BAB 2

TINJAUAN PUSTAKA

2.1. *Game*

Game merupakan bentuk pastisipatif, interaktif dan hiburan. Menonton televisi, membaca, dan pergi ke teater merupakan segala bentuk hiburan pasif. Sedangkan ketika seseorang bermain *game*, mereka terhibur dengan berpartisipasi secara aktif. *Game* ditempatkan pada sebuah dunia buatan yang diatur melalui aturan-aturan (*rules*). Aturan tersebut menentukan tindakan atau langkah yang pemain dapat dan tidak dapat lakukan dalam sebuah *game* [7].

Game dapat diklasifikasikan menjadi beberapa jenis, yaitu berdasarkan *platform* yang digunakan, dimensi, dan *genre* dari *game* itu sendiri. *Platform* merupakan kombinasi spesifik dari komponen elektronik atau perangkat keras komputer dengan perangkat lunak yang memungkinkan *game* untuk beroperasi [7]. Berdasarkan *platform* yang digunakan, *game* dapat dibagi menjadi beberapa jenis, diantaranya:

a. *Arcade games,* yaitu *game* yang biasanya memiliki box atau mesin yang memang khusus di desain untuk jenis *video game*s tertentu, bahkan memiliki fitur yang dapat membuat pemainnya lebih merasa masuk dan menikmati , seperti pistol, kursi khusus, sensor gerakan, sensor injakkan dan stir mobil. *Arcade games* biasanya berada di daerah/tempat khusus, sebagai contoh di Indonesia dikenal dengan sebutan ding-dong
b. *PC games,* yaitu *game* yang dimainkan menggunakan komputer pribadi
c. *Console games,* yaitu *game* yang dimainkan menggunakan konsol tertentu, seperti Playstation 2, Playstation 3, XBOX 360, dan Nintendo Wii
d. *Handheld games,* yaitu *game* yang dimainkan di konsol khusus *video game* yang dapat dibawa kemana-mana (*portable*), sebagai contoh Nintendo DS dan Sony PSP
e. *Mobile games,* yaitu *game* yang dapat dimainkan khusus untuk *mobile phone* atau PDA.

Dimensi adalah angka yang berhubungan dengan sifat metrik atau topologi dari suatu objek. Berdasarkan dimensi dari objek-objeknya, *game* dapat dibagi menjadi beberapa jenis, diantaranya:

a. *Game* 2D

   Game dua dimensi atau 2D merupakan suatu konsep dimana semua objek berada pada satu bidang datar. Gerakan pada *game* 2D dibatasi hanya horizontal dan vertikal atau secara koordinat gerakan pemain dibatasi hanya dapat bergerak pada sumbu X danY. Pada g*ame* 2D terdapat dua pergerakan kamera. Pertama adalah kamera statis dimana gambar latar (*backround*) dan tempat *game* 2D tidak bergerak sama sekali, contoh dalam jenis ini adalah tetris. Kedua adalah *Side Scrolling* dimana *game* yang kita mainkan mempunyai kamera yang dapat bergeser ke kanan atau ke kiri dengan kecepatan sesuai dengan gerakan dan kecepatan karakter yang kita mainkan/gerakkan pada *game* tersebut, contoh *game* yang termasuk pada jenis ini adalah *Super Mario Bross*, *Sonic* dan *Megaman.*

b. *Game* 2.5D

   Setelah *game* 2D, muncul *game* dengan tampilan 3D datar (3D *Plane*). *Game* seperti ini bukan 2D tapi tidak juga full 3D. Teori grafik seperti ini disebut dengan 2.5D atau *pseudo*-3D, sedangkan pada istilah *game* lebih dikenal dengan istilah *isometric*, *diametric* atau *trimetric projection.* Biasanya *gameplay* dari *game* 2.5D mirip 2D dimana kita hanya bisa bergerak secara horizontal dan vertikal, namun beberapa objek menggunakan teknik *rendering* secara 3D. Salah satu hal yang membuat *developer game* membuat *game* seperti ini karena pemrosesan 3D secara total memerlukan banyak waktu dan biaya untuk membuatnya

c. *Game* 3D

   Game 3D menggunakan tiga dimensional representasi geometris data (X, Y, Z) yang disimpan dalam komputer untuk keperluan perhitungan dan *rendering* gambar 2D. Dalam sebuah *game* 3D, pemain bisa melihat sebuah objek dari sudut 360°. Terdapat tiga dasar dalam pembuatan

sebuah objek 3D, yaitu 3D *modeling*, 3D *rendering*, dan 3D *computer graphics software* [5].

Genre pada suatu *game* memperlihatkan pola umum tantangan dari *game* tersebut. Dengan perkembangan informasi seperti sekarang, *genre* dari *game* masih terus berkembang, sebagai contoh simulasi menari (*dance simulation*) yang diperkenalkan oleh desainer *game* dari Jepang. Namun secara umum *game* dapat dibagi menjadi beberapa jenis berdasarkan *genre* yang diterapkannya, yaitu:

a. *Game* Strategi (*Strategy Games*)

   Asal-usul dari *game* strategi berasal dari *game* papan seperti catur dan *Othello*. Pada g*ame* strategi biasanya pemain dapat mengendalikan tidak hanya satu karakter, melainkan beberapa karakter dalam *game* tersebut dengan berbagai jenis tipe kemampuan, kendaraan, hingga pembuatan berbagai bangunan, pabrik dan pusat pelatihan tempur, tergantung dari tema ceritanya. G*ame* strategi dibagi menjadi 2 bentuk utama, yaitu *classical turn-based strategy* dan *real-time strategy*

b. *Role-Playing Games* (RPG)

   Game RPG sama seperti *game* strategi, yaitu salah satu *genre* yang dibuat dari *game* yang berasal dari kertas dan pena. Dua hal yang hampir sama pada semua *game* RPG, yaitu konfigurasi dari karakter pemain yang meningkat berdasarkan *experience* dan jalan cerita (*storyline*) yang kuat. Oleh karena itu, terdapat dua elemen utama yang menjadi kunci suksesnya sebuah *game* RPG. Fitur yang pertama adalah cerita (*story*) dan fitur yang kedua adalah pembangunan karakter (*character development*)

c. *Game* Olahraga (*Sports Games*)

   Game olahraga mempunyai tantangan yang tidak biasa bagi seorang desainer *game*. Tidak seperti *game-game* yang lain, dimana pemain memiliki sedikit pengetahuan mengenai dunianya, tetapi *game* olahraga meniru aturan pada olahraga di kehidupan nyata

d. Simulasi Kendaraan (*Vehicles Simulation*)

   Pada simulasi kendaraan, pemain dihadapkan dalam suatu kondisi seolah-olah pemain mengemudikan atau menerbangkan sebuah kendaraan, secara

nyata atau imajinasi. Pada simulasi kendaraan yang nyata, salah satu tujuan yang harus dicapai adalah kemiripan kendaraan, seperti karakteristik kinerja mesin (kecepatan dan manuver) dengan kendaraan yang sebenarnya. Namun, jika mendesain kendaraan imajinasi, kita bebas untuk membuat *driving experience* untuk pemain, tanpa harus terbatas oleh gravitasi, *G-forces*, kapasitas bensin, dan lainnya

e. *Game* Petualangan (*Adventure Games*)

   Game petualangan bukan merupakan sebuah kompetisi atau simulasi seperti *game* yang lainnya. G*ame* jenis ini tidak menawarkan proses untuk dikelola atau mengalahkan musuh melalui strategi dan taktik. *Game* petualangan merupakan cerita interaktif mengenai karakter yang dikontrol oleh pemain

f. *Game* Puzzle (*Puzzle Games*)

   Game puzzle merupakan *game* yang bertujuan untuk memecahkan sebuah puzzle, terkadang tanpa menyatukan dengan jalan cerita atau tujuan yang lebih besar. *Game* jenis ini biasanya bervariasi pada satu tema saja. Untuk kesuksesan secara komersil, sebuah *game* puzzle haruslah memiliki tantangan, visual yang atraktif, dan disamping itu, nyaman untuk dimainkan

g. *Game* Aksi (*Action Games*)

   Games aksi merupakan sumber yang baik bagi elemen desain dari sebuah *game*. *Game* ini relatif sederhana karena membuat analisis dari *game* tersebut lebih mudah, jika dibandingkan dengan *genre* lain. Aturan dari sebuah *game action* menjelaskan dasar dari mekanika *game.* Mereka biasanya lebih simpel dalam aturan, karena kealamian dari *gameplay*nya. Intinya, *game action* merupakan *game* yang membutuhkan keterampilan seperti pengolahan informasi sensorik dan tindakan secara cepat. Hal ini memaksa pemain untuk membuat keputusan dan melakukan tanggapan pada kecepatan yang jauh lebih besar [7].

2.2 *Artificial Intelegence*

Artificial Intelligence (AI) atau kecerdasan buatan merupakan cabang dari ilmu komputer yang berhubungan dengan tingkah laku cerdas secara otomatis. Definisi ini menunjukkan bahwa AI adalah bagian dari komputer, sehingga harus didasarkan pada teori suara (*sound theoretical*) dan prinsip-prinsip aplikasi dari bidangnya. Prinsip-prinsip ini meliputi struktur data yang digunakan dalam representasi pengetahuan, algoritma yang diperlukan untuk mengaplikasikan pengetahuan tersebut, serta bahasa dan teknik pemrograman yang digunakan dalam mengimplementasikannya [1]. Dari beberapa perspektif, AI dapat dikategorikan sebagai berikut :

1. Dari perspektif kecerdasan, AI berfungsi untuk membuat mesin yang cerdas dan dapat melakukan hal-hal yang sebelumnya hanya dapat dilakukan manusia
2. Dari perspektif bisnis, AI adalah sekelompok alat bantu (*tools*) yang berdayaguna dan metodologi yang menggunakan alat-alat bantu tersebut untuk menyelesaikan masalah masalah bisnis
3. Dari perspektif pemrograman, AI meliputi studi tentang pemrograman simbolik, pemecahan masalah, dan proses pencarian (*search*).

2.2.1 Teknik-Teknik Dasar Pencarian

Pencarian atau pelacakan merupakan salah satu teknik untuk menyelesaikan permasalahan AI. Keberhasilan suatu sistem salah satunya ditentukan oleh kesuksesan dalam pencarian dan pencocokan. Teknik dasar pencarian memberikan suatu kunci bagi penyelesaian yang penting dalam bidang AI. Ada beberapa aplikasi yang menggunakan teknik pencarian ini, yaitu :

1. *Game* papan dan puzzle, contoh: tic-tac-toe, catur, dan menara hanoi
2. Penjadwalan dan masalah *routing* pada *travelling salesman problem*
3. Parsing bahasa dan inteprestasinya pada masalah pencarian struktur dan arti
4. Logika pemrograman dalam hal pencarian fakta dan implikasinya
5. *Computer vision* dan pengenalan pola

6. Sistem pakar bebasis kaidah (*rule based expert system*)

Pencarian adalah proses mencari solusi dari suatu permasalahan melalui sekumpulan kemungkinan ruang keadaan (*state space*). Ruang keadaan merupakan suatu ruang yang berisi semua keadaan yang mungkin. Kondisi suatu pencarian meliputi :

1. Keadaan sekarang atau awal
2. Keadaan tujuan atau solusi yang dijangkau dan perlu diperiksa apakah telah mencapai sasaran
3. Biaya atau nilai yang diperoleh dari solusi.

Solusi merupakan suatu lintasan dari keadaan awal sampai keadaan tujuan. Secara umum, proses pencarian dapat dilakukan seperti berikut :

1. Memeriksa keadaan sekarang atau awal
2. Mengeksekusi aksi yang dibolehkan untuk memindahkan ke keadaan berikutnya
3. Memeriksa jika keadaan baru merupakan solusinya. Jika tidak, keadaan baru tersebut menjadi keadaan sekarang dan proses ini diulangi sampai solusi ditemukan atau ruang keadaan habis terpakai.

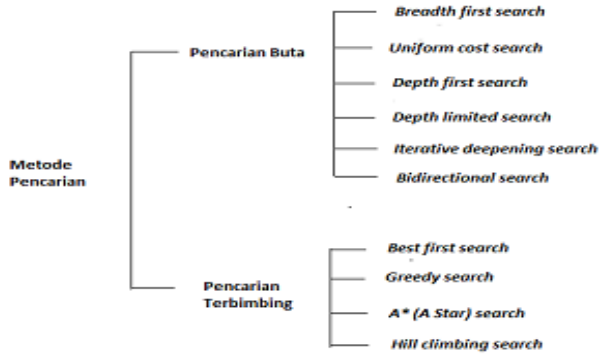
2.2.2 Algoritma Pencarian (*Search Algorithm*)

Permasalahan pencarian dapat diselesaikan dengan beberapa metode, yaitu:

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

Metode pencarian adalah bagian dari kesimpulan, dimana setiap *state* menggambarkan hipotesis dalam sebuah rangkaian deduktif. Metode pencarian dikatakan penting untuk menyelesaikan permasalahan, karena setiap keadaan (*state*) menggambarkan langkah-langkah untuk menyelesaikan permasalahan.

Selain itu, metode pencarian dikatakan penting untuk perencanaan, karena dalam sebuah *game* akan menentukan apa yang harus dilakukan, dimana setiap *state* menggambarkan kemungkinan posisi pada suatu saat.



Gambar 2.1 Bagan Metode Pencarian (*Searching*)

Berdasarkan Gambar 2.1, metode penelusuran dibagi menjadi dua golongan, yakni pencarian buta (*blind search*) dan pencarian terbimbing (*heuristic search*).

2.2.2.1 Pencarian Buta (*Blind Search*)

Blind Search adalah pencarian solusi tanpa adanya informasi yang dapat mengarahkan pencarian untuk mencapai keadaan akhir (*goal state*) dari keadaan sekarang (*current state*). Informasi yang ada hanyalah definisi *goal state* itu sendiri, sehingga algoritma dapat mengenali *goal state* apabila menemukan informasi yang dicari.

Dengan tidak adanya informasi, maka *blind search* akan memeriksa/ mengembangkan *node-node* secara tidak terarah dan kurang efisien untuk kebanyakan kasus karena banyaknya *node* yang dikembangkan.

Beberapa contoh algoritma yang termasuk *blind search*, antara lain *Breadth First Search, Uniform Cost Search, Depth First Search, Depth Limited Search, Iterative Deepening Search*, dan *Bidirectional Search*.

2.2.2.2 Pencarian Terbimbing (*Heuristic Search*)

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

Heuristic search menghitung (perkiraan) *cost* ke *goal state* dengan menggunakan fungsi *heuristic*. Fungsi *heuristic* berbeda dengan algoritma, dimana fungsi *heuristic* merupakan perkiraan untuk membantu algoritma, dan tidak harus *valid* setiap waktu. Namun, semakin bagus fungsi *heuristic* yang dipakai, semakin cepat dan akurat solusi yang didapat. Menentukan fungsi *heuristic* yang tepat untuk kasus dan implementasi yang ada juga sangat berpengaruh terhadap kinerja algoritma pencarian.

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

2.2.3 Algoritma A*

Algoritma A* (A *star*) merupakan perbaikan dari metode *Best First Search* (BFS) dengan memodifikasi fungsi *heuristic* dari BFS. Pada kondisi yang tepat, A* akan memberikan solusi yang terbaik dalam waktu yang optimal dengan meminimumkan total biaya lintasan.

Pencarian menggunakan algoritma A* pada dasarnya memmiliki prinsip yang sama dengan algoritma BFS, hanya saja dengan dua faktor tambahan, yaitu:

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

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

Cost untuk setiap simpul tidak harus berupa jarak. *Cost* dapat berupa waktu apabila kita ingin mencari jalan dengan waktu tercepat untuk dilalui. Sebagai contoh, bila kita berkendaraan melewati jalan biasa bisa saja merupakan jarak terdekat, tetapi melewati jalan tol biasanya memakan waktu lebih sedikit. Algoritma A* bekerja dengan prinsip yang hampir sama dengan BFS, kecuali dengan dua perbedaan, yaitu :

1. Simpul-simpul pada *OPEN list* diurutkan oleh *cost* keseluruhan dari simpul awal ke simpul tujuan, dimulai dari *cost* terkecil sampai *cost* terbesar. Pengurutan tersebut dikenal dengan istilah antrian prioritas (*priority queue*). *Cost* keseluruhan dihitung berdasarkan *cost* dari simpul awal ke simpul sekarang (*current node*) ditambah *cost* perkiraan menuju simpul tujuan
2. Simpul pada *CLOSED list* bisa dimasukkan ke dalam *OPEN list*, bila jalan terpendek (*cost* lebih kecil) menuju simpul tersebut ditemukan.

Algoritma A* menggunakan dua senarai (*list*) yaitu *OPEN* dan *CLOSED*. *OPEN* adalah *list* yang digunakan untuk menyimpan simpul-simpul yang pernah dibangkitkan dan nilai *heuristic* dari simpul tersebut telah dihitung tetapi belum terpilih sebagai simpul terbaik (*best node*), dengan kata lain *OPEN* berisi simpul-simpul yang masih memiliki peluang untuk terpilih sebagai simpul terbaik. Sedangkan *CLOSED* adalah *list* untuk menyimpan simpul-simpul yang pernah dibangkitkan dan terpilih sebagai simpul terbaik. Artinya, *CLOSED* berisi simpul-simpul yang tidak mungkin terpilih kembali sebagai simpul terbaik (peluang untuk terpilih sudah tertutup).

OPEN list diurutkan berdasarkan perkiraan *cost* keseluruhan, kemudian algoritma memeriksa simpul-simpul yang mempunyai perkiraan *cost* yang paling kecil terlebih dahulu dan mencari simpul-simpul yang kemungkinan mengarah ke simpul tujuan. Perkiraan *cost* ditentukan secara manual, lebih baik perkiraan yang dibuat lebih cepat pencariannya.

Cost antar simpul adalah jarak dan perkiraan *cost* dari suatu simpul ke simpul tujuan. Agar lebih mudahnya ditunjukkan pada persamaan 2.1.

$$f(n) = g(n) + g(n) \text{.....} (2.1)$$

Keterangan :

f(n) = fungsi evaluasi(jumlah *g*(*n*)dengan *h*(*n*))

g(n) = *cost* yang dikeluarkan dari keadaan awal sampai keadaan n

h(n) = estimasi biaya untuk sampai pada suatu tujuan mulai dari n [1]

Nilai F adalah *cost* perkiraan suatu *path* yang teridentifikasi. Nilai F merupakan hasil dari *f*(n). Nilai G adalah hasil dari fungsi *g*(n) yang menyatakan banyaknya langkah yang diperlukan untuk menuju ke *path* sekarang. Setiap simpul (*node*) harus memiliki nilai *h*(n), yaitu estimasi harga simpul dihitung dari simpul tujuan yang hasilnya menjadi nilai H (*heuristic*)[1].

A* merupakan algoritma pencarian yang menggunakan fungsi *heuristic* untuk pencarian rute. Dalam aplikasi ini, fungsi *heuristic* yang dipakai mengisi nilai H pada algoritma A*. Terdapat beberapa fungsi *heuristic* umum yang bisa dipakai untuk algoritma A*. Salah satunya dikenal dengan istilah *Manhattan Distance*. Fungsi ini digunakan untuk kasus peta yang hanya memiliki pergerakan horizontal dan vertikal.

Perhitungan nilai *heuristic* untuk *node* ke-n menggunakan *Manhattan Distance* adalah sebagai berikut :

$$h(n) = (\text{abs}(n.x - goal.x) + \text{abs}(n.y - goal.y)) \text{.....} (2.2)$$

Keterangan :

h(n) = nilai *heuristic* untuk *node* n

goal = *node* tujuan

Jika pergerakan diagonal pada peta diperbolehkan, maka digunakan fungsi *heuristic* selain *Manhattan Distance*. Untuk mendekati kenyataan, *cost* untuk perpindahan *node* secara diagonal dan orthogonal dibedakan. *Cost* diagonal

adalah 1,4 kali *cost* perpindahan secara orthogonal. Maka fungsi *heuristic* yang digunakan adalah sebagai berikut:

$$h_\text{diagonal}(n) = \min(\text{abs}(n.x - goal.x) + \text{abs}(n.y - goal.y)) \quad \ldots\ldots\ldots (2.3)$$

$$h_\text{orthogonal}(n) = (\text{abs}(n.x - goal.x) + \text{abs}(n.y - goal.y)) \quad \ldots\ldots\ldots (2.4)$$

$$h(n) = h_\text{diagonal}(n) + (h_\text{orthogonal}(n) - (2 * h_\text{diagonal}(n))) \quad \ldots\ldots\ldots (2.5)$$

Keterangan:

*h*_diagonal(n) = banyaknya langkah diagonal yang bisa diambil untuk mencapai *goal* dari *node* n

*h*_orthogonal = banyaknya langkah lurus yang bisa diambil untuk mencapai *goal* dari *node* n

Nilai *heuristic* kemudian diperoleh dari *h*_diagonal(n) ditambah dengan selisih *h*_orthogonal(n) dengan dua kali *h*_diagonal(n). Dengan kata lain, jumlah langkah diagonal kali *cost* diagonal ditambah jumlah langkah lurus yang masih bisa diambil dikali *cost* pergerakan lurus.

2.3 *Procedural Conten Generation* (PCG)

Procedural Content Generation (PCG) adalah metode yang digunakan untuk penempatan objek-objek yang diperlukan dalam pembentukan dunia *game* (*platform,* batu-batuan, pohon dan sebagainya). Konten *game* merupakan faktor penting dalam menjaga pemain agar tidak cepat merasa bosan dan tetap terlibat dalam dunia *game*. Namun permintaan para pelaku *game* untuk pembuatan konten baru semakin meningkat, sementara produksi konten manual oleh desainer *game* semakin mahal [10]. Berbeda dengan produksi konten secara manual, PCG dapat mengkombinasikan semua konten sederhana yang diperlukan dalam sebuah dunia *game* dengan menempatkan konten-kontennya secara otomatis. Jadi apabila ingin menciptakan kebun yang indah, kita hanya membutuhkan satu konten pohon dan satu konten lampu taman.

Saat ini, produksi *game* komersial berkualitas tinggi memerlukan banyak pekerja, termasuk seniman, desainer, *programmer*, dan insinyur *audio*, dan

banyak dari mereka menghasilkan konten *game* yang baru. Akibatnya, produksi konten telah tumbuh ke titik dimana ia telah menjadi hambatan kedua anggaran setelah pemasaran [10]. Teknik PCG merupakan alternatif untuk membuat dunia *game* dalam waktu terbatas tanpa menempatkan beban yang besar pada desainer konten dari *game*. Gagasan utama dari PCG ini adalah konten *game* tidak ditempatkan secara manual oleh desiner, tetapi oleh komputer dengan menjalankan prosedur yang jelas. Untuk menghindari kehilangan kontrol atas proses desain yang diinginkan, desainer masih dapat mempengaruhi hasil akhir dengan menyesuaikan parameter yang diperlukan dari prosedur.

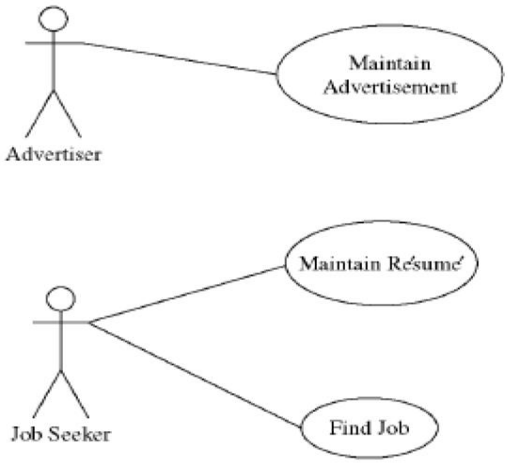
PCG memunculkan konten secara otomatis. Oleh karena itu, tantangannya adalah bagaimana menyeimbangkan *gameplay* yang merupakan aturan dari *action game*. Untuk itu harus disesuaikan waktu ketika musuh dimunculkan dengan jarak tempuh pemain unutk mencapai koordinatnya agar tidak terjadi penumpukan konten musuh pada satu tempat.

2.4 UML (*Unified Modelling Language*)

Unified Modeling Language (UML) adalah keluarga notasi grafis yang didukung oleh meta-*mode*l tunggal, yang membantu pendeskripsian dan desain sistem perangkat lunak, khususnya sistem yang dibangun menggunakan *Object Oriented Programming* (OOP) [4].

2.4.1 *Use Case Diagram*

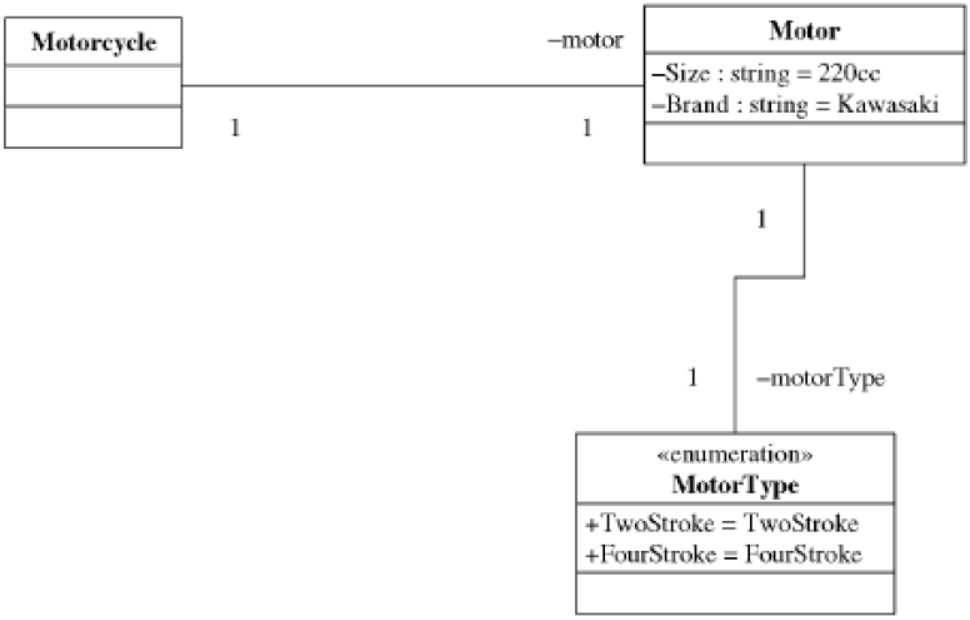
Use case diagram menggambarkan tujuan penting fungsionalitas sistem yang akan ditampilkan pada pengguna. *Use case* merepresentasikan sebuah interaksi antara *actor* dengan sistem. *Use case diagram* dapat membantu dalam penyusunan *requirement* sebuah sistem, mengkomunikasikan rancangan dengan klien, dan merancang *test case* untuk semua fitur yang ada pada sistem [4].



Gambar 2.2 Contoh *Use Case* Diagram [4]

2.4.2 *Class Diagram*

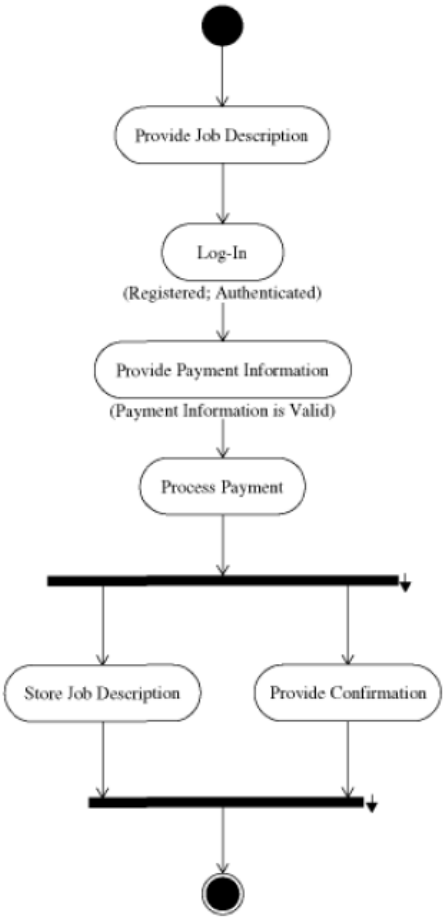
Class diagram digunakan untuk memperlihatkan *class* pada sistem dan hubungan antara *class*. *Class diagram* tidak menjelaskan fungsionalitas atau bagaimana *instances class* berinteraksi. *Class* menggambarkan keadaan (*atribut*/ *property*) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (*method*/fungsi) [4].



Gambar 2.3 Contoh *Class Diagram* [4]

2.4.3 *Activity Diagram*

Activity diagram digunakan untuk menggambarkan aliran aktivitas baik dalam proses bisnis maupun *use case*. *Diagram* ini dapat digunakan untuk memodelkan aksi yang akan dilakukan saat sebuah operasi dieksekusi dan memodelkan hasil dari aksi tersebut [4].

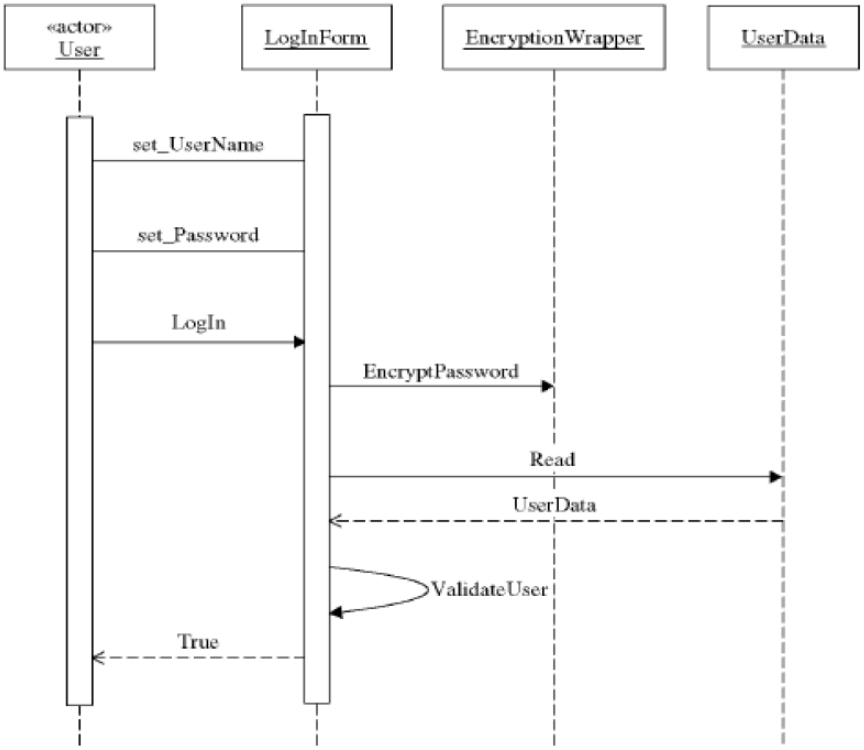


Gambar 2.4 Contoh *Activity Diagram* [4]

2.4.4 *Sequence Diagram*

Sequence diagram biasanya digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respon dari sebuah *event* untuk menghasilkan *output* tertentu. Diawali dari apa yang menjadi *trigger* dari aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara internal dan *output* yang dihasilkan. Masing-masing objek, termasuk *actor*, memiliki *lifeline*

vertical. *Message* digambarkan sebagai garis berpanah dari satu objek ke objek lainnya. Pada fase desain berikutnya, *message* akan dipetakan menjadi operasi/*method* dari *class*. *Activation bar* menunjukan lamanya eksekusi sebuah proses, biasanya diawali dengan diterimanya sebuah *message* [4].



Gambar 2.5 Contoh *Sequence Diagram* [4]