BAB 2
LANDASAN TEORI

2.1 *Game*

Game atau permainan merupakan media hiburan yang sudah dikenal sejak dahulu. *Game* dapat dimainkan oleh semua kalangan, tua maupun muda. *Game* juga sudah berkembang dengan pesat sesuai dengan perkembangan teknologi, dari *game* sederhana sampai *game* mutakhir saat ini. Hal ini terbukti dengan adanya perkembangan jenis, produk serta alat yang digunakan.

2.1.1 Pengertian *Game*

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

Terdapat dua pengertian *game* (permainan) yaitu, *game* (permainan) sebagai sebuah aktifitas bermain yang murni mencari kesenangan tanpa mencari menang atau kalah dan *game* (permainan) yang diartikan sebagai aktifitas bermain yang dilakukan dalam rangka mencari kesenangan dan kepuasan, namun ditandai pencarian menang atau kalah [5].

Berdasarkan representasinya, *game* dapat dibedakan menjadi 2 jenis yaitu *game* 2 dimensi (2D) dan 3 dimensi (3D). *Game* 2D adalah *game* yang secara matematis 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 [5].

Jenis *game* sangat banyak dan bervariasi, baik media untuk memainkannya yang berbeda, cara bermain dan jumlah pemainnya. Berikut ini adalah jenis *game* berdasarkan tipe *game* yang biasanya dimainkan di handphone dan komputer:

1. Tembak menembak (*First Persong Shooting*)

   Game jenis ini sangat memerlukan kecepatan refleks, koordinasi mata-tangan, juga timing, inti dari game jenis ini adalah tembak, tembak dan tembak. Contohnya GTA, *Call Of Duty*.

2. Pertarungan (Fighting)

   Game yang permainannya memerlukan refleks dan koordinasi mata dan tangan dengan cepat, tetapi inti dari game ini adalah penguasaan hafalan jurus. Contohnya *Mortal Kombat* dan *Tekken*.

3. Petualangan (*Adventure*)

   Game yang lebih menekankan pada jalan cerita dan kemampuan berfikir pemain dalam menganalisia tempat secara visual, memecahkan teka teki maupun menyimpulkan berbagai peristiwa. Contohnya *Kings Quest*, dan *Space Quest*.

4. Simulasi, Konstruksi dan Manajemen

   Game jenis ini seringkali menggambarkan dunia di dalamnya sedekat mungkin dengan dunia nyata dan memperhatikan dengan detil berbagai faktor. Contohnya *The Sims*.

5. Strategi

   Game jenis ini memerlukan koordinasi dan strategi dalam memainkan permainan ini. Kebanyakan game stategi adalah game perang. Contohnya *Warcraft III*.

6. Olahraga (Sport)

   Game ini merupakan adaptasi dari kenyataan, membutuhkan kelincahan dan juga strategi dalam memainkannya. Contohnya *Winning Eleven* dan NBA.

7. Puzzle

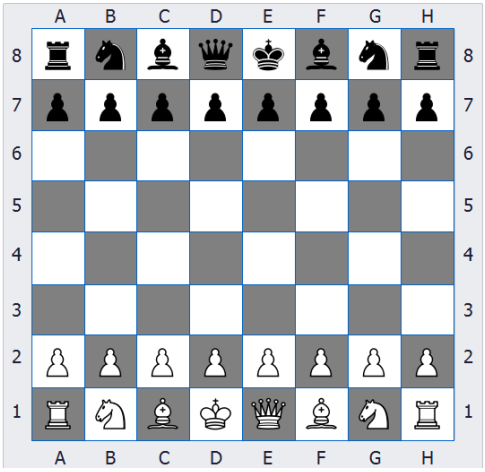
   Game teka-teki, pemain diharuskan memecahkan teka-teki dalam game tersebut. Contohnya *Tetris*, *minesweeper*, *bejeweled*, 2048, *Sudoku* dan *Skyscraper*.

8. Edukasi

Game jenis ini dibuat dengan tujuan spesifik sebagai alat pendidikan, untuk belajar mengenal warna untuk balita, mengenal huruf, angka, belajar matematika, sampai belajar bahasa asing. Developer yang membuatnya, harus memperhitungkan berbagai hal agar game ini benar - benar dapat mendidik, menambah pengetahuan dan meningkatkan ketrampilan yang memainkannya. Contohnya *The Farmer* dan *Dora the explorer*.

2.2 *Game* Catur

Catur adalah *game* pikiran yang dimainkan oleh dua orang. Catur termasuk game strategi yang membutuhkan pemikiran untuk menyelsaikannya. Pecatur adalah orang yang memainkan catur, baik dalam pertandingan satu lawan satu maupun satu melawan banyak orang (dalam keadaan informal). Sebelum bertanding, pecatur memilih bidak catur yang akan dimainkan. Terdapat dua warna yang membedakan bidak catur, yaitu hitam dan putih. Pemegang bidak putih memulai langkah pertama, yang selanjutnya diikuti oleh pemegang bidak hitam secara bergantian sampai permainan selesai.



Gambar 2.1 Tampilan papan dan bidak catur

2.2.1 Gerakan pada *game* catur

Sebelum bertanding, pecatur memilih warna bidak yang akan dimainkan. Pemegang bidak putih memulai langkah pertama, yang dilanjutkan oleh pemegang buah hitam secara bergantian. Setiap langkah hanya boleh menggerakkan satu bidak

saja (kecuali untuk rokade di mana ada dua bidak yang bergerak). Bidak dipindahkan ke petak kosong, ataupun yang telah ditempati oleh bidak lawan, dan berarti menangkapnya lalu memindahkan bidak lawan keluar dari permainan. Ada pengecualian gerakan, yaitu untuk gerakan *en-passant*.

Setiap bidak catur memiliki gerakan yang unik sebagai berikut:

1. Raja (*King)* dapat bergerak satu petak ke segala arah. Raja juga memiliki gerakan khusus yang disebut rokade dan turut melibatkan sebuah benteng.
2. Benteng (*Rook*) dapat bergerak sepanjang petak horizontal maupun vertikal, tetapi tidak dapat melompati bidak lain. Seperti yang telah dibahas, benteng terlibat dalam gerakan rokade.
3. Mentri (*Bishop*) dapat bergerak sepanjang petak secara diagonal, tetapi tidak dapat melompati bidak lain.
4. Ratu (*Queen)* memiliki gerakan kombinasi dari Benteng dan Mentri.
5. Kuda (*Knight*) memiliki gerakan mirip huruf L, yaitu memanjang dua petak dan melebar satu petak. Kudalah satu-satunya bidak yang *dapat* melompati bidak-bidak lain.
6. Pion (*Pawn*) dapat bergerak maju (arah lawan) satu petak ke petak yang tidak ditempati. Pada gerakan awal, pion dapat bergerak maju dua petak. Pion juga dapat menangkap bidak lawan secara diagonal, apabila bidak lawan tersebut berada satu petak di diagonal depannya. Pion memiliki dua gerakan khusus, yaitu gerakan menangkap *en passant* dan *promosi*.

2.2.2 Aturan *game* catur

Pada *game* catur, terdapat aturan – aturan yang membatasi gerakan dan aksi pada bidak catur. Aturan – aturan tersebut sebagai berikut:

1. Bidak kuda merupakan satu – satunya bidak yang dapat meloncati bidak – bidak yang lainnya, tetapi jika belum ada satu pion pun yang maju, bidak kuda tidak dapat melangkahi bidak – bidak yang lainnya.
2. Ada banyak gerakan khusus yang dapat dilakukan oleh bidak – bidak catur jika memenuhi kondisi tertentu.
3. Dalam permainan catur, bidak – bidak mempunyai poin – poin yang mewakili kepentingan keberadaan bidak tersebut. Semakin tinggi poin bidak tersebut,

maka bidak tersebut sangat penting keberadaannya, Berikut merupakan nilai poin bidak catur:

a. Pion: 1
b. Kuda: 3
c. Mentri: 3
d. Benteng: 5
e. Ratu: 9

Dengan nilai seperti itu, maka akan sangat baik jika bidak yang memiliki pion tinggi, tidak dikorbankan begitu saja.

2.2.3 Istilah – istilah pada *game* catur

Rokade (dalam bahasa Inggris disebut *castling*) merupakan gerakan khusus dalam catur di mana, Raja bergerak dua petak menuju Benteng di baris pertamanya, kemudian meletakkan benteng pada petak terakhir yang dilalui Raja. Persyaratan rokade adalah sebagai berikut:

1. Bidak Raja dan Benteng yang akan dilibatkan dalam rokade harus belum pernah bergerak.
2. Tidak ada bidak lain di antara Raja dan Benteng.
3. Raja tidak sedang di-skak, dan petak - petak yang dilalui Raja tidak sedang *diserang* oleh bidak lawan.

Hal - hal berikut ini merupakan kesalahan dalam rokade, yang semestinya tidak berlaku:

1. Bidak benteng yang terlibat rokade sedang diserang.
2. Jika Benteng yang dilibatkan berada di sisi ratu, petak yang berada persis di samping Benteng tersebut tidak boleh dalam serangan.

2.2.4 *En Passant*

Merupakan gerakan khusus ketika pion bergerak dua petak maju dan ada pion lawan yang berada satu petak dalam baris tujuan, maka pion lawan dapat menangkap dan menempati petak yang baru saja dilalui pion tersebut (seolah-olah pion tersebut bergerak satu petak maju). Namun, gerakan ini hanya dapat dilakukan sesaat setelah gerakan pion maju dua petak, atau hak lawan untuk melakukan gerakan *en passant* ini hilang. Contoh gerakan pion: promosi dan *en passant*.

2.2.5 Promosi

Ketika pion telah maju hingga menempati baris paling akhir, berbarengan dengan gerakan maju tersebut, pion yang telah berhasul promosi, harus ditukar dengan bidak berdasarkan keinginan pemain yaitu Ratu, Benteng, Mentri, ataupun Kuda dengan warna yang sama. Pada umumnya, pion dipromosikan menjadi Ratu. Tidak ada peraturan yang membatasi bidak yang dipilih sebagai promosi, jadi dimungkinkan memiliki bidak yang melebihi jumlahnya waktu awal permainan.

2.2.6 Skak

Ketika Raja sedang diserang oleh satu atau lebih bidak lawan, keadaan ini disebut dengan skak**.** Pemain yang Rajanya di-skak harus menggerakkan Rajanya supaya tidak terserang. Hal ini dapat dilakukan dengan menangkap bidak lawan yang menyerang, *menutup* serangan lawan dengan menempatkan sebuah bidak di antaranya (apabila yang menyerang Ratu, Benteng atau Mentri dan ada petak kosong di antara Raja dan bidak lawan), atau memindahkan Raja ke petak yang tidak sedang diserang. Rokade tidak diijinkan apabila Raja sedang di skak.

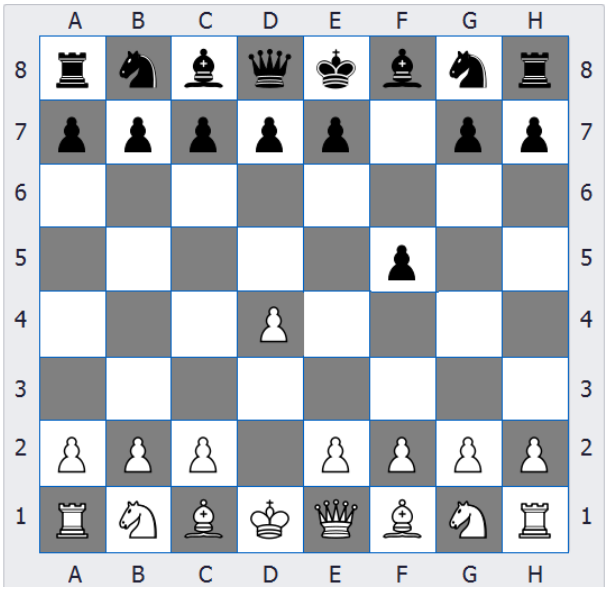
2.2.7 Akhir permainan

Tujuan permainan adalah mencapai posisi skak-mat, yaitu kondisi yang terjadi apabila raja terancam dan tidak bisa menyelamatkan diri ke petak lain.

2.2.8 Strategi dalam bermain catur

Permainan catur merupakan permainan yang termasuk ke dalam kategori permainan strategi. Strategi dalam permainan catur mencakup segala aspek dalam menggerakan bidak catur. Bahkan membuat lawan menunggu pun termasuk ke dalam strategi game catur. Strategi game catur disebut juga dengan pembukaan catur. Pembukaan catur terbagi menjadi bermacam – macam jenis seperti *Gambit benko*, pertahanan belanda, pertahanan prancis, sistem katalan dan lain – lain.

Pada pertahan belanda, jika bidak putih memulai dengan pion4 menuju kotak D – 4, maka bidak hitam akan melawan dengan pion 6 kotak F – 5, seperti gambar 2.2.



Gambar 2.2 Tampilan papan catur pertahanan belanda

2.3 Artificial Intelligence (AI)

Pada landasan teori ini, akan membahas apa saja definisi *Artificial intelligence* menurut para ahli.

2.3.1 Definisi AI

Sebagian kalangan menerjemahkan *Artificial Intelligence* sebagai kecerdasan buatan, kecerdasan artifisial, intelijensia artifisial, atau intelijensia buatan. Istilah *Artificial Intelligence* sengaja tidak diterjemahkan ke dalam bahasa Indonesia karena istilah tersebut sudah sangat akrab bagi orang indonesia. begitu juga dengan singkatannya yaitu AI, sudah sangat melekat di berbagai media ilmiah maupun non - ilmiah.

Terdapat beberapa pegelompokkan definisi AI, yang diperoleh dari beberapa *textbook* yang berbeda dan dibagi ke dalam empat kategori, yaitu [13]:

1. *Thinking Humanly : the cognitive modeling approach*

Pendekatan ini dilakukan dengan dua cara sebagai berikut:

a. Melalui introspeksi : mencoba menangkap pemikiran-pemikiran kita sendiri pada saat kita terpikir. Tetapi, seorang psikolog barat mengatakan "*how do you know that you understand* ?" bagaimana anda tahu bahwa anda mengerti? Karena pada saat anda menyadari pemikiran anda. Sehingga definisi ini terkesan mengada-ngada dan tidak mungkin dilakukan.
b. Melalui eksperimen - eksperimen psikologi.

2. *Acting Humanly : the Turing test approach*

Pada tahun 1950, Alan Turing merancang suatu ujian bagi komputer berintelijensia untuk menguji apakah komputer tersebut mampu mengelabui seorang manusia yang menginterogasinya melalui *teletype* (komunikasi berbasis teks jarak jauh). Jika *interrogator* tidak dapat membedakan yang diinterogasi adalah manusia atau komputer, maka komputer berintelijensia tersebut lolos dari *Turning test*. Komputer tersebut perlu memiliki kemampuan : *Natural Language Processing*, *Knowledge Representation*, *Automated Reasoning*, *Machine Learning*, *Computer Vision*, *Robotics*. *Turing test* sengaja menghindari interaksi fisik antara *interrogator* dan komputer karena simulasi fisik manusia tidak memerlukan intelijensi [13].

3. *Thinking Rationally : the laws of thought approach*

Terdapat dua masalah dalam pendekatan ini, yaitu:

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

4. *Acting Rationally : the rational agent approach*

Membuat inferensi yang logis merupakan bagian dari suatu *rational agent*. Hal ini disebabkan satu-satunya cara untuk melakukan aksi secara rasional adalah dengan menalar secara logis. Dengan menalar secara logis, maka bisa

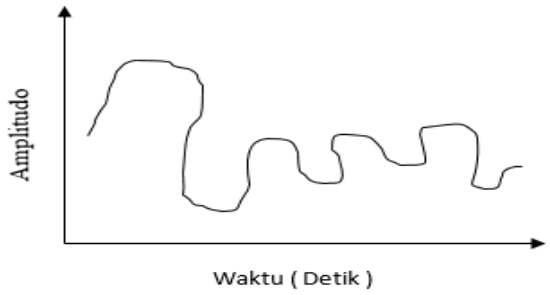
didapatkan kesimpulan bahwa aksi yang diberikan akan mencapai tujuan atau tidak. Jika mencapai tujuan, maka agent dapat melakukan aksi berdasarkan kesimpulan tersebut. *Thinking humanly* dan *acting humanly* adalah dua definisi dalam arti yang sangat luas. Sampai saat ini, pemikiran manusia yang diluar rasio, yakni refleks dan intuitif berhubungan dengan perasaan), belum dapat ditirukan sepenuhnya oleh komputer. Dengan demikian, kedua definisi ini dirasa kurang tepat untuk saat ini. Jika kita menggunakan definisi ini, maka banyak produk komputasi cerdas saat ini yang tidak layak disebut sebagai produk AI. Definisi *thinking rationally* terasa lebih sempit daripada *acting rationally*. Oleh karena itu, definisi AI yang paling tepat untuk saat ini adalah *acting rationally* dengan pendekatan *rational agent*. Hal ini berdasarkan pemikiran bahwa komputer bisa melakukan penalaran secara logis dan juga bisa melakukan aksi secara rasional berdasarkan hasil penalaran tersebut [13].

2.4 Suara

Suara kadang disebut juga sebagai gelombang akustik (gelombang akustik sesungguhnya merupakan kasus khusus dari gelombang elastik pada medium udara atau fluida). Manusia mulai memperhatikan suara sejak lama, bahkan alat musik sudah ada pada zaman mesir, yang kemudian dikembangkan secara terstruktur oleh *al-farabi*, *al-kindi* dan masyarakat *Tiongkok*. Sebuah kenyataan yang cukup unik bahwa pada awalnya, musik yaitu sebuah disiplin yang mempelajari suara dan bunyi-bunyian oleh *al-farabi* digolongkan ke dalam ilmu hitung dan bukan ilmu seni [6].

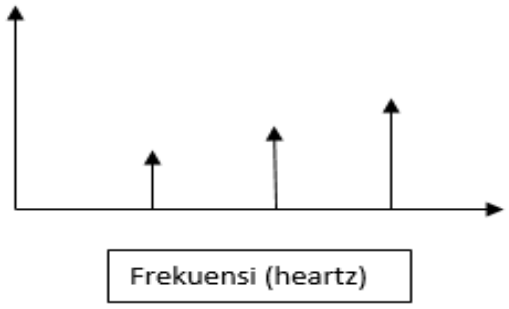
Suara merupakan suatu hal yang unik dan memiliki rentang yang bisa didengar dan tidak bisa didengar oleh manusia, mempunyai frekuensi tertentu dan juga intensitas. Batas frekuensi suara yang dapat di dengar oleh telinga manusia berkisar antara 20Hz (Hertz) sampai 20kHz (Kilo hertz). Satuan untuk mengukur intensitas suara tersebut adalah desibel (dB) diambil dari nama penemunya yaitu *Alexander Graham Bell*, penemu telepon. Satuan dari frekuensi suara adalah Hertz diambil dari nama seorang Fisikawan, *Heinrich Rudolf Hertz* untuk menghargai jasa atas kontribusinya dalam bidang *elektromagnetisme*.

Pengertian pengenalan suara Sinyal suara merupakan suatu sinyal yang dihasilkan oleh getaran suatu benda, yang bentuknya berupa sinyal analog. Sinyal suara dibentuk dari kombinasi berbagai frekuensi, *amplitudo*, dan fasa. Sinyal suara dapat direpresentasikan ke dalam 2 domain, yaitu domain waktu dan domain frekuensi. Dalam domain waktu, sinyal suara direpresentasikan dalam bentuk tegangan atau arus dalam fungsi waktu. Sinyal suara dalam domain waktu ditunjukan pada gambar 2.3 [4].



Gambar 2.3 Domain Waktu

Sedangkan dalam domain frekuensi, sinyal suara direpresentasikan dalam bentuk amplitudo dalam fungsi frekuensi. Dalam domain frekuensi, dapat terlihat frekuensi berapa saja yang terkandung dalam suatu sinyal. Sinyal suara dalam domain frekuensi ditunjukan pada gambar 2.4 [4].



Gambar 2.4 Domain Frekuensi

Sinyal digital merupakan sinyal data dalam bentuk pulsa yang dapat mengalami perubahan dengan tiba-tiba dan mempunyai besaran 0 dan 1. Sinyal

digital biasa juga sinyal diskrit. sinyal digital hanya memiliki dua keadaan yaitu 0 dan 1.



Gambar 2.5 Sinyal digital

2.4.1 Satuan ukur suara

Hertz (simbol : Hz) merupakan satuan ukur international untuk suatu frenkuensi. *Hertz* menyatakan banyaknya gelombang dalam waktu satu detik (1 *Hertz* = 1 gelombang per detik). Unit ini dapat digunakan untuk mengukur gelombang apa saja yang periodik.

2.4.2 Pengenalan suara

Pengenalan suara adalah suatu proses untuk mengenali huruf, kata atau kalimat yang diucapkan. Pengenalan suara lebih dikenal dengan istilah *Automatic Speech Recognition* atau *Computer Speech Recognition* dimana penggunaan sebuah mesin/komputer untuk mengenali sebuah suara atau identitas seseorang dari suara yang diucapkan. Umumnya pengucap berbicara di depan komputer atau mesin, kemudian komputer atau mesin mengenali suara seseorang dengan tepat sesuai yang diucapkan. Pengenalan pola suara dikenali ke dalam berbagai tingkatan tugas, pengenalan dalam tingkat sinyal akustik berupa uji tingkatan dalam susunan unit sub kata berupa fonem, kata, frase dan kalimat. Pengenalan suara huruf vokal merupakan dasar dari pengenalan suara sebab susunan kata merupakan susunan dari beberapa huruf salah satunya adalah huruf vokal sehingga jika diperoleh prinsip dasar proses pengenalan dari suara huruf vokal dapat digunakan dalam penelitian lebih lanjut.

2.4.3 Proses pengenalan suara secara umum

Prinsip kerja dari sistem ini diawali ketika sinyal suara berupa ucapan diberikan oleh *user* untuk memerintahkan suatu tugas tertentu, kemudian sinyal suara ini akan dikodekan dalam serangkaian kode – kode informasi yang disusun

berdasarkan suatu acuan yang sudah ditentukan. Setelah terjadinya pengkodean maka sistem pengenalan suara akan memberikan respon terhadap pengguna dalam bentuk keluaran (*output*) yang berbeda-beda, baik berupa suara maupun melakukan tindakan lain. Dalam penelitian ini, keluaran yang di harapkan berupa gerakan pada bidak catur.

2.4.4 Pengenalan suara pada manusia

Pada sistem pengenalan suara oleh manusia terdapat tiga organ penting yang saling berhubungan, yaitu telinga yang berperan sebagai media penerima sinyal masukan suara dan mengubahnya menjadi sinyal syaraf, jaringan syaraf yang berfungsi mentransmisikan sinyal ke otak, dan otak yang akan mengklasifikasi dan mengidentifikasi informasi yang terkandung dalam sinyal masukan.

2.4.5 Masalah-masalah yang Penting Pada Sistem Pengenalan Suara

Pada kenyataannya, Sistem Pengenalan suara sampai saat ini masih perlu meningkatkan beberapa hal guna memperbaiki kinerjanya, terutama masalah akurasi. Dalam pengenalan suara disebutkan beberapa masalah mengapa sulit meningkatkan akurasi. Masalah tersebut dijelaskan sebagai berikut:

1. Semakin besar ukuran kosakata, semakin tinggi kesalahan dalam pengenalan ucapan. Sebagai contoh, 10 digit dari “nol” sampai “sembilan” dapat dengan mudah dikenali. Tetapi dengan meningkatkannya ukuran *vocabulary* menjadi 200, 5000, atau 100000 maka bisa terjadi kealahan. Sementara itu, setiap bahasa mempunyai huruf yang pada saat pengucapannya sulit untuk dibedakan. Sebagai contoh huruf A dengan E dan B dengan G dalam kasus bahasa Inggris.
2. Persyaratan bahasa yang digunakan. Berbagai pekerjaan sulit untuk dilakukan jika tidak memenuhi syarat-syarat penggunaan bahasa. Termasuk didalamnya adalah Sintaks, Simantik dan Grammarnya. Sebagai contoh, *query* terhadapa jasa penerbangan dengan pernyataan ‘Pesawat itu berwarna merah’, diterima oleh menggunakan *Speaker* tertentu, yang sangat spesifik ucapannya. Hasil Sintaks, Simantik maupun Grammar. Sedangkan pernyataan ‘Pesawat itu sedang marah’, diterima Sintaks dan Grammar tetapi ditolak oleh Simantik.

Sehingga, semua struktur bahasa perlu ditinjau dalam membangun sistem pengenalan suara.

3. Logat bahasa adalah cara mengucapkan kata atau aksen yang khas, dan dimiliki oleh masing-masing orang sesuai dengan asal daerah ataupun suku bangsa. Logat dapat mengidentifikasi lokasi dimana pembicara berada, status sosial - ekonomi, dan lain lainnya. Logat bahasa dapat menjadi masalah serius pada saat *input* suara, karena logat suara dapat membuat kata yang diucapkan menjadi terdengar berbeda.
4. Ucapan Spontanitas. Sistem pengenalan suara dapat ditinjau dari pembicara yang sedang membaca skrip atau berbicara secara spontan. Pembicara spontan seperti emm, uh, e dan lain-lain, lebih sulit dikenali.
5. Kondisi lingkungan. Kinerja system pengenalan ucapan dapat juga di uji melalui lingkungan alamiah yang terdapat banyak interferensi, derau dan lain - lain. Sehingga usaha untuk dapat mengenali ucapan dari pembicara yang dimaksudkan dengan suara bersih telah banyak diteliti dengan berbagai metode. Diantaranya, penghilangan derau, penggandaan mikrofon, pembatas lebar pita frekuensi, dan perubahan gaya ucapan. Bahkan ada kelemahan sistem pengenalan ucapan yang mendasar, yaitu bahasa pembicaraan akan lebih efektif bila diterapkan pada interaksi antara manusia dengan komputer. Hal ini diakibatkan oleh lambatnya komputer merepresentasikan informasi dan sulitnya melakukan *review* dan *edit*.

2.4.6 Analisis sinyal ucapan

Informasi yang terdapat di dalam sebuah sinyal ucapan dapat dianalisis dengan berbagai cara. Beberapa peneliti telah membagi beberapa *level* pendekatan untuk menggambarkan informasi tersebut, yaitu *level akustik, fonetik, fonologi* [6].

1. *Level Akustik*

   Merupakan Sinyal ucapan variasi tekanan udara yang dihasilkan oleh sistem artikulasi. Untuk menganalisa aspek-aspek akustik dari sebuah sinyal ucapan, dapat dilakukan dengan transformasi dari bentuk sinyal ucapan menjadi sinyal listrik dengan menggunakan tranduser seperti *microphone*, telepon, dan sebagainya. Setelah melalui berbagai pengolahan sinyal digital, maka

akan di peroleh informasi yang menunjukkan sifat-sifat akustik dari sinyal ucapan tersebut yang meliputi frekuensi fundamental, intensitas, dan distribusi energi spektral.

2. *Level Fonetik*

   Menggambarkan bagaimana suatu sinyal suara diproduksi oleh organ-organ di dalam tubuh manusia.

3. *Level Fonologi*

   Dikenal istilah fonem yang merupakan unit terkecil yang membentuk sebuah kalimat atau ucapan. Deskripsi ini memuat informasi durasi, intensitas, dan *pitch* dari fonem-fonem yang membangun kalimat tersebut.

2.5 Pre Processing

Sinyal suara yang akan diproses dalam pengolahan sinyal suara ini bersifat analog sehingga agar dapat di kenali oleh komputer, sinyal analog tersebut harus dikonversi menjadi sinyal digital yang berupa urutan angka yang dinamakan *analog to digital conversion.*

2.5.1 Konversi Sinyal Analog ke Digital

Sinyal yang terdapat pada kehidupan sehari-hari disebut dengan sinyal analog. Untuk dapat memproses sinyal analog tersebut, maka harus diubah terlebih dahulu menjadi sinyal digital atau sinyal diskrit. Sinyal analog memiliki sifat kontinu pada domain waktu dan amplitudo, sedangkan sinyal digital memiliki sifat diskrit pada domain waktu dan amplitudo. Untuk mengubah sinyal analog menjadi sinyal digital maka diperluas pada proses digitalisasi. Proses ini terdiri dari 3 tahap, yaitu [6]:

1. *Sampling*

   Sampling mengambil bagian sinyal analog yang kontinu dan membagi menjadi beberapa bagian diskrit dalam domain waktu yang dilakukan dengan cara periodik menganalisis sinyal analog dengan menggunakan frekuensi *sampling* tertentu.

2. *Quantization*

   Quantization akan mengubah sinyal analog menjadi bentuk digital. Tahap ini akan mengambil suatu nilai tertentu untuk mengubah bentuk digital. Sebagai

contoh pada perusahaan telepon membagi sinyal analog menjadi 256 bagian. Hal ini diperlukan sebab sinyal analog memiliki kemungkinan nilai yang tidak terbatas.

3. *Coding*

   Coding mengubah sinyal digital menjadi angka biner. Setelah sinyal di *sampling*, jika menggunakan frekuensi *sampling* 8kHz, maka angka biner yang dihasilkan memiliki jangkauan antara 1 sampai dengan 256, Dengan menggunakan frekuensi *sampling* yang semakin besar maka akan didapatkan tingkat akurasi atau ketelitian yang lebih tinggi. Apabila frekuensi *sampling* yang dipilih lebih rendah dari frekuensi tertinggi yang terdapat pada sinyal maka terjadi suatu fenomena yang disebut dengan *aliasing* dimana fenomena tersebut muncul ketika sinyal digital diubah kembali ke sinyal analog.

2.5.2 Akuisisi Data

Tahap ini adalah proses pengambilan data berupa sinyal suara yang diperoleh dengan cara merekam suara melalui mikrofon yang dihubungkan dengan komputer. Format perekaman suara yang digunakan adalah *.wav* dengan frekuensi sampling *bits/sample, channel* dan durasi rekaman yang sudah ditentukan terlebih dahulu. Setelah semuanya ditentukan, akan dimasukan kedalam rumus dibawah ini:

$$X = FS * dt * \left(\frac{bit}{8}\right) * j \qquad (2.1)$$

Dimana : X: Data sampling sinyal

FS: Frekuensi sampling

dt: Durasi rekaman (detik)

Bit: Jumlah bit resolusi

j: 1 untuk *mono* atau 2 untuk *stereo*

Data sampling yang telah di dapatkan, akan dicari *sample rate* nya dengan rumus sebagai dibawah ini:

$$\textit{Sample rate} = \frac{x}{Ts} \qquad (2.2)$$

Dimana : x: Data sampling sinyal

Ts: Durasi rekaman (detik)

Setelah mendapatkan *sample rate*, maka *sample point* dapat dicari dengan rumus dibawah ini:

Sample point = *sample rate* * waktu pengambilan data (2.3)

2.5.3 *Windowing*

Untuk mengurangi proses *Frame blocking* yang disebabkan rendahnya jumlah *sampling rate*, dimana menyebabkan terjadinya kebocoran spektral (spectral leakage) atau aliasing dan membuat sinyal menjadi *discontinue*. Untuk mengurangi kemungkinan terjadinya kebocoran spektral, maka harus melewati proses *windowing*.

Fungsi *window* yang paling sering digunakan dalam aplikasi speaker recognation adalah *Windowing* atau disebut juga *Hamming Window*. Fungsi ini menghasilkan sidelobe level yang tidak terlalu tinggi (kurang lebih -43dB) selain itu noise yang dihasilkan pun tidak terlalu besar (kurang lebih 1.36 BINS).

Window hamming:

$$W_i = 0{,}54 - 0{,}46\cos(\frac{2\pi i}{N-1}), 0 \leq n \leq N-1) \quad (2.4)$$

Berikut ini adalah representasi fungsi *Window Hamming* terhadap sinyal suara yang telah dimasukan:

$$X_i = D_i * W_i \quad (2.5)$$

Dimana: X_i: Sinyal hasil *windowing*

D_i: nilai sinyal awal

W_i: nilai *window*

2.6 Algoritma Fast Fourier Transform

Transformasi *Fourier* adalah suatu model transformasi yang memindahkan domain spasial atau domain waktu menjadi domain frekuensi. Transformasi *Fourier* merupakan suatu proses yang banyak digunakan untuk memindahkan domain dari suatu fungsi atau obyek ke dalam domain frekuensi. Dalam pengolahan

citra digital atau sinyal suara, transformasi *fourier* digunakan untuk mengubah domain spasial pada citra atau sinyal menjadi domain frekuensi. Analisa – analisa dalam domain frekuensi banyak digunakan seperti filtering, Dengan menggunakan transformasi *fourier*, sinyal suara atau citra dapat dilihat sebagai suatu obyek dalam domain frekuensi.

FFT (*Fast Fourier Transform*) adalah teknik perhitungan cepat dari DFT (*Discrete Fourier Transform*). Transformasi fourier diskrit atau disebut dengan *Discrete Fourier Transform* (DFT) adalah model transformasi fourier yang dikenakan pada fungsi diskrit, dan hasilnya juga diskrit. FFT adalah DFT dengan teknik perhitungan yang cepat dan memanfaatkan sifat periodikal (konjugasi) dari transformasi *fourier*. Perhatikan rumus dari DFT:

$$F(x) = \frac{1}{N}\sum_{n=0}^{N-1} f(n)\ (\cos\left(\frac{2\pi nx}{N}\right) - \text{j}\sin\ \left(\frac{2\pi nx}{N}\right)) \quad (2.6)$$

Dimana: N: Jumlah data
$f(n)$: Nilai diskrit
π: Phi (3,14)

Untuk melihat nilai hasil FFT digunakan rumus:

$$|F(x)| = [R^2 + I^2]^{1/2} \quad (2.7)$$

Dimana: $f(x)$: Hasil perhitungan DFT
R: Bilangan Real
I: Bilangan Imajiner yang mengandung J

Perhitungan FFT mengimplementasikan pencerminan transformasi ganda hasil DFT dengan hanya menghitung nilai setengahnya data sinyal sehingga perhitungan akan lebih cepat, lalu nilai setengahnya lagi dihitung dengan cara *conjugate* nilai yang telah dihitung dengan DFT. Untuk membagi data sinyal adalah dengan rumus:

$$b = \frac{(n+1)}{2} \quad (2.8)$$

Dimana: b: Hasil nilai tengah

n: Jumlah data

2.7 Algoritma Least Mean Square

Algoritma *Least Mean Square* atau filter *adaptif* ini merupakan algoritma penghilang derau atau disebut juga noise. Noise adalah suatu sinyal gangguan yang bersifat akustik (suara), elektris, maupun elektronis yang hadir dalam suatu sistem (rangkaian listrik atau elektronika) dalam bentuk gangguan yang bukan merupakan sinyal yang di inginkan.

Sistem *adaptif* merupakan suatu sistem yang mampu menyesuaikan dan dapat beadaptasi langsung dengan kondisi lingkunganya. *Least Mean Square* (LMS) adalah algoritma dalam *filter adaptif* digunakan untuk memperbaiki koefisien filter (bobot) yang diharapkan yang nanti nya akan menghasilkan *error* yang sangat kecil.

Rumus dari algoritma LMS adalah sebagai berikut:

$$\mathrm{d'(n)} = \mathrm{W_n} * (n) \quad (2.9)$$

$$e(n) = d(n) - d'(n) \quad (2.10)$$

$$\mathrm{W_{n+1}} = \mathrm{W_n} + \mu * \mathrm{e(n)} * \mathrm{(n)} \quad (2.11)$$

Dimana: $\mathrm{W_{n+1}}$: Update koefisien filter

$\mathrm{W_n}$: Koefisien filter yang lama

$d(n)$: Nilai sinyal FFT

μ: Rate convergence

e(n): Error yang dihasilkan

(n): Sinyal masukan ke-n

Dengan alur sebagai berikut :

1. Pada waktu ke-n, bobot filter Wn diketahui.
2. Inisialisai awal: $Wn = 0$
3. Hitung *output* filter: d’(n) = $w(n) * x(n)$
4. Hitung estimasi *error*: e(n) = d(n) – d’(n)
5. Hitung bobot filter berikutnya: w(n+1)= w(n) + μ * e(n) * (n)
6. Lanjutkan ke: n = n+1

Algoritma LMS ini tidak memerlukan proses perhitungan yang rumit karena tidak membutuhkan perhitungan fungsi korelasi maupun perhitungan invers matriks. Sifat-sifat perhitungan yang sederhana ini akan dapat dengan mudah diterapkan dalam bentuk program komputer. Karena kemudahannya inilah algoritma LMS ini menjadi salah satu standar dan paling sering digunakan dalam perhitungan filter adaptif.

2.8 Divide and Conquer

Algoritma *Divide and Conquer* merupakan algoritma yang sangat populer di dunia ilmu komputer. *Divide and Con*quer merupakan algoritma yang berprinsip memecah-mecah permasalahan yang terlalu besar menjadi beberapa bagian kecil sehingga mudah untuk diselesaikan. Langkah – langkah umum algoritma *Divide and Conquer*:

1. *Divide*: membagi masalah menjadi beberapa sub-masalah yang memiliki kemiripan dengan masalah semula namun berukuran lebih kecil (idealnya berukuran hampir sama).
2. *Conquer*: memecahkan atau menyelesaikan masing-masing sub-masalah secara rekursif.
3. *Combine*: menggabungkan solusi masing-masing sub-masalah sehingga membentuk solusi masalah semula.

Objek masalah yang dibagi adalah masukan (*input*) atau *instances* yang berukuran n: tabel (*array*), matriks, dan sebagainya, bergantung pada masalahnya. Tiap – tiap sub-masalah mempunyai karakteristik yang sama (*the same type*) dengan karakteristik masalah asal, sehingga metode *Divide and Conquer* lebih natural diungkapkan dalam skema rekursif. Sesuai dengan karakteristik pembagian dan pemecahan masalah tersebut, maka algoritma ini dapat berjalan baik pada persoalan yang bertipe rekursif (perulangan dengan memanggil dirinya sendiri). Maka dari itu, algoritma ini dapat digunakan dengan cara iteratif (perulangan biasa), karena pada prinsipnya iteratif hampir sama dengan rekursif. Salah satu penggunaan algoritma ini yang paling populer adalah dalam hal pengolahan data yang bertipe *array* (elemen larik). Karena pada umumnya pengolahan *array* pada umumnya selalu menggunakan prinsip rekursif atau iteratif.

Pada peneletian kali ini, menggunakan perhitungan array 2 dimensi dengan penjelasan sebagai berikut:

1. Urutkan data fitur suara dari bagian yang terkecil sampai terbesar.
2. Untuk setiap elemen data acuan pada *database*, dicocokan dengan data baru menggunakan *binary search*.
3. Hitung nilai peramalan *error* terkecil, menggunakan rumus MSE dan hasil terkecil adalah solusi atau *output* yang dihasilkan.

Tahapan menentukan hasil pencocokkan adalah dengan cara membandingkan hasil dari *error* terkecil. Cara mencari *error* terkecil adalah menggunakan rumus *Mean Square Error* dengan cara sebagai berikut:

$$MSE = a - b \qquad (2.12)$$

Dimana: a: Nilai tengah hasil *Divide and Conquer*

b: Nilai data suara baru yang dicocokan

Proses ini akan terus dilakukan berulang kali sejumlah banyaknya data acuan yaitu data pada *database*. Setelah mendapatkan hasil *error* terkecil tersebut, semua data dibandingkan dan *error* yang paling terkecil dari *error* terkecil tersebut merupakah hasil akhir dari pencocokkan menggunakan *Divide and Conquer*.

2.9 Euclidean Distance

Algoritma *Euclidean distance* merupakan metode perhitungan 2 buah titik dalam *Euclidean space*. *Euclidean space* berkaitan dengan *Teorema Phytagoras* dan biasanya diterapkan pada suatu *template array* yang berbasis 1, 2 atau 3 dimensi. Pada permainan catur ini, menggunakan perhitungan *array* 2 dimensi.

Rumus dari algoritma *Euclidean Distance* adalah sebagai berikut:

$$\mathrm{d_{ij}} = \sqrt{\sum_{k=1}^{n} (\mathrm{x_{ik}} - \mathrm{x_{jk}})^2} \qquad (2.13)$$

Dimana: d: Jarak

i: Nilai data masukan

j: Nilai pada template database

k: Data

n: banyak data

2.10 Rata – rata tingkat akurasi

Proses rata – rata tingkat akurasi suara suatu bidak merupakan proses yang menghitung berapakah tingkat akurasi suara suatu metode pencocokkan dalam menggerakan suatu bidak pion, menggunakan rumus rata – rata hitung data berkelompok statistika dengan perhitungan sebagai berikut:

$$X = \frac{\sum_{i-1}^{k} fi.xi}{\sum_{i-1}^{k} fi}\% \qquad (2.14)$$

Dimana: X: Rata – rata hitung data berkelompok

fi: Frekuensi data kelas ke-i

xi: Nilai tengah kelas ke-i

Setelah mendapatkan hasil tingkat akurasi sesuai dengan metode yang digunakan, maka hasil tersebut akan dibandingkan untuk mengetahui algoritma pencocokkan manakah yang lebih baik.

2.11 Pemrograman Berorientasi Objek (OOP)

Istilah OOP sudah sering didengar dalam ilmu pemrograman. Berikut ini istilah-istilah yang sering kali terdengar dalam teknik pemrograman OOP :

1. *Object*

Objek memiliki atribut sebagai status (*state*) dan tingkah laku sebagai *behavior*. Di dalam OOP, state disimpan pada variabel dan tingkah laku disimpan pada method. Dalam bahasa teoretis OOP, Objek berfungsi untuk membungkus data dan fungsi bersama menjadi satu unit dalam sebuah program komputer. Objek merupakan dasar dari modularitas dan struktur dalam sebuah program komputer berorientasi objek.

2. *Class*

Class adalah suatu frame yang merupakan definisi yang memuat data dan metod pengolah data. *Class* juga dapat diartikan sebagai tempat untuk membuat obyek. Di dalam *class* dideklarasikan variable dan method yang dimiliki oleh obyek. Proses pembuatan obyek dari sebuah *class* disebut dengan instantiation. Jadi obyek merupakan hasil instansiasi dari *class*. Obyek disebut juga dengan instance.

Class memiliki anggota yang disebut Anggota *Class* (*class member*) yaitu atribut dan *method*.

3. *Attributes*

Atribut adalah data yang membedakan antara objek satu dengan yang lainnya. Dalam *class*, atribut sering disebut sebagai variabel.

Atribut dibedakan menjadi dua jenis yaitu *Instance Variable* dan *Class Variable*.

1. *Instance variable* adalah atribut untuk tiap objek dari kelas yang sama. Tiap objek mempunyai dan menyimpan nilai atributnya sendiri. Jadi, tiap objek dari *class* yang sama boleh mempunyai nilai yang sama atau berbeda.
2. *Class Variable* adalah atribut untuk semua objek yang dibuat dari *class* yang sama. Semua objek mempunyai nilai atribut yang sama. Jadi semua objek dari *class* yang sama mempunyai hanya satu nilai yang *value* nya sama.

2.11.1 Konsep-konsep *Object Oriented Programming*

Konsep dasar dari Pemrograman Berorientasi *Objek Pemrograman orientasi*, objek menekankan konsep berikut:

1. Enkapsulasi

Enkapsulasi berfungsi untuk memastikan pengguna sebuah objek tidak dapat mengganti keadaan dalam/dari sebuah objek dengan cara yang tidak layak, hanya metode dalam objek tersebut yang diberi izin untuk mengakses keadaannya. Setiap objek mengakses interface yang menyebutkan bagaimana objek lainnya dapat berinteraksi dengannya. Objek lainnya tidak akan mengetahui dan tergantung kepada representasi dalam objek tersebut.

2. Pewarisan *(Inheritance)*

Pewarisan merupakan pewarisan atribut dan method dari sebuah *class* ke *class* lainnya. *Class* yang mewarisi disebut *superclass* dan *Class* yang diwarisi disebut *subclass*. *Subclass* bisa berlaku sebagai *superclass* bagi *class* lainya, disebut sebagai *multilevel inheritance*.

Prinsip dasar *inheritance* yaitu persamaan-persamaan yang dimiliki oleh beberapa kelas dapat digabungkan dalam sebuah *class* induk sehingga setiap kelas

yang diturunkannya memuat hal - hal yang spesifik untuk kelas yang bersangkutan. Keuntungan pewarisan:

1. *Subclass* menyediakan *state/behaviour* yang spesifik yang membedakan dengan *superclass*, sehingga memungkinkan *programmer* untuk menggunakan ulang *source code* dari *superclass* yang telah ada.
2. *Programmer* dapat mendefinisikan *superclass* khusus yang bersifat generik, yang disebut *abstract class* (abstraksi), untuk mendefinisikan *class* dengan tingkah laku dan *state* secara umum.
3. *Polimorfisme* adalah kemampuan suatu obyek untuk mempunyai lebih dari satu bentuk. *Polimorfisme* tidak bergantung kepada pemanggilan subrutin. Metode tertentu yang berhubungan dengan sebuah pengiriman pesan tergantung kepada objek tertentu di mana pesan tersebut dikirim. Sebuah variabel tunggal dalam program dapat memegang berbagai jenis objek yang berbeda selagi program berjalan dan teks program yang sama dapat memanggil beberapa metode yang berbeda di saat yang berbeda dalam pemanggilan yang sama, hal ini berlawanan dengan bahasa fungsional yang mencapai *polimorfisme* melalui penggunaan fungsi kelas pertama.

2.12 UML (Unified Modeling Language)

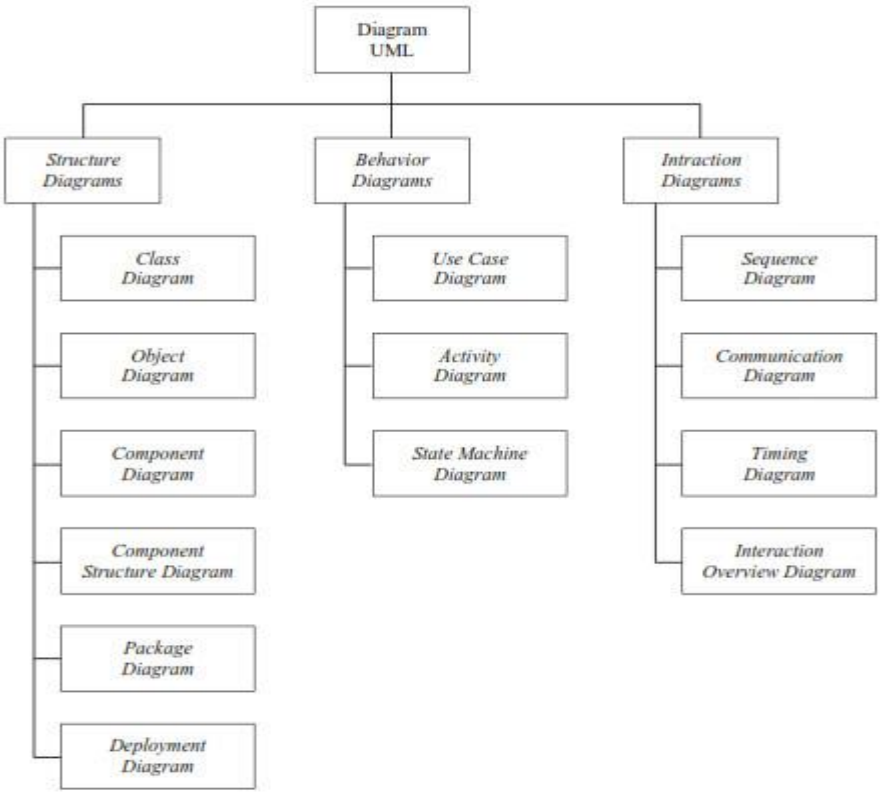
Pada perkembangan teknik pemrograman berorientasi objek, muncul sebuah standarisasi bahasa pemodelan untuk pembangunan perangkat lunak yang dibangun dengan menggunakan teknik pemrograman berorientasi objek, yaitu *Unified Modeling Language* (UML). UML muncul karena adanya kebutuhan pemodelan visual untuk menspesifikasikan, menggambarkan, membangun, dan dokumentasi dari sistem perangkat lunak. UML merupakan bahasa visual untuk pemodelan dan komunikasi mengenai sebuah sistem dengan menggunakan diagram dan teks-teks pendukung.

UML hanya berfungsi untuk melakukan pemodelan. Jadi penggunaan UML tidak terbatas pada metodologi tertentu, meskipun pada kenyataanya UML paling banyak digunakan pada metodologi berorientasi objek [12]. Seperti yang telah

diketahui bahwa banyak hal di dunia informasi yang tidak dapat dibakukan, semua tergantung kebutuhan, lingkungan, dan konteksnya.

Begitu juga dengan perkembangan penggunaan UML bergantung pada level abstraksi penggunanya. Jadi, belum tentu pandangan yang berbeda dalam penggunaan UML adalah suatu yang salah, tapi perlu ditelaah dimanakah UML digunakan dan hal apa yang ingin divisualkan. Secara analogi jika dengan bahasa yang digunakan sehari-hari, belum tentu penyampaian bahasa dengan puisi adalah hal yang salah. Sistem informasi bukanlah ilmu pasti, maka jika ada banyak perbedaan dan interpretasi di dalam bidang sistem informasi merupakan hal yang sangat wajar.

UML terdiri dari 13 macam diagram yang dikelompokkan dalam 3 kategori. Pembagian kategori dan macam-macam diagram tersebut dapat dilihat pada gambar di bawah [12].



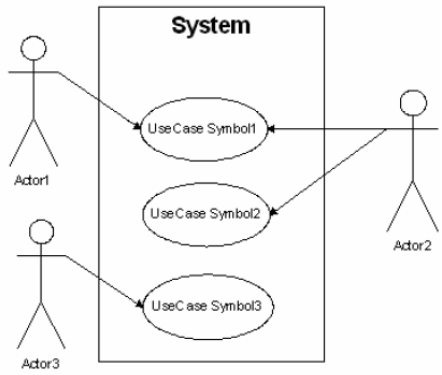
Gambar 2.6 Diagram UML

Berikut ini penjelasan singkat dari pembagian kategori tersebut [12].

1. *Structure* diagram yaitu kumpulan diagram yang digunakan untuk menggambarkan suatu struktur statis dari sistem yang dimodelkan.
2. *Behavior* diagram yaitu kumpulan diagram yang digunakan untuk menggambarkan kelakuan sistem atau rangkaian perubahan yang terjadi pada sebuah sistem.
3. *Interaction* diagram yaitu kumpulan diagram yang digunakan untuk menggambarkan interaksi sistem dengan sistem lain maupun interaksi antar sub-sistem pada suatu sistem.

2.12.1 *Use Case* Diagram

Secara grafik menggambarkan interaksi antara sistem, hal – hal diluar sistem, dan user. Dengan kata lain, mengambarkan siapa yang akan memakai sistem dan dengan cara itu seorang user akan berinteraksi dengan sistem. *Use case* merupakan urutan langkah yang secara tindakan saling terkait (Skenario), baik termotivasi maupun secara manual, untuk tujuan melengkapi satu tugas bisnis tunggal.



Gambar 2.7 *Use Case* Diagram

Notasi yang digunakan dalam *use case* diagram antara lain :

1. *Actor*

Adalah posisi yang dimiliki user terhadap sistem. Yang menjadi actor dapat berupa manusia, hardware atau sistem lain yang berhubungan dengan sistem yng berjalan. Atau actor adalah sesuatu yang memerlukan interaksi dengan sistem untuk bertukar informasi.

2. *Use case* simbol

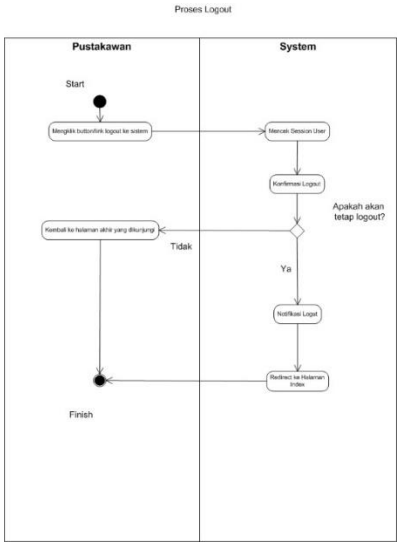
Menggambarkan sekumpulan urutan, dimana setiap urutan mewakili intervensi antara actor dengan sistem. *use case* simbol mempresentasikan fungsionalitas sistem secara keseluruhan.

2.12.1.1 Deskripsi *Use case*

Bagian umum kejadian *use case* adalah langkah demi langkah mulai dengan pelaku menginisialisasi *use case* dan melanjutkan hingga akhir kejadian bisnis. Dimana dalam deskripsi use case terdapat :

1. Nama *use case* merupakan nama dari suatu *use case*.
2. Aktor (*actor*) adalah pelaku yang berpartisipasi didalam suatu *use case*.
3. Tujuan (*goal*) yaitu tujuan dari use-case.
4. Prakondisi (*precondition*) adalah batasan pada keadaan sistem sebelum *use case* dapat dieksekusi.
5. Ringkasan (*summary*) adalah deskripsi ringkasan pendek yang berisi jumlah kalimat yang menunjukan secara garis besar tujuan *use case* dan berbagai kegiatanya.
6. Related *use case* yaitu *use case* yang saling berhubungan.
7. Langkah – langkah (*steps*) yang menjelaskan setiap langkah dari *use case* dengan mengunakan dua kolom, dimana kolom sebelah kiri menunjukan aksi dari pelaku dan kolom sebelah kanan merupakan respon yang diberikan oleh sistem.

2.12.2 *Activity* Diagram



Gambar 2.8 *Activity* Diagram

Activity diagram atau diagram aktivitas adalah sebuah diagram yang dapat digunakan untuk menggambarkan secara grafis aliran proses bisnis, langkah – langkah sebuah *use case* atau logika *behavior object*. Diagram ini serupa dengan *flowchart*, dimana secara grafis diagram ini menggambarkan aliran sekuential dengan *flowchart* dimana diagram ini menggambarkan mekanisme kegiatan paralel.

Oleh karena itu, diagram ini sangat berguna untuk memodelkan kegiatan yang akan dilakukan saat sebuah operasi dieksekusi dan untuk memodelkan hasil – hasil dari kegiatan ini, seperti memodelkan *event* yang menyebabkan *window* akan ditampilkan atau ditutup.

Dibawah ini merupakan simbol-simbol yang ada pada diagram aktivitas:

1. *Initial node*

   Lingkaran yang diisi penuh merupakan awal mulainya dari diagram ini.Initial ode tidak harus ada, tetapi dengan mengunakanya membuat diagramnya lebih mudah untuk dibaca.

2. *Activity*

   Segi empat bersudut tumpul menggambarkan kegiatan yang perlu dilakukan. Secara *activity* bisa secara fisik, seperti *inspect Forms* atau secara elektronik, seperti *Display Create Student Screen*.

3. *Control Flow*

   Panah yang ada di diagram, menggambarkan sasaran yang mengawa kegiatan.

4. *Fork*

   Bar hitam dengan satu flow yang ke dalamnya dan beberapa *flow* lainnya meninggalkannya, merupakan bar sinkronissi dimana kegiatan dapat dilakukan secara paralel.

5. *Join*

   Bar hitam dengan satu *flow* mengarah ke dalamnya dan ada satu *flow* yang meninggalkannya, merupakan bar sinkronisasi dimana beberapa aktivitas yang mempunyai tujuan yang sama dari keduannya digabung menjadi satu.

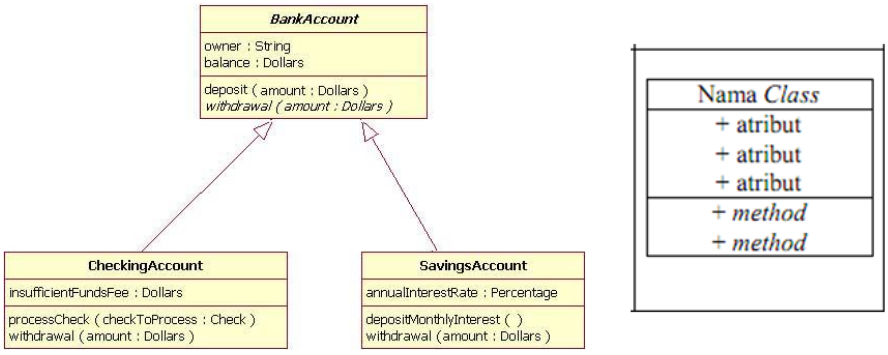
6. *Decicion*

   Gambar sebuah wajik, menggambarkan sebuah kegiatan keputusan.

7. *Final Node*

   Gambar lingkaran yang diisi penuh dan ada batas lagi di luarnya, menggambarkan akhir dari sebuah proses.

2.12.3 *Class* Diagram



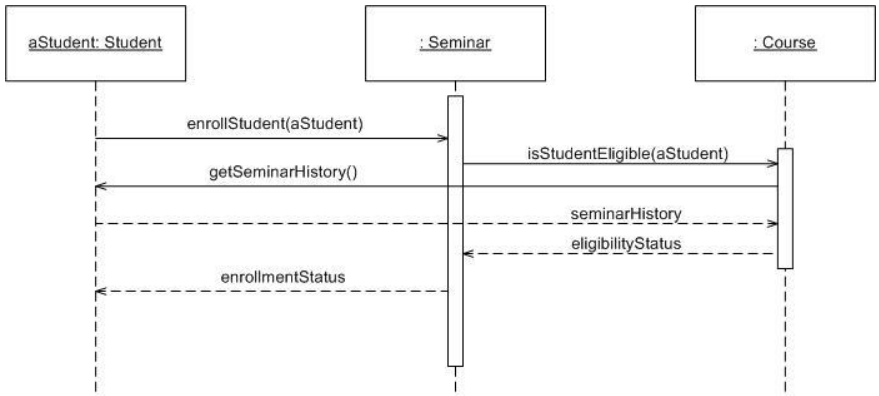
Gambar 2.9 *Class* Diagram

Class diagram merupakan salah satu diagram struktur statis yang menggambarkan struktur hubungan antar kelas. *Class diagram* digunakan untuk mensimulasikan objek-objek dalam dunia nyata ke dalam sistem yang akan dibangun. Notasi UML pada class diagram adalah sebuah persegi yang dibagi menjadi 3 area, yaitu nama kelas, atribut, dan operasi atau *method*.

Dalam sistem yang akan dibangun, kelas-kelas akan dihubungkan secara konseptual. Hubungan ini disebut asosiasi.

Class diagram dapat juga menggambarkan keanekaragaman (*multiplicity*), yaitu jumlah objek dari suatu kelas yang berhubungan dengan sebuah objek dari kelas yang berasosiasi.

2.12.4 *Sequence* Diagram



Gambar 2.10 *Sequence Diagram*

Sequance diagram digunakan terutama untuk menunjukan interaksi antar objek dalam urutan sekuensial. *Sequance diagram* sangat berguna untuk mengkomunikasikan bagaimana objek - objek berinteraksi dalam suatu proses bisnis. Analisis sistem umum nya mengunakan *sequance diagram* untuk memperjelas *use case*.

Sequance diagram terdiri dari objek - objek yang digambarkan dengan sebuah persegi yang memiliki nama. Objek - objek tersebut diletakan diatas garis hidup tersebut, ada kotak kecil memanjang yang dinamakan aktivasi. Aktivasi mempresentasikan eksekusi dari operasi yang objek lakukan. Suatu objek dapat

bertukar pesan dengan objek lain. Pesan tersebut digambarkan sebagai sebuah panah dari garis hidup suatu objek ke objek lain. Ada empat jenis pesan yang bisa digunakan, yakni pesan sederhana (*simple message*), pesan sinkron (*synchronous message*), pesan asinkron (*asynchronous message*), dan pesan nilai pengembalian (retrun value). Pesan sederhana digunakan untuk memindahkan kontrol dari satu objek ke objek lain. Pesan sinkron digunakan apabila suatu objek harus menunggu jawaban dari objek lain (yang dipanggil) terhadap pesan tersebut sebelum melanjutkan proses lainnya.

Pesan sinkron digunakan apabila suatu objek tidak perlu menunggu jawaban dari objek lain sebelum melanjutkan proses lainnya. *Sequance diagram* merepresentasikan waktu dan arah vertikal, dari atas ke bawah. Pesan yang lebih atas menandakan bahwa pesan tersebut terjadi terlebih dahulu.

2.13 *Tools* Yang Digunakan

Dalam pembuatan aplikasi game catur berbasis dekstop ini, menggunakan *Microsoft Visual Studio 2012* sebagai pembuatan aplikasi dan menggunakan *tools DevExpress Universal*.

2.13.1 *Microsoft Visual Studio*

Microsoft Visual Studio merupakan sebuah perangkat lunak yang dapat digunakan untuk melakukan pembuatan maupun pengembangan aplikasi, baik itu aplikasi bisnis, aplikasi personal, ataupun komponen aplikasinya, dengan bentuk aplikasi *console*, aplikasi *windows*, ataupun aplikasi *web*. *Visual Studio* mencakup *kompiler, SDK, Integrated Development Environment (IDE),* dan dokumentasi (umumnya berupa *MSDN Library*). Kompiler yang dimasukkan ke dalam paket Visual Studio antara lain adalah : *Visual C++, Visual C#, Visual Basic, Visual Basic .NET, Visual InterDev, Visual J++, Visual J#, Visual FoxPro, dan Visual SourceSafe*.



Gambar 2.11 Logo *Microsoft Visual Studio*

2.13.2 *DevExpress Universal*

DevExpress Universal merupakan *tools* yang dapat digunakan sebagai editor untuk tampilan pada aplikasi yang akan dibuat. Menu *ribbon* seperti *office 2007/2010*, *autohide docking panel*, *report* dan *preview* (termasuk *export* ke berbagai file format), *datagrid* dengan berbagai macam *view*, *auto seach*, *auto sort* dan *drag drop auto grouping* serta masih banyak lagi fitur yang lainnya yang disediakan pada *tools* ini.

2.13.3 *Audacity*

Audacity adalah aplikasi editor sinyal suara yang dapat digunakan untuk berbagai macam kebutuhan, seperti merekam sinyal suara, menghitung data sinyal suara, dan lain – lain.



Gambar 2.12 Logo *Audacity*

2.14 Metode Pengujian Sistem

Metode pengujian sistem untuk mengetahui efektifitas dari *software* yang digunakan selain memberikan kesempatan kepada pengguna untuk mengoperasikan dan melakukan pengecekan terhadap laporan yang dihasilkan melalui *software*. Metode pengujian sistem terdiri dari Pengujian alpha yang menerapkan pengujian *Whitebox*, Pengujian *Blackbox* dan pengujian beta [10].

2.14.1 Skenario pengujian alpha

Skenario pengujian alpha menjelaskan pengujian terhadap sistem yang dibangun, dimana dilakukan pengujian *white box* dan pengujian *black box.* Metode yang digunakan dalam pengujian *white box* ini adalah metode *Basis Path*. Metode *Basis Path* mengijinkan pendesain kasus uji untuk membuat perkiraan lojik yang kompleks dari desain prosedural dan menggunakan perkiraan ini untuk mendefinisikan aliran eksekusi.

2.14.1.1 *Whitebox*

Pengujian *whitebox* dilakukan untuk menguji prosedur-prosedur yang ada. Lintasan lojik yang dilalui oleh setiap bagian prosedur diuji dengan memberikan kondisi/loop spesifik. Pengujian *whitebox* menjamin pengujian terhadap semua lintasan yang tidak bergantungan minimal satu kali, mencoba semua keputusan lojik dari sisi "*true*" dan "*false*", eksekusi semua *loop* dalam batasan kondisi dan batasan operasionalnya dan pengujian validasi data internal. Konsep Pengujian Basis *Path* sebagai berikut :

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

Untuk menentukan banyaknya jalur independen pada *Basis Path* dapat digunakan *cyclomatic complexity*. Jalur independen adalah jalur yang melalui program yang menghasilkan sedikitnya satu rangkaian statemen proses baru atau suatu kondisi baru. Untuk mencari *cyclomatic complexity*, menggunakan rumus sebagai berikut:

$$\mathrm{V(G)} = edge - node + 2 \qquad (2.15)$$

Dimana: $edge$: Jumlah baris pada program

node: Jumlah baris pada diagram alir

Dan untuk mencari *Predicate note* suatu *graph matriks* menggunakan rumus sebagai berikut:

$$\mathrm{V(G)} = Jumlah\ Graph\ matriks + 1 \qquad (2.16)$$

2.14.1.2 Pengujian *Blackbox*

Black Box Testing adalah tipe pengujian yang memperlakukan perangkat lunak yang tidak diketahui kinerjanya pada bagian *internal* [11]. Para *tester* memandang perangkat lunak sebagai "kotak hitam" yang tidak penting di lihat isinya, tapi cukup dikenali proses *testing* di bagian luar, dimana *testing* ini banyak berkonsentrasi pada bagaimana proses berjalannya perangkat lunak. Pada *black box testing* atau pengujian fungsional, kondisi pengujian dikembangkan berdasarkan fungsionalitas dari program atau sistem yang akan diuji, oleh karena itu, penguji membutuhkan informasi mengenai data input dan output yang diamati, tetapi tidak mengetahui bagaimana program atau sistem tersebut bekerja atau tidak perlu mengetahui bagaimana struktur internal dari program tersebut melakukan eksekusi [14].

Penguji berfokus pada pengujian fungsionalitas dari program terhadap spesifikasi, juga memfokuskan pada *requirement* fungsi dari perangkat lunak, pengujian ini merupakan komplenetari dari pengujian *whitebox*. Pengujian *whitebox* dilakukan terlebih dahulu pada proses pengujian, sedangkan pengujian *blackbox* dilakukan pada tahap akhir dari pengujian perangkat lunak.

Proses yang terdapat dalam proses pengujian *blackbox* antara lain sebagai berikut:

1. Pembagian kelas data untuk pengujian setiap kasus yang muncul pada pengujian *whitebox*.
2. Analisis batasan nilai yang berlaku untuk setiap data.

2.14.2 Pengujian Beta

Pengujian beta merupakan, pengujian perangkat lunak yang dilakukan oleh *user* dalam lingkungan yang sebenarnya. Pengembangan biasanya tidak ada pada pengujian ini. *User* merekam semua masalah (*real* atau *imajiner*) yang ditemukan selama pengujian dan melaporkan pada pengembang pada interval waktu tertentu.

2.15 Penarikan kesimpulan

Penarikan kesimpulan merupakan tahapan terakhir dari proses pengujian sistem. Pada tahapan ini, hasil dari pengujian sistem akan diambil dan ditarik kesimpulan yang menjadi tolak ukur dari penelitian ini.