BAB 2
TINJAUAN PUSTAKA

2.1 Pengolahan Citra Digital

Pengolahan citra digital artinya melakukan pemrosesan gambar berdimensi dua melalui komputer digital. Menurut Efford (2000), pengolahan citra adalah istilah umum untuk berbagai teknik yang keberadaanya untuk memanipulasi dan memodifikasi citra dengan berbagai cara [8]. Foto adalah contoh gambar berdimensi dua yang dapat diolah dengan mudah. Setiap foto dalam bentuk citra digital dapat diolah menggunakan perangkat lunak tertentu. Misalnya citra yang gelap dapat diolah supaya menjadi lebih terang, merubah format warna citra menjadi *grayscale*, *resize* citra dan lain sebagainya.

2.1.1 *Grayscale*

Citra *grayscale* menangani gradasi warna hitam dan putih, yang tentu saja menghasilkan efek warna abu-abu. Warna gambar dinyatakan dengan intensitas. Dalam hal ini, intensitas berkisar antara 0 sampai dengan 255. Nilai 0 menyatakan hitam dan nilai 255 menyatakan putih [9]. Berikut adalah rumus konversi citra berwana (RGB) menjadi *grayscale*.

$$I = (0.2989 * R) + (0.5870 * G) + (0.1141 * B) \quad (2.1)$$

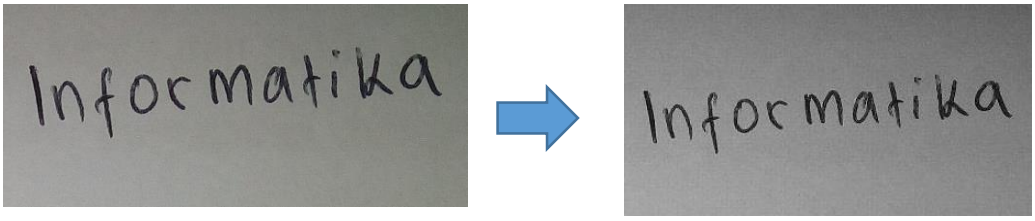
Keterangan :

R = komponen nilai merah (*Red*) dari suatu titik pixel

G = komponen nilai hijau (*Green*) dari suatu titik pixel

B = komponen nilai biru (*Blue*) dari suatu titik pixel

Persamaan di atas merupakan salah satu rumus yang digunakan untuk mengkonversi citra berwarna menjadi grayscale. Persamaan 2.1 dipilih dalam penelitian ini karena mata manusia secara alami lebih sensitif terhadap cahaya merah dan hijau. Maka dari itu, warna-warna ini diberi bobot yang lebih tinggi untuk memastikan bahwa keseimbangan intensitas relatif dalam citra grayscale yang dihasilkan mirip dengan citra warna RGB [10].



Gambar 2. 1 Contoh Hasil *Grayscale*

2.1.2 *Sauvola Threshold*

Sauvola threshold merupakan metode *threshold* yang mampu memberikan peningkatan dengan cara menghitung ambang menggunakan rentang nilai dinamis dari nilai standar deviasi citra *grayscale* [11]. Sauvola termasuk dalam *local threshold* dimana nilai ambang ditentukan oleh nilai pixel tetangga.

$$T(x,y) = m(x,y) * (1 + k * (\frac{s(x,y)}{R} - 1)) \quad (2.2)$$

$$m(x,y) = \frac{\sum_{i=min}^{i=max}\sum_{j=min}^{j=max} img(i,j)}{i*j} \quad (2.3)$$

$$s(x,y) = \sqrt{\frac{\sum_{i=min}^{i=max}\sum_{j=min}^{j=max} (img(i,j)-m(x,y))^2}{(i*j)-1}} \quad (2.4)$$

Keterangan :

R : Nilai maksimum dari standar deviasi (128 untuk citra *grayscale*)

k : Kernel dengan nilai antara 0.2 – 0.5.

m : Fungsi yang menghasilkan nilai rata-rata dari sejumlah pixel citra.

s : Fungsi yang menghasilkan nilai standar deviasi dari sejumlah pixel citra

T : Fungsi yang menghasilkan nilai *threshold* (ambang)

x : Nilai koordinat lebar citra

y : Nilai koordinat tinggi citra

i : Nilai ukuran piksel tetangga

j : Nilia ukuran piksel tetangga

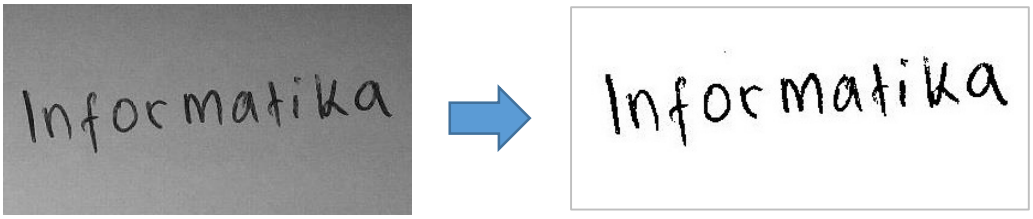
Setelah nilai ambang T(x,y) sudah didapatkan, selanjutnya masukkan ke persamaan di bawah ini.

$$f(x,y) = \begin{cases} 0, & \text{img}(x,y) < \text{T}(x,y) \\ 255, & \text{img}(x,y) \geq \text{T}(x,y) \end{cases} \quad (2.5)$$

Keterangan :

f : Fungsi yang menghasilkan nilai 0 atau 255

img : Nilai *grayscale* citra



Gambar 2. 2 Contoh hasil *Sauvola Threshold*

2.1.3 Operasi Dilasi

Operasi dilasi digunakan untuk memperbaiki citra yang pecah. Bila suatu objek (citra *input*) dinyatakan dengan A dan SE dinyatakan dengan B serta Bx menyatakan translasi B sedemikian sehingga pusat B terletak pada x. Operasi dilasi A dengan B dapat dinyatakan sebagai berikut [8].

$$D(A,B)=A \oplus B= \{x:B_x \cap A \neq \emptyset\} \quad (2.6)$$

Dengan ∅ menyatakan himpunan kosong.

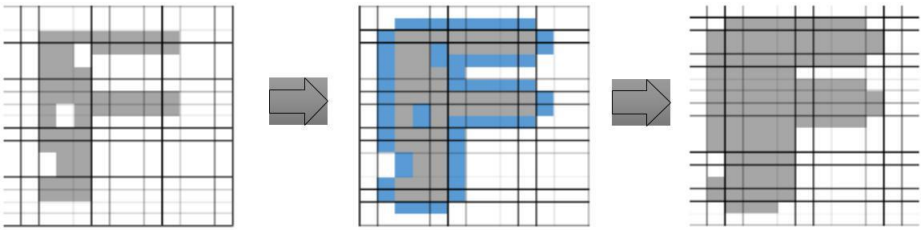
Keterangan :

D : Fungsi yang menghasilkan citra *dilasi*

A : Citra *input*

B : *Structuring Elements* (SE)

Proses *dilasi* dilakukan dengan membandingkan setiap *pixel* citra *input* dengan nilai pusat SE dengan cara (*superimpose*) SE dengan citra sehingga pusat SE tepat dengan posisi *pixel* citra yang di proses. Jika paling sedikit ada 1 pixel pada SE sama dengan nilai *pixel* objek (*foreground*) citra maka *pixel* input diset nilainya dengan nilai *pixel foreground* dan bila semua *pixel* yang berhubungan adalah *background* maka maka input *pixel* diberi nilai *background*. Proses serupa dilanjutkan dengan menggerakkan (translasi) SE *pixel* demi *pixel* pada citra input Berikut adalah contoh matriks citra *dilasi* dengan menggunakan *structure element* 3x3



Gambar 2. 3 Contoh Dilasi

2.1.4 Operasi Erosi

Proses *erosi* merupakan kebalikan dari proses *dilasi*. Jika dalam proses *dilasi* menghasilkan objek yang lebih luas maka dalam proses *erosi* akan menghasilkan objek yang menyempit (mengecil). Operasi *erosi* dapat dinyatakan sebagai berikut [8].

$$D(A,B)=A\ominus B=\{x:B_x\cap A\neq\emptyset\} \quad (2.7)$$

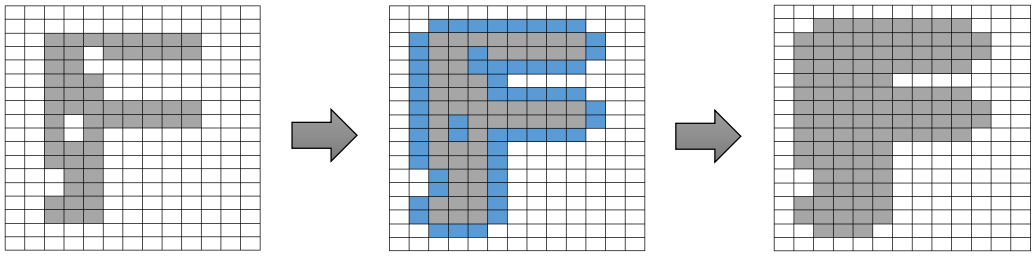
Keterangan :

A : Citra input

B : *Structuring Elements* (SE)

E : Fungsi yang menghasilkan citra *erosi*

Sama Seperti *dilasi*, proses *erosi* dilakukan dengan membandingkan setiap *pixel* citra input dengan nilai pusat SE dengan cara melapiskan SE dengan citra sehingga pusat SE tepat dengan posisi *pixel* citra yang diproses. Jika semua *pixel* pada SE tepat sama dengan semua nilai *pixel* objek (*foreground*) citra maka *pixel* input diset nilainya dengan nilai *pixel foreground*, bila tidak maka input *pixel* diberi nilai *pixel background*. Proses serupa dilanjutkan dengan menggerakkan SE *pixel* demi *pixel* pada citra *input*. Berikut adalah contoh *erosi* dengan menggunakan *structure element* 3x3.



Gambar 2. 4 Contoh Erosi

2.1.5 Segmentasi

Segmentasi merupakan teknik untuk membagi suatu citra menjadi beberapa daerah (*region*) dimana setiap daerah memiliki kemiripan atribut [8]. Sampai sejauh mana pembagian tersebut dilakukan tergantung pada masalah yang dihadapi. Idealnya, langkah segmentasi tersebut dihentikan pada saat objek yang diinginkan sudah berhasil dipisahkan. Dalam penelitian ini segmentasi dilakukan dengan memotong baris dan kolom citra. Pemotongan pertama dilakukan untuk semua baris, kemudian dilanjutkan dengan pemotongan kolom. Segmentasi dapat dilihat pada Gambar 2.4.



Gambar 2. 5 Segmentasi Citra

2.1.6 *Resize*/Normalisasi

Resize citra artinya mengubah besarnya ukuran citra digital dalam *pixel*. Adakala ukurannya berubah menjadi lebih kecil dari ukuran aslinya dan ada kalanya sebaliknya. Proses *Resize* citra dapat mempengaruhi *size* citra lebih besar atau lebih kecil dan juga kualitas citra menjadi lebih baik atau lebih buruk. *Resize*/Normalisasi dengan ukuran 16x16 [12] piksel dapat dilihat pada Gambar 2.5.



Gambar 2. 6 *Resize*/Normalisasi

2.2 Algoritma *Principal Component Analysis* (PCA)

PCA adalah suatu teknik handal untuk mengekstraksi struktur dari suatu set data dengan dimensi yang cukup banyak. Masalah dalam PCA adalah menemukan *eigenvalue* dan *eigenvector*. PCA adalah transformasi *orthogonal* (tegak lurus) dari sistem koordinat dimana kita mendeskripsikan data. Koordinat baru kita merepresentasikan data dinamakan *principal component* atau PC. Koordinat ini dipilih dimana variansi dari data mencapai maksimum. Seringkali hanya dengan beberapa *principal component* sudah cukup untuk menjelaskan struktur dari data

aslinya. Jika data dalam dimensi asli sulit untuk direpresentasikan melalui grafik, maka dengan dua PC atau satu PC bisa digambarkan melalui grafik [13]. Berikut adalah metodologi dari PCA [14]:

Setiap gambar karakter 2-D digambarkan sebagai vektor 1-D dengan menggabungkan setiap kolom (atau baris) ke dalam vektor tipis yang panjang. Jadi, vektor yang dihasilkan harus hadir sebagai:

$$\bar{x}^i = [x_1^i \dots x_N^i]^T \tag{2.8}$$

Keterangan :

$\bar{x}$: kumpulan citra karakter 1D

T : *Transpose*

x : citra karakter 1D

N : banyak karakter

Gambar rata-rata adalah vektor kolom dimana setiap entri adalah rata-rata dari semua piksel gambar pelatihan yang sesuai. Teori sebelumnya dapat dinyatakan dalam ungkapan berikut:

$$\bar{x}^i = x^i - m \tag{2.9}$$

Keterangan :

$\bar{x}^i$: citra karakter 1D

m : matriks kolom rata-rata karakter

dimana

$$m = \frac{1}{p}\sum_{i=1}^{p} x^i \tag{2.10}$$

Keterangan :

m : matriks kolom rata-rata citra karakter

p : banyak data

x : citra karakter 1D ke-i

i : banyak karakter

Data pusat ;

$$\bar{X} = [\bar{x}^1|\bar{x}^2| \dots |\bar{x}^p|] \quad (2.11)$$

Keterangan :

$\bar{X}$: kumpulan citra karakter 1D

p : banyak data

x : citra karakter 1D

Matriks kovarian

$$\Omega = \bar{X}\bar{X}^T \quad (2.12)$$

Keterangan :

Ω : matriks kovarian

$\bar{X}$: kumpulan citra karakter 1D

T : *transpose*

$$det(C - \lambda) = 0 \quad (2.13)$$

Keterangan :

C : matriks kovarian

λ : matriks *eigenvalue*

Mengurutkan urutan vector *Eigen* :

$$V = [V_1|V_2| \dots |V_p|] \quad (2.14)$$

Keterangan :

V : Matriks *Eigenvector*

p : banyak data

Proyeksi gambar :

$$\tilde{x}^i = V^T\bar{x}^i \quad (2.15)$$

Keterangan :

$\tilde{x}$: matriks proyeksi latih

V : matriks *eigenvector*

T : *transpose*

$\bar{x}$: gambar citra 1D

Mengidentifikasi gambar baru

$$\bar{y}^i = y^i - m \tag{2.16}$$

Keterangan :

$\bar{y}$: pusat data citra karakter 1D baru

y : citra karakter 1D baru

m : matriks kolom citra karakter rata-rata

dan

$$\tilde{y}^i = V^T \bar{y}^i \tag{2.17}$$

Keterangan :

$\tilde{y}$: matriks proyeksi baru

$\bar{y}$: pusat data citra karakter 1D baru

V : matriks *eigenvector*

T : *transpose*

2.3 Algoritma *K-Nearest Neighbor* (K-NN)

Teknik ini termasuk dalam kelompok klasifikasi *nonparametric*. Teknik ini sangat sederhana dan mudah diimplementasikan. Mirip dengan teknik klastering, pengelompokan suatu data baru berdasarkan jarak data baru itu ke beberapa data/tetangga (*neighbor*) terdekat. Dalam hal ini jumlah data/tetangga terdekat ditentukan oleh user yang dinyatakan dengan *k*. Misalkan tentukan nilai $k = 5$, maka setiap data testing dihitung jaraknya terhadap data training dan pilih 5 data training yang jaraknya paling dekat ke data testing. Setelah diketahui 5 data ini, periksa output atau labelnya masing-masing. Lalu tentukan output mana yang frekuensinya paling banyak [13].

Untuk menghitung jarak antara dua titik *x* dan *y* bisa digunakan jarak Euclidean sebagai berikut :

$$d(x,y) = \| x - y \|^2 = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \quad (2.18)$$

Keterangan :

d(x,y) : Jarak antara dua titik

x_i : Data latih ke-i

y_i : Data uji ke-i

i : Bilangan cacah i = (1,2,3,...n)

2.4 Algoritma *Rabin Karp*

Algoritma Rabin-karp adalah algoritma pencocokan string yang menggunakan fungsi *hash* sebagai pembanding antara *string* yang dicari (m) dengan *substring* pada teks (n). Algoritma Rabin-karp didasarkan pada fakta jika dua buah *string* sama maka harga *hash value*-nya pasti sama. Akan tetapi ada dua masalah yang timbul dari hal ini, masalah pertama yaitu ada begitu banyak *string* yang berbeda, permasalahan ini dapat dipecahkan dengan meng-*assign* beberapa *string* dengan *hash value* yang sama. Masalah yang kedua belum tentu *string* yang mempunyai *hash value* yang sama cocok untuk mengatasinya maka untuk setiap *string* yang di-*assign* dilakukan pencocokan *string* secara Brute-Force. Kunci agar algoritma Rabin-Karp efisien, terdapat pada pemilihan *hash value*-nya. Salah satu cara yang efektif adalah memperlakukan setiap *substring* sebagai suatu bilangan dengan basis tertentu [15]. Tahap-tahap yang dilakukan dalam evaluasi jawaban esai yaitu proses *tokenizing* dan *case folding*, *filtering*, algoritma Rabin-Karp (K-gram, *hashing*, *similarity*).

2.4.1 *Tokenizing* dan *Case Folding*

Pada tahan *tokenizing* dan *case folding* dilakukan proses pemisahan teks menjadi per kata, kemudian mengubah semua kata yang sudah ditoken menjadi huruf kecil (*lowercase*). Berikut adalah contoh *Tokenizing* dan *Case Folding*.

Tabel 2. 1 *Tokenizing* dan *Case Folding*

Teks	Bandung ada di Provinsi Jawa Barat

Hasil *tokenizing* dan *case folding*	bandung ada di provinsi jawa barat

2.4.2 *Filtering*

Filtering adalah tahap mengambil kata-kata penting dari hasil *tokenizing* dan *case folding*, dengan menggunakan *stoplist* / *stopword* (membuang kata yang kurang penting). Contoh kata yang termasuk ke dalam *stoplist* seperti : ‘di’, ‘yaitu’, ‘adalah’, ‘merupakan’, ‘dan’, ‘yang’, ‘ini’. Berikut adalah contoh dari proses *filtering*.

Tabel 2. 2 *Filtering*

Teks	Bandung ada di Provinsi Jawa Barat
Hasil *filtering*	Bandung provinsi jawa barat

Pada contoh di atas kata ‘ada’ dan ‘di’ dihilangkan karena kata tersebut termasuk dalam stoplist / stopword, sehingga kata yang akan diproses dalam algoritma Rabin-Karp adalah

Tabel 2. 3 *Stoplist* / *Stopword*

Teks	Bandung, provinsi, jawa, barat

2.4.3 K-gram

K-gram adalah rangkaian *terms* dengan panjang K. Kebanyakan yang digunakan sebagai *terms* adalah kata. K-gram merupakan sebuah metode yang diaplikasikan untuk pembangkitan kata atau karakter. Metode K-gram ini digunakan untuk mengambil potongan-potongan karakter huruf sejumlah k dari sebuah kata yang secara kontinuitas dibaca dari teks sumber hingga akhir dari kalimat atau dokumen [15]. Berikut adalah contoh K-gram dengan nilai k=4.

Tabel 2. 4 K-gram

Teks	Bandungprovinsijawabarat
Hasil *K-gram*	band, andu, ndun, dung, ungp, ngpr, gpro, prov, rovi, ovin, vins, insi, nsij, sija, ijaw, jawa, awab, waba, abar, bara, arat

2.4.4 *Hashing*

Hashing adalah suatu cara untuk mentransformasi sebuah *string* menjadi suatu nilai yang unik dengan panjang tertentu (*fixed-length*) yang berfungsi sebagai penanda *string* tersebut. Fungsi untuk menghasilkan nilai ini disebut fungsi *hash*, sedangkan nilai yang dihasilkan disebut nilai *hash*. Nilai *hash* pada umumnya digambarkan sebagai *fingerprint* yaitu suatu *string* pendek yang terdiri atas huruf dan angka yang terlihat acak (data biner yang ditulis dalam heksadesimal) [15]. Berikut adalah persamaan untuk mendapatkan nilai *hasing* untuk setiap K-gram.

$$H = C_1 * a^{(k-1)} + C_2 * a^{(k-2)} + C_3 * a^{(k-3)} \dots C_k * a^{()} \quad (2.19)$$

Keterangan :

H : Nilai *hashing*

C_k : Nilai ascii dari setiap huruf pada K-grams

a : nilai konstanta / basis

k : Banyak karakter

2.4.5 *Similarity*

Similarity digunakan untuk menghitung tingkat kemiripan suatu teks, dimana yang dibandingkan adalah nilai hasil proses *hashing*. Untuk mengukur persentase kemiripan menggunakan metode *Dice's Similarity Coefisient* dimana *hash value* yang sama dari dua dokumen teks abstrak yang berbeda dihitung dan jumlahnya disimpan dalam bentuk array. Berikut adalah persamaan *similarity* metode *Dice's Similarity Coefisient.*

$$S = \frac{2 * C}{A + B} \quad (2.20)$$

Keterangan :

S : Nilai *similarity* antar kedua teks

A : Jumlah kumpulan K-grams dalam teks 1

B : Jumlah kumpulan K-grams dalam teks 2

C : Jumlah dari K-grams yang sama dari teks yang dibandingkan

2.5 *Unified Modelling Language* (UML)

Unified Modeling Language (UML) adalah bahasa grafis untuk mendokumentasi, menspesifikasikan, dan membangun sistem perangkat lunak

[16]. UML berorientasi objek, menerapkan banyak level abstraksi, tidak bergantung proses pengembangan tidak bergantung bahasa dan teknologi, pemaduan beberapa notasi di beragam metodologi, usaha bersama dari banyak pihak, didukung oleh kakas-kakas yang diintegrasikan lewat XML. Standar UML dikelola oleh OMG (*Object Management Group*). UML bukanlah :

1.Bahasa pemrograman visual, tetapi bahasa pemodelan visual.

2.Spesifikasi Kakas, tetapi spesifikasi bahasa pemodela.

3.Proses, Tetapi yang memungkinkan proses-proses.

Teknik pemodelan dalam UML ada dua jenis diagramnya [8], yaitu :

1. Diagram struktur

   Diagram ini untuk memvisualisasi, menspesifikasikan, membangun dan mendokumentasikan aspek statik dari sistem. Diagram struktur UML terdiri dari : Diagram kelas (*Class diagram*), Diagram objek (*Object diagram*), Diagram Komponen (*Component diagram*), dan Diagram deployment (*Deployment diagram*).

2. Diagram prilaku

   Diagram ini untuk memvisualisasi, menspesifikasikan, membangun dan mendokumentasikan aspek dinamis dari sistem. Diagram prilaku dalam UML terdiri dari : Diagram use case (*Use case diagram*), Diagram sekuen (*Sequence diagram*), Diagram kolaborasi (*Collaboration diagram*), Diagram aktivitas (*Activity Diagram*).

2.5.1 *Use Case Diagram*

Use Case diagram merupakan salah satu diagram untuk memodelkan aspek tingkah laku sistem. Masing-masing use case diagram menunjukkan sekumpulan use case, aktor dan hubungannya. Use case diagram penting untuk memvisualisasikan, menspesifikasikan, dan mendokumentasikan kebutuhan perilaku sistem. Diagram-diagram use case merupakan pusat pemodelan prilaku sistem, subsistem dan kelas [19]. Use case adalah interaksi antara aktor eksternal dan sistem, hasil yang dapat diamati oleh aktor, berorientasi pada tujuan, dideskripsikan di use case diagram dan teks. Relasi anta use case dengan use case lainnya ada tiga jenis, yaitu

1. Relasi *Include*

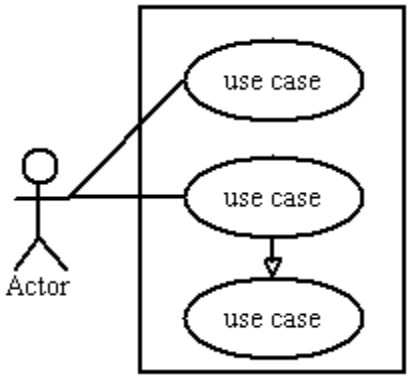
   Merupakan relasi antar use case dimana jika use case A meng-include use case B, maka use case B akan diimplementasikan setiap kali use case A diimplementasikan. Direpresentasikan dengan garis putus-putus bertuliskan <<include>> ke arah use case yang akan di-include.

2. Relasi *Extend*

   Merupakan relasi antar use case dimana jika use case A di extend oleh use case B maka use case B akan bisa saja diimplementasikan atau tidak setiap kali use case A diimplementasikan. Direpresentasikan dengan garis putusputus bertuliskan <<extend>> ke arah use case yang akan di-extend.

3. Relasi *Generalization*

   Digunakan untuk membuat use case lebih spesifik. Relasi *Generalization* dapat dilihat pada Gambar 2.6



Gambar 2. 7 *Use Case Diagram*

2.5.2 *Use Case Scenario*

Use case scenario merupakan hasil instansiasi dan penjelasan dari setiap use case. Use case scenario terbagi menjadi tiga bagian, yaitu:

1. Identifikasi dan inisiasi
2. Step performed
3. Kondisi, asumsi dan pertanyaan.

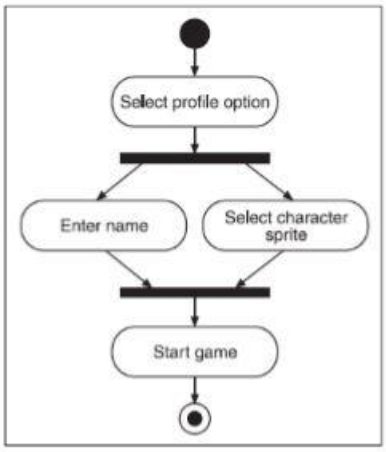
2.5.3 *Activity Diagram*

Activity diagram adalah diagram *flowchart* yang diperluas yang menunjukkan aliran kendali satu aktivitas ke aktivitas lain. Kita menggunakan diagram ini memodelkan aspek dinamis sistem. Aktivitas adalah eksekusi notatomik yang

berlangsung di *state machine*. *Activity diagram* mendeskripsikan aksi-aksi dan hasilnya. *Activity diagram* berupa operasi-operasi dan aktivitas-aktivitas di *use case*. *Activity diagram* dapat digunakan untuk :

1. Pandangan dalam yang dilakukan di operasi.
2. Pandangan dalam bagaimana objek-objek bekerja.
3. Pandangan dalam di aksi-aksi dan pengaruhnya pada objek-objek
4. Pandangan dalam dari suatu use case.
5. Logik dari proses bisnis.

Diagram aktivitas merupakan jenis khusus dan diagram *statechart*. *State* adalah aksi-aksi yang menuju state berikutnya setelah selesainya aksi itu. Contoh activity diagram dapat dilihat pada Gambar 2.7



Gambar 2. 8 *Activity Diagram*

Dari gambar 2.7 di atas merupakan contoh *activity diagram* dengan alur :

1. Lingkaran hitam penuh menandakan memulai aktivitas
2. Lingkaran hitam penuh menandakan memulai aktivitas
3. Masuk ke proses fork untuk menjalankan aktivitas secara pararel kemudian menjalankan aktivitas di bawahnya
4. Masuk ke proses join untuk menggabungkan proses yang dipisahkan oleh fork
5. Melakukan aktivitas Start game

Lingkaran hitam sebagian menandakan akhir dari suatu aktivitas

2.5.4 *Class Diagram*

Class diagram merupakan diagram paling umum dipakai di semua pemodelan berorientasi objek. Pemodelan kelas merupakan pemodelan yang paling utama di pendekatan berorientasi objek. Pemodelan kelas menunjukkan kelas-kelas yang ada di sistem dan hubungan antar kelas-kelas itu, atribut-atribut dan operasi - operasi di setiap kelas [19]. Class diagram menunjuk aspek statis sistem terutama untuk mendukung kebutuhan fungsional sistem. Meskipun class diagram serupa dengan model data, namun kelas-kelas tidak hanya menunjuk struktur informasi tapi juga mendeskripsikan perilaku. Salah satu maksud class diagram adalah untuk mendefinisikan fondasi bagi diagram-diagram lain dimana aspek-aspek lain dari sistem ditunjukkan. Elemen-elemen yang penting dalam class diagram adalah sebagai berikut [19].

1. Kelas

   Kelas merupakan elemen terpenting di sistem berorientasi objek. Kelas mendeskripsikan satu blok pembangunan sistem. Berikut adalah bagian dari kelas :

   a. Nama

      Nama kelas harus unik karena akan menjadi identifier di program.

   b. Atribut

      Atribut adalah properti bernama di kelas yang mendeskripsikan range nilai yang dipunyai instan kelas. Kelas dapat mempunyai sejumlah atribut atau tidak sama sekali.

   c. Operasi

      Operasi adalah implementasi layanan yang dapat diminta pada sembarang objek kelas itu untuk mempengaruhi prilaku sistem.

   d. Acces Modifer

      Access Modifier dalam kelas digunakan untuk membatasi akses dari kelas lain yang ingin mengakses atribut atau operasi dari suatu kelas. Ada tiga *Access Modifier* yang digunakan yaitu *public* (+), *protected* (#) dan *private* (-).

2. Antarmuka

   Antarmuka (Interface) merupakan korelasi operasi yang menspesifisikan layanan dari suatu kelas atau komponen. Antarmuka mendeskripsikan perilaku tampak secara eksternal dari elemen.

3. Kolaborasi

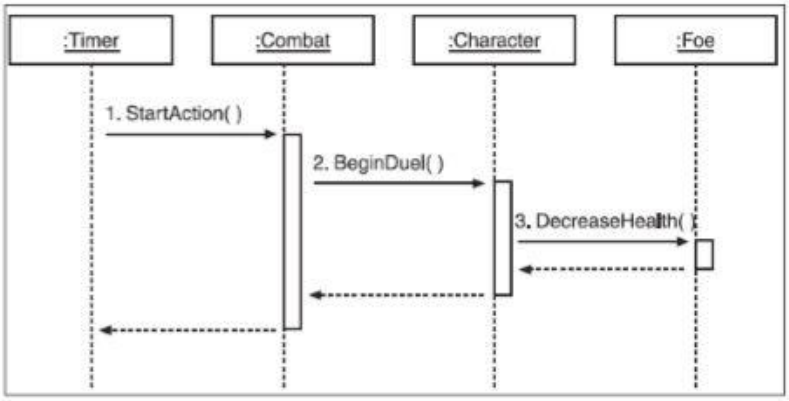
   Kolaborasi merupakan pendefinisian suatu interaksi dan sekelompok peran elemen-elemen lain yang bekerja sama untuk menyediakan suatu perilaku kooperatif yang lebih besar dari penjumlahan seluruh elemen. Kolaborasi memiliki struktur, prilaku dan dimensi. Kolaborasi ini merepresentasikan implementasi pola tertentu yang membentuk sistem.

4. Hubungan

   Hubungan antar kelas di diagram kelas terdiri dari Asosiasi dan Generalisasi.

2.6.1 *Sequence Diagram*

Sequence diagram digunakan untuk memodelkan skenario penggunaan. Skenario penggunaan adalah barisan kejadian yang terjadi selama satu eksekusi sistem. Cakupan skenario dapat beragam, dari mulai semua kejadian di sistem atau hanya kejadian pada objek-objek tertentu. Sequence diagram menunjukkan objek sebagai garis vertikal dan tiap kejadian sebagai panah horizontal dari objek pengirim ke objek pengirim ke objek penerima. Gambar 2.8 merupakan contoh *sequence diagram*.



Gambar 2. 9 *Sequence Diagram*

2.6 Bahasa Pemrograman

Bahasa pemrograman diperlukan untuk menjalankan instruksi-instruksi apa yang harus dilakukan komputer. Komputer tidak bisa memahami bahasa manusia, sehingga diperlukan penggunaan bahasa komputer di dalam program komputer. Penelitian ini menggunakan bahasa pemrograman Java untuk mengembangkan aplikasi dan bahasa pemrograman SQL untuk mengelola *database*.

2.6.1 C#

Bahasa pemrograman C# (baca: C-sharp) dirancang oleh Microsoft Corp. sebagai Bahasa perograman yang sangat berdaya-guna, aman (*secure*), serta mudah digunakan. Sebagai bagian dari platform .NET, bahasa pemrograman C# dirancang sedemikian rupa untuk bekerja dengan sangat baik di atas framework .NET yang mampu digunakan untuk menulis perangkat lunak handal demi layanan yang cepat. Bahasa pemrograman C# juga dapat digunakan untuk membangun aplikasi sarana bergerak (*mobile application*), aplikasi berbasis Web (*Web-based application*), serta aplikasi-aplikasi berskala besar (*enterprise*) [17].

2.6.2 SQL

SQL (*Structured Query Language*) merupakan sebuah bahasa permintaan *database* yang tersetruktur yang dibuat sebagai bahasa yang dapat merelasikan beberapa tabel dalam *database* maupun merelasikan antar *database* [18]. Bahasa SQL ditulis langsung dalam sebuah program *database* sehingga pengguna dapat melihat langsung permintaan yang diinginkan dan sekaligus melihat hasilnya. Ada beberapa program *database server* yang menggunakan standar *query* berupa SQL yaitu : Oracle, PostgreSQL, MySQL, MsQL, SQL *Server*, dan lain-lain. Dalam penelitian ini SQL diimplementasikan pada *database server* MySQL.

2.7 Pengujian Sistem

Pengujian adalah proses pemeriksaan atau evaluasi sistem atau komponen sistem secara manual atau otomatis untuk memverikasi apakah sistem memenuhi kebutuhan-kebutuhan yang dispesifikan atau mengidentifikasi perbedaan-perbedaan antara hasil yang diharapkan dengan hasil yang terjadi [19]. Pengujian seharusnya meliputi tiga konsep berikut.

1. Demonstrasi validitas perangkat lunak pada masing-masing tahap di siklus pengembangan sistem.
2. Penentuan validitas sistem akhir dikaitkan dengan kebutuhan pemakai.
3. Pemeriksaan perilaku sistem dengan mengeksekusi sistem pada data sampel pengujian.

Awalnya pengujian diartikan sebagai aktivitas yang dapat atau hanya dilakukan setelah pengkodean (kode program selesai). Namun, pengujian seharusnya dilakukan dalam skala lebih luas. Pengujian dapat dilakukan begitu spesifikasi kebutuhan telah dapat didefinisikan. Evaluasi terhadap spesifikasi dan perancangan juga merupakan teknik di pengujian. Kategori pengujian dapat dikategorikan menjadi dua [19], yaitu :

1. Berdasarkan ketersediaan logik sistem, terdiri dari *Black box testing* dan *White box testing*.
2. Berdasarkan arah pengujian, terdiri dari Pengujian *top down* dan Pengujian *bottom up*.

2.7.1 Pengujian *Black Box*

Konsep *black box* digunakan untuk merepresentasikan sistem yang cara kerja di dalamnya tidak tersedia untuk diinspeksi. Di dalam *black box*, item-item yang diuji dianggap "gelap" karena logiknya tidak diketahui, yang diketahui hanya apa yang masuk dan apa yang keluar dari *black box* [19].

Pada pengujian *black box*, kasus-kasus pengujian berdasarkan pada spesifikasi sistem. Rencana pengujian dapat dimulai sedini mungkin di proses pengembangan perangkat lunak. Teknik pengujian konvensional yang termasuk pengujian "black box" adalah sebagai berikut [19].

1. *Graph-based testing*
2. *Equivalence partitioning*
3. *Comparison testing*
4. *Orthogonal array testing*

Pada pengujian *black box*, kita mencoba beragam masukan dan memeriksa keluaran yang dihasilkan. Kita dapat mempelajari apa yang dilakukan kotak, tapi tidak mengetahui sama sekali mengenai cara konversi dilakukan. Teknik pengujian

black box juga dapat digunakan untuk pengujian berbasis skenario, dimana isi dalam sistem mungkin tidak tersedia untuk diinspeksi tapi masukan dan keluaran yang didefinisikan dengan *use case* dan informasi analisis yang lain.

2.7.2 Pengujian Akurasi

Akurasi merupakan seberapa dekat suatu angka hasil pengukuran terhadap angka sebenarnya (*true value* atau *reference value*). Tingkat akurasi diperoleh dengan persamaan sebagai berikut.

$$Akurasi = \frac{Jumlah\ Karakter\ Sama}{Jumlah\ Seluruh\ Karakter}\ X\ 100\% \tag{2.21}$$