black box* berfokus pada persyaratan fungsional perangkat lunak. Dengan demikian, pengujian *black box* menungkinkan perekayasa perangkat lunak mendapatkan serangkaian kondisi input yang sepenuhnya menggunakan semua persyaratan fungsional untuk suatu program. Pengujian *black box* bukan merupakan alternatif dari teknik *White Box*, tetapi merupakan pendekatan komplementer yang kemungkinan besar mampu mengungkap kelas kesalahan daripada metode *White Box*.

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

A. Fungsi-fungsi yang tidak benar atau hilang
B. Kesalahan dalam *interface*

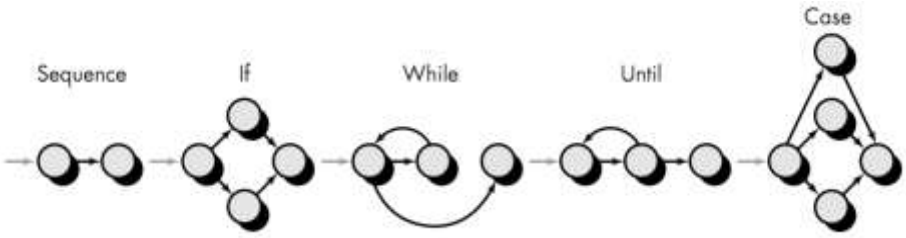
C. Kesalahan dalam struktur data atau akses database eksternal
D. Kesalahan kinerja
E. Inisialisasi dan kesalahan terminasi

2.5.2 Pengujian *White Box*

Menurut Roger S. Pressman[15], pengujian *White Box*, yang kadang-kadang disebut pengujian glass box, adalah metode desain test case yang menggunakan struktur kontrol desain procedural untuk memperoleh test case. Pengujian *whitebox* dilakukan untuk menguji prosedur-prosedur yang ada. *Logic path* yang dilalui oleh setiap bagian prosedur diuji dengan memberikan kondisi/loop spesifik. Pengujian *whitebox* menjamin pengujian terhadap semua lintasan yang tidak bergantungan minimal satu kali, mencoba semua keputusan *logic* dari sisi '*true*' dan '*false*', eksekusi semua *loop* dalam batasan kondisi dan batasan operasionalnya dan pengujian validasi data internal. Konsep Pengujian Basis *Path* sebagai berikut*:*

1. Merupakan bagian dari pengujian *whitebox* dalam hal pengujian prosedur-prosedur.
2. Mempergunakan notasi aliran *graph* (*node, link* untuk merepresentasikan *sequence, if, while, until* dan sebagainya).
3. Konsep kompleksitas *cyclomatic* antara lain cara perhitungan daerah tertutup pada graph planar dimana dapat menghubungkan batas atas jumlah pengujian yang harus direncanakan dan dieksekusi untuk menjamin pengujian seluruh *statement* program.
4. Memunculkan kasus-kasus yang akan diuji dengan membuat daftar lintasan kasus pengujian berdasarkan kompleksitas dan *cyclomatic* yang didapat.
5. Membuat alat bantu graph matriks yang membantu pengawasan pengujian.

Pengujian *White Box* yang berupa notasi diagram alir dapat dilihat pada gambar 2.33.



Gambar 2.33 Notasi Diagram Alir [15]

2.5.2 Pengujian Kuesioner

Angket atau kuesioner merupakan suatu teknik pengumpulan data secara tidak langsung (peneliti tidak langsung bertanya jawab dengan responden). Instrumen atau alat pengumpulan datanya juga disebut angket berisi sejumlah pertanyaan-pertanyaan yang harus dijawab atau direspon oleh responden. Responden mempunyai kebebasan untuk memberikan jawaban atau respon sesuai dengan presepsinya.[20]

Menurut Sugiyono [20], Skala *Likert* digunakan untuk mengukur sikap, pendapat dan persepsi seseorang atau sekelompok orang tentang fenomena sosial. Dalam penelitian fenomena sosial ini telah ditetapkan secara spesifik oleh peneliti yang selanjutnya disebut sebagai variabel penelitian. Dengan skala *likert*, maka variabel yang akan diukur dijabarkan menjadi indikator variabel. Kemudian indikator tersebut dijadian sebagai titik tolak untuk menyusun item-item instrumen yang dapat berupa pernyataan atau pertanyaan. Jawaban setiap item instrumen yang menggunakan skala likert mempunyai gradasi dari sangat positif sampai sangat negatif.

Jawaban setiap item instrumen yang menggunakan skala *Likert* mempunyai gradasi dari sangat positif sampai negatif yang dapat berupa kata-kata antara :

1. Sangat setuju
2. Setuju
3. Biasa Saja

1. Setuju
2. Sering
3. Kadang-kadang

4. Kurang Setuju 4. Hampir tidak pernah
5. Tidak Setuju 5. Tidak pernah

Untuk keperluan analisis kuantitatif, maka jawaban itu dapat diberi skor, misalnya :

1	Sangat Setuju/Selalu/Sangat positif diberi skor	5
2	Setuju/Sering/Positif diberi skor	4
3	Ragu – Ragu/ kadang – kadang / Netral diberi skor	3
4	Tidak Setuju/ Hampir tidak pernah / Negatif diberi skor	2
5	Sangat Tidak Setuju / Tidak Pernah/Sangat Negatif diberi skor	1

2.6 Construct 2

Construct 2 adalah tools yang digunakan dalam membuat sebuah game tanpa memerlukan pengetahuan tentang pemrograman. caranya hanya melakukan drag and drop item yang tersedia, menambahkan perilaku mereka, dan membuat mereka menjadi bergerak dengan sebuah event.

Construct 2 memiliki interface yang sangat mirip dengan produk microsoft office. hal ini membuat mudah bagi orang-orang yang terbiasa dengan produk office. karena sifat HTML 5 masih eksperimental, ada beberapa fitur dalam pembuatan game yang mungkin tidak bekerja dengan baik di semua perangkat. Construct 2 merupakan *tools* yang dapat dipelajari dengan cepat, dalam pembuatan *game* dapat dilakukan dengan mengekspor permainan sendiri dan bekerja dalam *mobile device* [16].

2.7 PhoneGap

PhoneGap adalah sebuah *open source development tool* diciptakan oleh Nitobi yang berfungsi sebagai penghubung antara aplikasi berbasis HTML dan aplikasi berbasis mobile [17].PhoneGap bekerja dengan menggabungkan HTML5 Canvas dan JavaScript. Saat ini PhoneGap mendukung beberapa sistem operasi perangkat *mobile* terkemuka yaitu Android, iOS dan Blackberry. Pembuatan aplikasi menjadi lebih efisien dengan menggunakan PhoneGap dikarenakan cukup

menggunakan HTML5 Canvas dan JavaScript untuk aplikasi pada Android, iOS dan Blackberry.

2.8 Android

Android adalah sebuah system operasi untuk perangkat mobile berbasis linux yang mencakup system operasi, *middleware* dan aplikasi.Android menyediakan platform terbuka bagi para pengembang untuk menciptakan aplikasi mereka.Android merupakan generasi baru *platform mobile*, *platform* yang memberikan pengembangan untuk melakukan pengembangan sesuai dengan yang diharapkan. Pengembangan aplikasi Android diperbolehkan untuk mendistribusikan aplikasi mereka di bawah skema lisensi apapun yang mereka inginkan.Pengembang memiliki beberapa pilihan ketika membuat aplikasi yang berbasis android. Sebagian besar pengembang menggunakan *eclipse* yang tersediasecara bebas untuk merancang dan mengembangkan aplikasi android, karena memiliki android *plug-in* yang tersedia untuk memfasilitasi pengembangan android. Selain itu, *eclipse* juga mendapat dukungan langsung dari google untuk menjadi IDE pengembangan aplikasi Android,ini terbukti dengan adanya penambahan *plug-in* untuk *eclipse* untuk membuat *project* android dimana *source software* langsung dari situs resminya google [18].