BAB 2

TINJAUAN PUSTAKA

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 [4].

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 [4].

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.

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 . 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 karean 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.

2.1.4 Jenis-jenis *Game*

Berikut ini beberapa jenis *game* berdasarkan cara pembuatannya, cara pemasarannya dan mesin yang menjalankannya [5]. Jenis-jenis *game* tersebut adalah :

1. *Game* PC

   Game PC adalah *game* yang dimainkan pada PC (*Personal Computer*) yang memiliki kelebihan yaitu tampilan antarmuka yang baik untuk *input* maupun *output*. *Output* visual berkualitas tinggi karena layar komputer biasanya memiliki resolusi yang jauh lebih tinggi dibandingkan dengan layar televisi biasa. Kekurangannya adalah spesifikasi komputer yang sangat bervariasi antar satu komputer dengan komputer yang lainnya menyebabkan beberapa *game* dapat ditampilkan dengan baik pada satu komputer tetapi tidak berjalan dengan baik pada komputer yang lainnya.
2. *Game Console*

   Game console adalah *game* yang dijalankan pada suatu mesin spesifik yang biasanya tersedia di rumah seperti Xbox, Nintendo Wii dan lain-lain.
3. *Game Arcade*

Game arcade adalah *game* yang dijalankan pada mesin dengan *input* dan *output* audio visual yang telah terintegrasi dan tersedia ditempat-tempat umum.

4. *Game Online*

Game online adalah *game* yang hanya dapat dumainkan secara *online* melaui LAN atau internet.

2.1.5 *Genre Game*

Berdasarkan *genre*, *game* dapat dibagi menjadi beberapa *genre* yaitu [5] :

1. *Action Game*

Action game dikategorikan sebagai *game play* dengan model pertarungan. Berikut beberapa macam *game* yang termasuk dalam *genre action game* yaitu :

a. *Action Adventure Game*

Genre game yang berfokus pada eksplorasi dan biasanya mempunyai unsur *item gathering*, penyelesaiaan *puzzle simple* dan pertarungan. Contoh *game* dari *genre* ini adalah The Legend Of Zelda series dan Metroid series.

b. *Stealth Game*

Termasuk dalam *genre* terbaru, biasanya digolongkan dalam mata-mata yang bias melakukan aksinya secara rahasia. Contoh *game* dari *genre* ini adalah Metal Gear series.

c. *Survival Horror Game*

Genre game yang berusaha membuat pemain menjadi tegang dan takut dengan elemen-elemen horror. Contoh *game* dari *genre* ini adalah Resident Evil series dan Alone in The Dark.

d. *Beat'em Up Game*

Genre game combat dimana satu orang melawan banyak musuh yang telah disediakan. Contoh *game* dari *genre* ini adalah Dynasty Warrior series dan Final Fight.

e. *Fighting Game*

Game pertarungan dua pemain dengan jurus-jurus yang biasa dikeluarkan dengan menekan beberapa tombol pada *keyboard* dengan urutan tertentu. Contoh *game* dari *genre* ini adalah Street Fighter dan Tekken series.

f. *Maze Game*

Genre game yang membutuhkan kecepatan berpikir dan bereaksi serta berunsur ketepatan menavigasi. Contoh *game* dari *genre* ini adalah Pac-Man.

Platfromers Game

Genre Game dengan *game play* berlari, melompat, mengayun dan sebagainya. Contoh *game* dari *genre* ini adalah Donkey Kong dan Ray Man.

g. *Shooter*

a) *First Person Shooter Game*

Genre game yang mengutamakan *shooting* dan *combat* dari perspektif langsung mata karakter yan bertujuan untuk memberikan pemain perasaan berada ditempat itu dan bisa fokus menembak.

b) *Massively Multiplayer Online First Person Shooter Game*

Genre game yang mengkombinasikan *game play first person shooter* dengan dunia virtual dimana banyak *player* juga ikut bermain melaui internet. Contoh *game* dari *genre* ini adalah Counter Strike *Online*.

c) *Third Person Shooter Game*

Genre game yang sama seperti *first person shooter game* yaitu mengutamakan *shooting* dan *combat* dari perspektif karakter yang bertujuan untuk memberikan pemandangan yang lebih luas dan gerakan yang lebih banyak.

d) *Tactical Shooter Game*

Genre yang mengutamakan perencanaan dan kerja sama tim untuk memenangkan *game*. Contoh *game* dari *genre* ini adalah Tom Clancy's Ghost Recon series.

e) *Light Gun Game*

Genre dengan lebih banyak pada *arcade* dengan peralatan tertentu seperti senjata mainan yang mempunyai sensor khusus terhadap layar. Contoh *game* dari *genre* ini adalah Time Crisis dan Duck Hunt.

f) *Shoot'em Up Game*

Genre dengan ciri khas gambar 2D dan *scrolling playing area*. Contoh *game* dari *genre* ini adalah Star Fox series.

2. *Adventure Game*

Adventure game dikategorikan sebagai *game play* yang mengharuskan pemain memecahkan bermacam-macam teka-teki melaui interaksi dengan orang lingkungan dalam *game* tersebut.

a. *Text Adventure*

Pemain akan menggunakan *keyboard* untuk mengetikkan berupa perintah dan komputer akan menganalisa perintah tersebut lalu menjalankan karakter sesuai perintah tersebut.

b. *Graphical Adventure Game*

Genre yang merupakan perkembangan dari *text adventure*. Pemain dapat menggunakan *mouse* untuk menggerakkan karakter.

c. *Visual Novel Game*

Genre yang memberikan keleluasaan untuk memilih jalan ceritanya sendiri.

d. *Interactive Movie Game*

Genre game dengan rangkaian *live action* dari karakter yang dimainkan pemain. Contoh *game* dari *genre* ini adalah Space Ace.

e. *Dialog Game*

Pada *genre* ini, pemain akan mengalami kemajuan tergantung pada apa yang mereka katakan. Contoh *game* dari *genre* ini adalah Law And Order : The Vangeful Heart.

3. *Role Playing Game*

Role playing game adalah *game* yang memiliki *game play* dimana karakter milik *player* akan berpetualang dengan *skill combat* dalam cerita *game*.

a. *Action Role Playing Game*

Genre game yang memasukkan unsur *action game* dan *action adventure game*. Contoh *game* dari *genre* ini adalah Diablo 1 & 2.

b. *Massively Multyplayer Online Role Playing Game*

Konsep dari *genre* ini terkombinasi dengan *genre-genre* lainnya yang berupa fantasi. Contoh *game* dari *genre* ini adalah Rising Force *Online*.

c. *Tactical Role Playing Game*

Dalam *genre* ini, pemain akan diberikan giliran masing-masing untuk menentukan langkah-langkah yang akan dilakukan oleh karakter. Contoh *game* dari *genre* ini adalah Final Fantasy Tactics.

4. *Simulation Game*

Genre ini bertujuan untuk memberikan pengalaman simulasi kepada pemain.

a. *Construction and Management Simulation Game*

Genre ini merupakan bagian dari *economic simulation game*. Contoh *game* dari *genre* ini adalah Sims City series.

b. *Economic Simulation Game*

Genre ini berupa simulasi keadaan ekonomi dimana pemain mengontrol keadaan ekonomi dari *game* tersebut. Contoh dari *genre* ini adalah Monopoly Tycoon.

c. *God Game*

Dalam *genre* ini tidak ada tujuan akhir yang membuat pemain memenangakan *game*. Contoh *game* dari *genre* ini adalah The Sims series.

d. *Government Simulation Game*

Genre game yang memasukkan unsur kepolisian, pemerintahan atau politik sebuah negara.

5. *Strategy Game*

Genre strategy game berfokus pada *game play* dimana dibutuhkan pemikiran yang tepat agar dapat meraih kemenangan.

a. *Real Time* Strategi

Dalam *real time* strategi, *action* dilakukan dalam waktu yang bersamaan oleh masing-masing pihak dimana *action* dimainkan per*ronde* atau bergiliran. Contoh *game* dari *genre* ini adalah Warcraft series.

b. *Tactical Game*

Dalam *genre* ini pemain harus menggunakan bermacam-macam taktik dan strategi untuk mencapai kemenangan. Contoh *game* dari *genre* ini adalah Dark Omen.

c. *4X Game*

Genre ini berarti penjelajahan, menjajah dan memusnahkan. Contoh *game* dari *genre* ini adalah Galactic Civizations.

d. *Artillery Game*

Genre game ini biasanya mengikutkan *combat* dengan tank atau tentara militer. Contoh *game* dari *genre* ini adalah Tanarus.

6. *Vehicle Simulation Game*

Genre ini merupakan simulasi yang memberikan pemain sebuah pengalaman realistik dalam mengendarai kendaraan tertentu.

a. *Flight Game*

Dalam *genre* ini,pemain tidak hanya bersimulasi mengontrol pesawat terbang tetapi juga bisa *combat* di udara. Contoh *game* dari *genre* ini adalah Falcon 4.0.

b. *Racing Game*

Genre yang menempatkan pemain sebagai *driver* dengan kendaraan seperti mobil. Contoh *game* dari *genre* ini adalah Need For Speed series.

c. *Space Game*

Genre ini bersifat pertarungan di angkasa luar. Contoh *game* dari genre ini adalah Star Wars dan Homeworld.

d. *Train Game*

Genre ini mensimulasikan yang berhubungan dengan trasnportasi kereta. Contoh dari *genre* ini adalah Rail Simulator.

2.1.6 Unsur *Game*

Dalam sebuah *game* terdapat unsur-unsur yang melengkapinya. Berikut beberapa unsur dalam suatu *game* yaitu [5]:

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. 1 Warna *Primer* dan *Sekunder* [5]

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. 2 Bentuk Dasar [5]

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.

2.1.7 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*.[15]

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.[15]

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.[15]

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.[15]

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. [15]

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.[15]

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.[15]

7. RP (*Rating Pending*) telah disampaikan kepada ESRB dan sedang menunggu penilaian akhir. (Simbol ini hanya muncul dalam iklan sebelum permainan dibebaskan.[15]

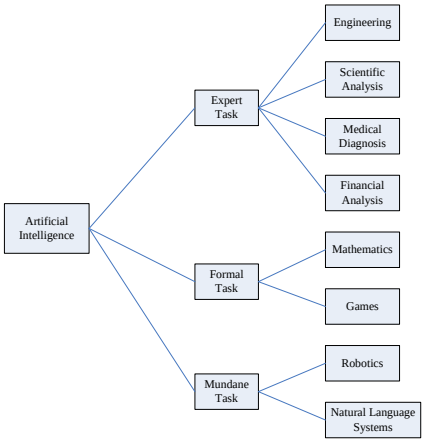
2.2 Kecerdasan Buatan

Kecerdasan buatan atau *Artificial Intelligence* (AI) merupakan cabang dari ilmu komputer yang berhubungan dengan pengautomatisan tingkah laku cerdas. Kecerdasan buatan didasarkan pada teori suara (*sound theoretical*) dan prinsip-prinsip aplikasi dari bidangnya [6]. Prinsip-prinsip ini meliputi struktur data yang digunakan dalam representasi pengetahuan, algoritama yang diperlukan untuk mengaplikasikan pengetahuan tersebut serta bahasa dan teknik pemrograman yang digunakan dalam mengimplementasikannya.

Berdasarkan sudut pandang, AI dapat dipandang sebagai berikut :

1. Sudut pandang kecerdasan, AI adalah bagaimana membuat mesin yang cerdas dan dapat melakukan hal-hal yang sebelumnya hanya dapat dilakukan manusia.
2. Sudut pandang bisnis, AI adalah sekelompok alat bantu yang berdayaguna dan metodologi yang menggunakan alat-alat bantu tersebut untuk menyelesaikan masalah-masalah bisnis.
3. Sudut pandang pemrograman, AI meliputi studi tentang pemrograman simbolik, pemecahan masalah dan proses pencarian.
4. Sudut pandang penelitian :
   a. Riset tentang AI dimulai pada awal tahun 1960-an, percobaan pertama adalah membuat program permainan catur, membuktikan teori dan *general problem solving*.
   b. AI adalah nama pada akar dari studi area.

Kecerdasan buatan memiliki sejumlah sub disiplin ilmu yang sering digunakan untuk pendekatan yang esensial bagi penyelesaian suatu masalah dan dengan aplikasi bidang AI yang berbeda. Berikut gambar bidang-bidang tugas dari AI yaitu :



Gambar 2. 3 Bidang-bidang tugas (*task domains*) dari AI [6]

Aplikasi penggunaan AI dapat dibagi ke dalam tiga kelompok yaitu :

1. *Expert Task*

   AI dibentuk berdasarkan pengalaman dan pengetahuan yang dimiliki oleh para ahli. Penggunaan ini dapat membantu para ahli untuk menyampaikan ilmu-ilmu yang dimiliki. Contohnya adalah :

   a. Analisis finansial.
   b. Analisis medikal.
   c. Analisis ilmu pengetahuan.
   d. Rekayasa (desain, pencarian, kegagalan, perencanaan, manufaktur).

2. *Formal Task*

   AI digunakan untuk melakukan tugas-tugas formal yang selama ini manusia biasa lakukan dengan lebih baik. Contohnya adalah :

   a. *Game.*
   b. Matematika (geometri, logika, kalkulus, integral).

3. *Mundane Task*

   Secara harfiah *mundane* adalah keduniaan. AI digunakan untuk melalukan hal-hal yang sifatnya duniawi atau melakukan kegiatan yang dapat membantu manusia. Contohnya adalah :

   a. Persepsi.
   b. Bahasa alami.
   c. Robot control.

Aplikasi kecerdasan buatan memiliki dua bagian utama yaitu :

1. Basis Pengetahuan (*Knowledge Base*) : berisi fakta-fakta, teori, pemikiran dan hubungan antara satu dengan lainnya.
2. Motor Inferensi (*Inference Engine*) : kemampuan menarik kesimpulan berdasarikan pengalaman.

2.2.1 Perbandingan Kecerdasan Buatan dengan Kecerdasan Alamiah

Ada beberapa keuntungan dari kecerdasan buatan dibandingkan dengan kecerdasan alamiah yaitu :

1. Lebih permanen.
2. Memberikan kemudahan dalam duplikasi dan penyebaran.
3. Ralatif lebih murah dari kecerdasan alamiah.
4. Konsisten dan teliti.
5. Dapat didokumemtasikan.
6. Dapat mengerjakan beberapa task dengan lebih cepat dan lebih baik dibanding manusia.

2.2.2 Teknik Dasar Pencarian

Ada beberapa keuntungan dari kecerdasan buatan dibandingkan dengan kecerdasan alamiah yaitu :

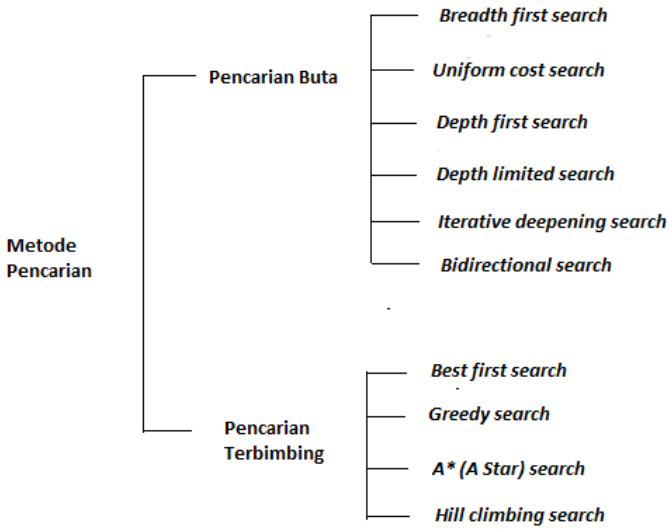
1. Lebih permanen.
2. Memberikan kemudahan dalam duplikasi dan penyebaran.
3. Ralatif lebih murah dari kecerdasan alamiah.
4. Konsisten dan teliti.
5. Dapat didokumemtasikan.
6. Dapat mengerjakan beberapa task dengan lebih cepat dan lebih baik dibanding manusia.

2.2.3 Algoritma Pencarian

Permasalahan pencarian dapat diselesaikan dengan beberapa metode yaitu [6]:

1. Metode pencarian yang pertama adalah metode sederhana yang hanya berusaha mencari kemungkinan penyelesaian yang disebut juga pencarian buta.
2. Metode yang lebih kompleks yang akan mencari jarak terpendek. Metode ini adalah *British Museum Procedure, Branch and Bound, Dynamic Programming, Best First Search, Greedy Search*, A* (A Star) dan *Hill Climbing Search*. Metode-metode ini digunakan pada saat perjalanan untuk mencari kemungkinan menjadi perhitungan.

Metode pencarian sangat penting untuk menyelesaikan permasalahan karena setiap *state* atau keadaan menggambarkan langkah-langkah untuk menyelesaikan permasalahan. Dalam sebuah permainan, metode pencarian akan menentukan apa yang harus dilakukan dimana setiap state menggambarkan kemungkinan posisi pada suatu saat. Metode pencarian adalah bagian dari kesimpulan dimana setiap state menggambarkan hipotesis dalam sebuah rangkaian deduktif.



Gambar 2. 4 Bagan Metode Pencarian [6]

Pada gambar di atas, metode penulusuran dibagi menjadi dua golongan, yakni pencarian buta *(blind search)* dan pencarian terbimbing *(heuristic search)*.

Beberapa contoh algoritma pencarian yang menggunakan metode *heuristic search* adalah : *Best First Search*, *Greedy Search*, A* *(A Star) Search*, dan *Hill Climbing Search*.

2.2.4 Pencarian Buta

Pencarian buta (*Blind Search*) adalah pencarian solusi tanpa adanya informasi yang dapat mengarahkan pencarian untuk mencapai *goal state* dari *current state* [6]. Informasi yang ada hanyalah definisi *goal state* itu sendiri

sehingga algoritma dapat mengenali *goal state*. Apabila tidak ada informasi maka pencarian buta dalam kerjanya akan memeriksa *node-node* secara tidak terarah dan kurang efisien untuk kebanyakan kasus karena banyanknya *node* yang dikembangkan. Beberapa contoh algoritma yang termasuk *blind search* adalah *Breadth First Search, Uniform Cost Search, Depth First Search, Depth Limited Search, Interative Deepening Search* dan *Bidirectional Search* [6].

2.2.5 Pencarian Terbimbing

Kata heuristik berasal dari kata kerja bahasa Yunani, heuriskein, yang berarti mencari atau menemukan. Dalam dunia pemograman, sebagian orang menggunakan kata heuristik sebagai lawan kata dari algoritmik, dimana kata heuristik ini diartikan sebagai suatu proses yang mungkin dapat menyelesaikan suatu masalah tetapi tidak ada jaminan bahwa solusi yang dicari selalu dapat ditemukan. Di dalam mempelajari metode – metode pencarian ini, kata heuristik diartikan sebagai suatu fungsi yang memberikan suatu nilai berupa biaya perkiraan (estimasi) dari suatu solusi [6].

Pencarian terbimbing (*Heuristic Search*) mempunyai informasi tentang biaya untuk mencapai *goal state* dari *current state*. Pencarian terbimbing dapat melakukan pertimbangan untuk mengembangkan atau memeriksa *node-node* yang mengarah ke *goal state*. Pencarian terbimbing untuk menghitung *cost* ke *goal state* digunakan fungsi *heuristic*. Fungsi *heuristic* berbeda dari pada algoritma dimana *heuristic* lebih merupakan perkiraan untuk membantu algoritma dan tidak harus valid setiap waktu. Beberapa contoh algoritma pencarian yang mengguanakan metode *heuristic search* adalah *Best First Search, Greedy Search*, A* (A Star) dan *Hill Climbing Search* [6].

2.2.6 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.

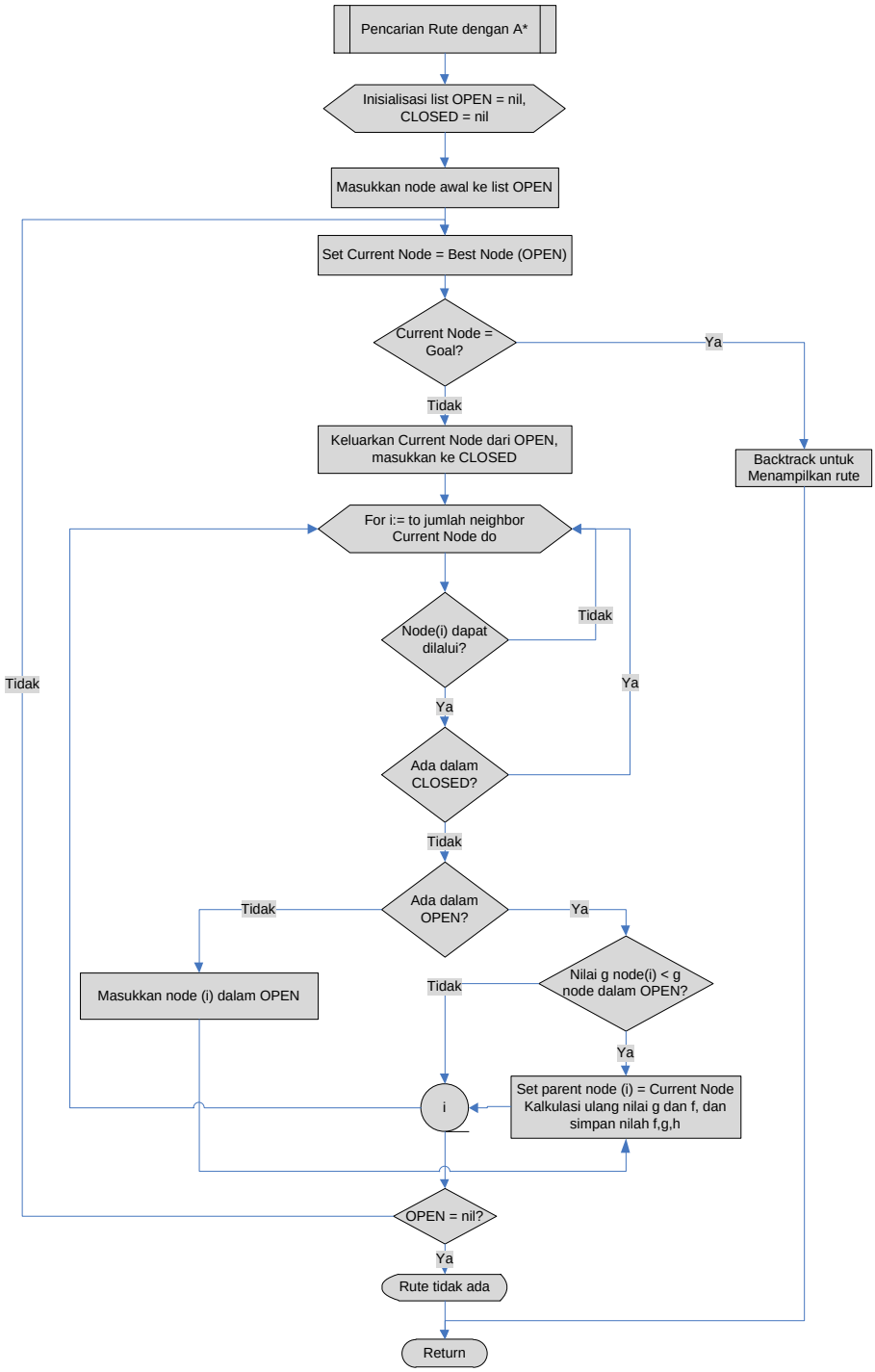
   d. Berhenti ketika:
      1. Menambahkan *target point* ke dalam *closed list*, dalam hal ini jalan telah ditemukan, atau,
      2. Gagal untuk menemukan *target point*, dan *open list* kosong. Dalam kasus ini, tidak ada jalan.
   e. Walaupun telah mencapai *target point*, jika masih ada *neighbor node* 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* pada gambar 2.5.

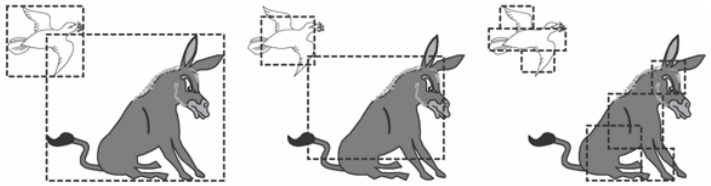


Gambar 2. 5 *Flowchart* Algoritma A* [7]

2.2.7 Metode Collision Detection

Hampir setiap *game* 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*[8], dengan jarak respon yang telah diaplikasikan secara spesifik. Dalam *collision detection*, terdapat banyak sekali jenis dari *collision detection* itu sendiri, pada gambar 2.6 merupakan tiga tipe dari *collision detection*.



Gambar 2. 6 Tiga jenis *collision detection*[8]

Setiap objek diberi sebuah kotak sebagai acuan jika *collision* itu terjadi. Untuk gambar yang pertama, *collision detection* terjadi ketika objek burung bertabrakan dengan kotak yang memuat objek keledai, pada jenis *collision detection* ini rentan terhadap ketidaktelitian[8], 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 faktor reduksi yang cocok itu sulit[8], 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[8].

Selain itu pada sumber lain[9] 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 OOP (Object Oriented Programming)

Objek adalah kesatuan entitas yang memiliki sifat dan tingkah laku. Dalam kehidupan sehari-hari, objek adalah benda, baik benda berwujud nyata seperti manusia, hewan, mobil, komputer, handphone, pena, ataupun benda yang tidak nyata atau konsep, seperti halnya tabungan bank, sistem antrian, sistem internet banking, dan sebagainya. Jadi pengertian OOP adalah konsep yang membagi program menjadi objek-objek yang saling berinteraksi satu sama lain. Objek adalah benda, baik benda yang berwujud nyata maupun benda yang tidak nyata (konsep). Jika kita menggunakan OOP maka akan ada enam keuntungan yang dapat diperoleh, yaitu [10]:

1. Alami (*Natural*).
2. Dapat diandalkan (*Reliable*).
3. Dapat digunakan kembali (*Reusable*).
4. Mudah untuk dalam perawatan (*Maintainable*).
5. Dapat diperluas *(Extendable*).
6. Efisiensi waktu.

Berikut ini beberapa bahasa pemrograman yang sudah menggunakan konsep OOP, adalah :

1. *C++*.
2. *Visual C++*.
3. *Visual Basic*.

4. *Java*.

2.4 UML (Unified modeling Language)

UML adalah bahasa spesifikasi standar untuk mendokumentasikan, menspesifikasikan, dan membangun sebuah sistem. *UML* adalah himpunan struktur dan teknik untuk pemodelan desain program berorientasi objek (OOP) serta aplikasinya [11]. *UML* adalah metodologi untuk mengembangkan sistem OOP dan sekelompok perangkat *tool* untuk mendukung pengembangan sistem tersebut.

UML mulai diperkenalkan oleh *Object Management Group*, sebuah organisasi yang telah mengembangkan model, teknologi, dan standar OOP sejak tahun 1980-an. Sekarang UML sudah mulai banyak digunakan oleh para praktisi OOP. UML merupakan dasar bagi perangkat (*tool*) desain berorientasi objek dari IBM. UML menyediakan 10 macam diagram yang dapat digunakan untuk memodelkan aplikasi berorientasi objek yang dibuat, yaitu :

2.4.1 *Use Case Diagram*

Use case diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah "apa" yang diperbuat sistem, dan bukan "bagaimana". Sebuah *use case* merepresentasikan sebuah interaksi antara aktor dengan sistem. *Use case* merupakan sebuah pekerjaan tertentu, misalnya *login* ke sistem, meng-*create* sebuah daftar belanja, dan sebagainya. Seorang atau sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu.

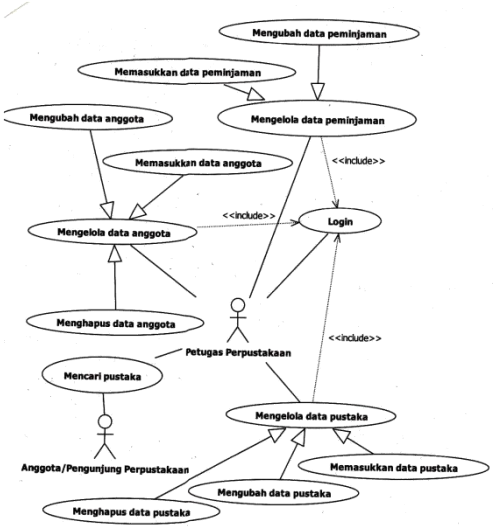
Use case diagram dapat sangat membantu menyusun *requirement* sebuah sistem, mengkomunikasikan rancangan dengan klien, dan merancang *test case* untuk semua *feature* yang ada pada sistem.

Sebuah *use case* dapat meng-*include* fungsionalitas *use case* lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa *use case* yang di-*include* akan dipanggil setiap kali *use case* yang meng-*include* dieksekusi secara normal.

Sebuah *use case* dapat di-*include* oleh lebih dari satu *use case* lain, sehingga duplikasi fungsionalitas dapat dihindari dengan cara menarik keluar fungsionalitas yang *common*.

Sebuah *use case* juga dapat meng-*extend use case* lain dengan *behaviour*-nya sendiri. Sementara hubungan generalisasi antar *use case* menunjukkan bahwa *use case* yang satu merupakan spesialisasi dari yang lain.

Contoh dari *use case diagram* dapat dilihat pada gambar 2.7.



Gambar 2. 7 Use case Diagram [11]

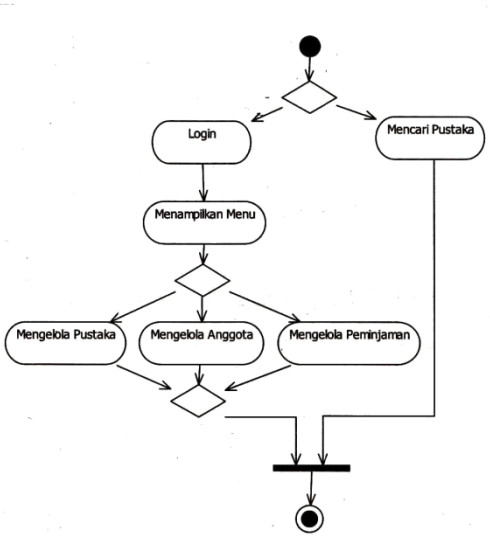
2.4.2 *Activity Diagram*

Diagram aktivitas atau *activity diagram* adalah sebuah diagram yang menggambarkan *workflow* (aliran kerja) atau aktivitas dari sebuah sistem atau proses bisnis[11]. 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 :

1. Rancangan proses bisnis di mana setiap urutan aktivitas yang digambarkan merupakan proses bisnis sistem yang didefinisikan.
2. Urutan pengelompokan tampilan dari sistem/*user interface* di mana setiap aktivitas dianggap memiliki sebuah rancangan antarmuka tampilan.

Contoh dari *activity diagram* dapat dilihat pada gambar 2.8.



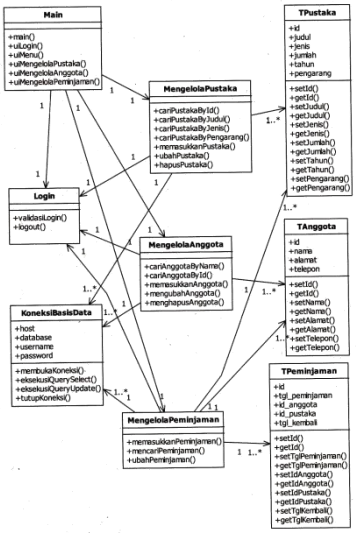
Gambar 2. 8 contoh Activity Diagram [11]

2.4.3 Class Diagram

Class adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek. *Class* menggambarkan keadaan (atribut/properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda/fungsi).

Class diagram menggambarkan struktur dan deskripsi *class, package* dan objek beserta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi, dan lain-lain.

Contoh dari *class diagram* dapat dilihat pada gambar 2.9.



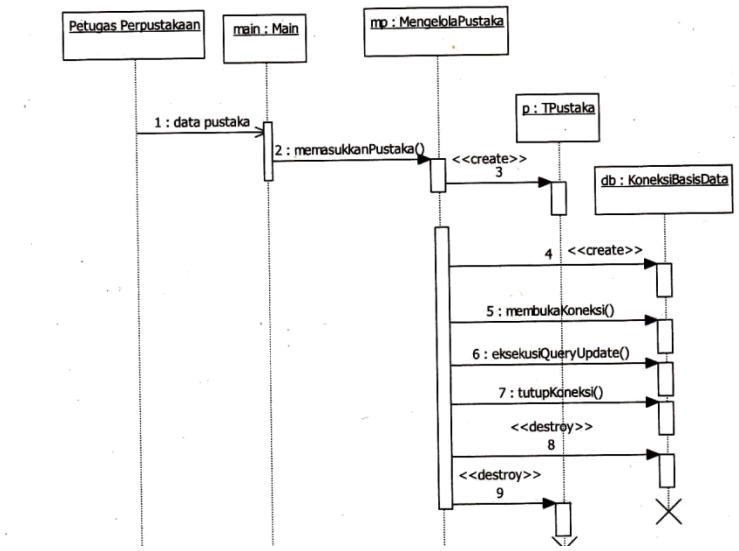
Gambar 2. 9 Contoh Class Diagram [11]

2.4.4 *Sequence Diagram*

Sequence diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, *display*, dan sebagainya) berupa *message* yang digambarkan terhadap waktu. *Sequence diagram* terdiri atar dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait).

Sequence diagram biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respons dari sebuah *event* untuk menghasilkan *output* tertentu. Diawali dari apa yang men-*trigger* aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara internal dan *output* apa yang dihasilkan.

Contoh dari *sequence diagram* dapat dilihat pada gambar 2.10.



Gambar 2. 10 Contoh Sequence Diagram [11]

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[7].

2.5.1 Pengujian *Black Box*

Menurut Roger S. Pressman [7], 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 :

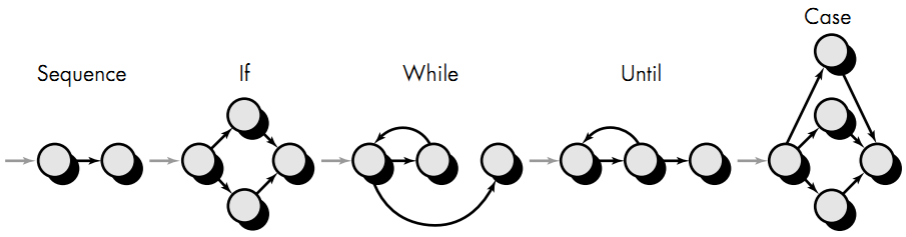
1. Fungsi-fungsi yang tidak benar atau hilang
2. Kesalahan dalam *interface*
3. Kesalahan dalam struktur data atau akses *database* eksternal
4. Kesalahan kinerja
5. Inisialisasi dan kesalahan terminasi

2.5.2 Pengujian *White Box*

Menurut Roger S. Pressman [7], 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*. Dengan menggunakan metode pengujian *white box*, perekayasa sistem dapat melakukan *test case* sebagai berikut :

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

Pengujian *white box* yang berupa notasi diagram alir dapat dilihat pada gambar 2.11.



Gambar 2. 11 Notasi Diagram Alir[7]

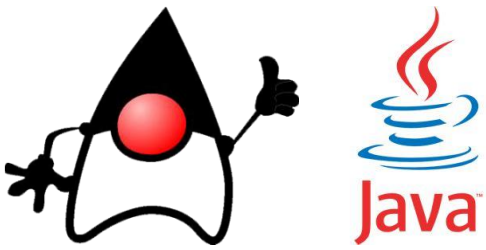
2.6 Java

Java adalah salah satu bahasa pemrograman komputer yang berorientasi objek[13], yang diciptakan oleh satu tim dari perusahaan *Sun Microsystem*, perusahaan *workstation* UNIX (*Sparc*) yang cukup terkenal. *Java* diciptakan berdasarkan bahasa C++, dengan tujuan *platform independent* (dapat dijalankan pada berbagai jenis *hardware* tanpa kompilasi ulang), dengan slogan "*Write Once, Run Anywhere* (WORA), dan *Java* pada hakikatnya lebih sederhana dan memakai objek secara murni dibanding dengan bahasa pemrograman C++.

Asal usul *java* dimulai pada tahun 1991 ketika *Sun Microsystem* melakukan penelitian terhadap berbagai produk elektronika. James Gosling (kepala proyek penelitian) beserta Patrick Naughton ditugaskan untuk merancang perangkat lunak aplikasi yang independen, tidak bergantung pada jenis perangkat keras, agar bisa dipakai pada berbagai peralatan elektronik itu. James Gosling kemudian memakai bahasa pemrograman C++ untuk menulis beberapa aplikasi untuk peralatan mikro, namun ternyata dia mengalami banyak kesulitan. Setiap kali sebuah peralatan mikro menggunakan jenis *microchip* yang berbeda, program yang digunakannya harus dikompilasi ulang. Berdasarkan hal itu kemudian munculah ide dari James Gosling untuk menciptakan sebuah bahasa baru, yang ditulis berdasarkan C++ namun dengan beberapa perbaikan. Bahasa ini kemudian disebut *Oak* (nama pohon yang terlihat dari jendela ruangan kerja James Gosling), yang digunakan untuk membuat perangkat lunak cerdas bagi peralatan elektronika buatan *Sun Mirosystem.*

Pada tahun 1994, ketika Web mulai popiler nama bahasa *Oak* kemudian diganti menjadi *Java* (salah satu sebabnya ternyata nama *Oak* telah terlebih dahulu dipakai untuk hal lain). Menurut gosip, Gosling bingung mencari nama untuk bahasa baru tersebut karena setiap nama yang dipikirkannya untuk nama bahasa pemrogramannya ternyata sudah ada yang memakai, sampai suatu ketika dia mampir di sebuah kafe untuk minum kopi, dan ia meminta "*Java"*. Timbullah idenya untuk memakai nama *Java*. Menurut Gosling nama ini cukup *cool*.

Java kemudian dipakai untuk membuat sebuah browser yang bernama *WebRunner*. Setelah berhasil maka browser ini kemudian dinamakan *HotJava* dengan simbol secangkir kopi panas, seperti pada gambar 2.12.



Gambar 2. 12 Maskot *Java* (Duke) dan *HotJava*[12]

Menurut Definisi yang diberikan oleh *Sun Microsystem*, *Java* adalah bahasa pemrograman *simple, object oriented, distributed, robust, secure, architecture neutral, portable, high-performance, multithreaded,* dan *dynamic.*

A. *Java* bahasa yang *simple* : Rancangan bahasa *Java* dibuat sedemikian rupa sehingga dengan cepat dapat dikenali dan dipahami oleh pemrogram pemula sekalipun karena mirip dengan bahasa C / C++ dan kedua lebih sederhana dari C/C++ dalam beberapa hal mengingat berbagai hal yang sulit pada C/C++ sudah tidak ada pada *Java*, misalnya pemakaian *pointer*.
B. *Java* adalah bahsa yang *object-oriented* : Konsep objek diterapkan pada pembuatan program *Java*, bahkan *Java* menggunakan konsep objek secara murnni sehingga tidak mungkin seseorang membuat program *Java* tanpa mendefinisikan kelas dan menggunakan objek.
C. *Java* adalah bahasa yang *distributed* : Bahasa *Java* dirancang sedemikian rupa sehingga mendukung pembuatan aplikasi dalam jaringan komputer. *Java* menyediakan beberapa kelas objek untuk mendukung aplikasi jaringan yang dihimpun dalam paket *java.net*.
D. *Java* adalah bahasa yang *robust* : Rancangan *Java* dibuat sedemikian rupa hingga mengurangi kemungkinan menjadi beku (*freeze*) ketika dijalankan. Kemungkinan *error* pun diminimalkan, misalnya dengan menerapkan *strong-type* sehingga program bisa terhindar dari kesalahan karena tipe data yang

keliru, *Java* juga menerapkan model memori yang dapat mencegah memori *corrupt* dan *overwrited*, misalnya dengan membuang pemakaian *pointer*.

E. *Java* adalah bahasa yang *secure* : Secara otomatis *Java* menerapkan pengamanan terhadap aplikasi sehingga mengurangi kemungkinan terjadinya serangan dari pengguna jaringan.

F. *Java* adalah bahasa yang *architecture neutral* : Program *Java* yang telah dikompilasi dapat dijalankan pada berbagai mesin dan tidak hanya bergantung pada satu arsitektur komputer saja.

G. *Java* adalah bahasa yang *portable* : Program *Java* sangat mudah dipindahkan dari satu mesin ke mesin lain tanpa harus dikompilasi ulang.

H. *Java* adalah bahasa yang *high-performance* : Pada hakikatnya program C yang telah dikompilasi berjalan jauh lebih cepat dari *Java*, karena hasil kompilasi C adalah bahasa mesin dari komputer yang dipakai untuk mengompilasi, namun hasil kompilasi ini belum tentu bisa jalan pada komputer yang berbeda arsitektur. Program *Java* dikompilasi menjadi *bytecode* yang bisa berjalan di berbagai mesin, namun ketika dijalankan *bytecode* ini harus diterjemahkan oleh JVM ke dalam bahasa mesin yang sesuai, sehingga lebih lambat dari C, namun karena *Java* dapat digunakan pada jaringan komputer yang kenyataannya tidak terlalu cepat karena hambatan saluran komunikasi maka program *Java* masih terasa cukup cepat.

I. *Java* adalah bahasa yang *multithreaded* : *Java* dirancang untuk menangani berbagai aplikasi yang berjalan secara bersamaan, misalnya memainkan lagu, sambil melakukan *download*, dan pengguna sedang melakukan *scrolling* pada *window* yang diamatinya.

J. *Java* adalah bahasa yang *dynamic* : Bahasa *Java* dirancang untuk beradaptasi dengan lingkungan yang dapat berubah dengan cepat. *Java* dapat memanggil berbagai kelas objek pada saat dibutuhkan, walaupun harus melalui jaringan. Di samping itu *Java* memiliki tepresentasi *run-time* sehingga objek yang sedang digunakan oleh suatu aplikasi dapat diperiksa berasal dari kelas yang mana melalui informasi *run-time*.

2.7 Greenfoot

Greenfoot merupakan sebuah IDE berbasis java yang dikhususkan untuk pembuatan game sederhana. Greenfoot diprakarsai oleh Michael Kolliung pada tahun 2003. Prototype yang pertama dibangun oleh Poul Henriksson dan Michael Kolling pada tahun 2003/2004. semenjak tahun 2005 pembuatan greenfoot ini dilanjutkan dengan melibatkan anggota kelompok dari University of Kent dan Deakin University.[13]

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

Didalam greenfoot terdapat project yang dinamakan scenario. Di dalam project tersebut, kita bisa membuat dua macam tipe class dari library greenfoot. Yaitu class world dan actor. World merupakan latar dari game yang akan dibuat. Untuk membuat class world, caranya adalah dengan mengklik new sub class di bagian world. Maka akan mucul sebuah kotak dialog. Kemudian kita diminta untuk memberi nama untuk kelas world tersebut. Setelah itu kita dapat memilih gambar sebagai latar belakang class world yang akan kita buat. Di dalam greenfoot sendiri telah tersedia beberapa gambar untuk latar belakang, seperti gambar tanah, pasir, batu, air, dll.

Class actor adalah sebuah kelas yang berfungsi sebagai karakter pada game yang akan kita buat. Untuk membuat actor caranya adalah dengan meng-klik icon aktor, kemudian pilih new subclass. caranya hampir mirip dengan pembuatan class world. Pada class actor juga telah disediakan beberapa gambar, seperti gambar *animal, people,* simbol, dan lain-lain.