BAB 2
LANDASAN TEORI

1.1 Artificial Intelligence

Kecerdasan buatan atau dalam bahasa Inggrisnya *Artificial Intelligence* sering disingkat AI merupakan cabang terpenting dalam dunia komputer. Akhir-akhir ini teknologi AI telah begitu banyak mempengaruhi kehidupan manusia.

AI ialah ilmu dan rekayasa, yang membuat mesin mempunyai intelegensi tertentu khususnya program komputer yang 'cerdas'. Intelegensi merupakan bagian kemampuan komputasi untuk mencapai tujuan di dalam dunia. Ada beberapa macam jenis dan derajat intelegensi untuk manusia, binatang dan beberapa mesin.

AI berhubungan dengan penggunaan komputer untuk melaksanakan tugas atau menyelesaikan masalah yang memerluakan kecerdasan atau kepintaran. Pada umumnya komputer memerlukan aplikasi atau software untuk melaksanakan tugas atau menyelesaikan masalah, maka aplikasi atau software yang berdasarkan teknologi AI diperlukan.

Para ahli mendefinisikan AI secara berbeda-beda tergantung pada sudut pandang mereka masing-masing. Ada yang focus pada logika berpikir manusia saja, tetapi ada juga yang mendefinisikan AI secara lebih luas pada tingkah laku manusia. Stuart Russel dan Peter Norvig mengelompokkan definisi AI, yang diperoleh dari beberapa textbook berbeda, ke dalam empat kategori [8], yaitu :

1. *Thinking humanly : the cognitive modeling approach*
2. *Acting humanly : the Turing test approach*
3. *Thinking rationally : the laws of thought approach*
4. *Acting rationally : the rational agent approach*

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.

2.1.1 Algoritma

Algoritma adalah proses langkah-demi-langkah yang menghasilkan solusi untuk masalah AI. Kami akan melihat algoritma yang menghasilkan rute melalui tingkat permainan untuk mencapai tujuan, algoritma yang bekerja keluar arah mana yang harus bergerak untuk mencegat musuh melarikan diri, algoritma yang mempelajari apa yang pemain akan dilakukan selanjutnya, dan banyak lainnya. Struktur data adalah sisi lain dari koin algoritma. Mereka menyimpan data sedemikian rupa bahwa algoritma cepat dapat memanipulasi untuk mencapai solusi. Seringkali, struktur data harus disetel khusus untuk satu algoritma tertentu, dan kecepatan eksekusi mereka secara intrinsik terkait [9].

2.1.2 Action Prediction

Hal ini sering berguna untuk dapat menebak apa yang pemain akan dilakukan selanjutnya. Apakah itu menebak yang bagian mereka akan mengambil, yang senjata mereka akan memilih, atau rute mana mereka akan menyerang dari, permainan yang dapat memprediksi tindakan pemain bisa me-mount oposisi lebih menantang.

Manusia terkenal buruk di berperilaku secara acak. Penelitian psikologi telah dilakukan selama puluhan tahun dan menunjukkan bahwa kita tidak dapat secara akurat mengacak tanggapan kita, bahkan jika kita secara khusus mencoba. Pikiran penyihir dan pemain ahli poker memanfaatkan ini. Mereka dapat dengan mudah sering bekerja di luar apa yang akan kita lakukan atau pikirkan berikutnya berdasarkan jumlah yang relatif kecil dari pengalaman apa yang telah kita lakukan di masa lalu. Seringkali, bahkan tidak perlu untuk mengamati tindakan pemain yang sama. Kami telah berbagi karakteristik yang berjalan begitu mendalam

bahwa belajar untuk mengantisipasi tindakan satu pemain sering dapat menyebabkan bermain lebih baik melawan pemain yang sama sekali berbeda [9].

2.1.3 N-gram

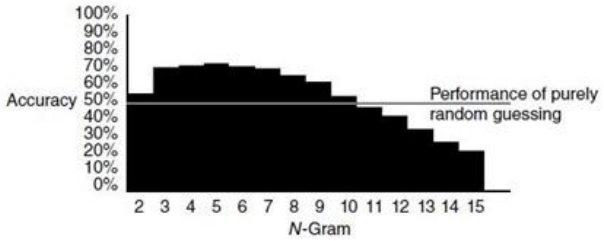
Teknik pencocokan string jarang dilaksanakan oleh mencocokkan string. Hal ini lebih umum untuk menggunakan satu set probabilitas mirip dengan probabilitas mentah di bagian sebelumnya. ini dikenal sebagai prediktor N-Gram (di mana N adalah salah satu lebih besar dari parameter ukuran jendela, sehingga 3-Gram akan menjadi prediktor dengan ukuran jendela dua dari tiga).

Dalam N-Gram kita mencatat probabilitas membuat setiap langkah yang diberikan semua kombinasi. Di dalam N-gram terdapat *window size, sequence length,* dan *memory concerns* [9]. Berikut adalah algoritma metode N-gram pada *game*:

1. Memasukkan gerakan dari *player* ke dalam array sehingga terbentuk *sequence*
2. Memotong-motong *sequence* yang terbentuk sebanyak n
3. Menghitung frekuensi kemunculan *pattern* yang terbentuk (*key*).
4. Setelah didapat *key* nya, selanjutnya mencocokkan *window* terakhir dengan *key* yang ada
5. Setelah ditemukan yang cocok, selanjutnya mengambil dan mencari frekuensi yang paling tinggi
6. Tercipta prediksi akhirnya.

2.1.3.1 Windows size

Window size adalah banyak data yang digunakan untuk memprediksi gerakan *player* selanjutnya. *Window size* sendiri selalu bernilai n-1. Jadi untuk 3-Gram maka *window size* nya adalah 2, sementara untuk 4-Gram *window size*nya 3, dan seterusnya memperbesar *window size.*



Gambar 2.1 Different Window Sizes [9]

Diagram di atas diambil dari hasil eksperimen yang telah dilakukan, dalam 1000 kali trials pada *game* Left or Right. Akurasi akan optimal jika nilai N nya antara 2 sampai dengan 9. Selebihnya nilai akurasinya justru malah dibawah jika kita melakukan random [9].

2.1.3.2 Memory Concerns

Memory Concerns adalah banyak data yang disimpan dalam N-gram.prediksi akan menajdi lebih akurat jika AI menyimpan banyak data dari hasil aksi seorang *player*. Jadi AI yang menyimpan 1000 data aksi *player* tentu prediksinya akan lebih akurat dibanding yang hanya menyimpan 100 data. Hanya saja, tentu akan berpengaruh kepada performa *game* itu sendiri, *game*nya menjadi lebih lambat, memakan *memory* lebih besar, hang ketika si AI melakukan prediksi [9].

2.1.3.3 Sequence Length

Sequence length adalah panjang urutan yang digunakan dalam N-gram. Pada awalnya prediksi tidak akan pernah akurat, ini karena belum ada data yang masuk sehingga semua kemungkinan masih dapat terbentuk, atau masih mempunyai frekuensi yang masih terlalu kecil dan juga nilainya masih sama satu sama lain. Jika nilai N nya besar, maka pembentukan pattern juga akan menjadi lebih lama karena kombinasi pattern nya lebih banyak, sehingga akhirnya proses untuk menuju prediksi yang optimal juga menjadi lebih lambat [9].

2.2 Futsal

Futsal Adalah kata yang digunakan secara internasional untuk permainan sepakbola dalam ruangan. Kata itu berasal dari kata *FUTbol* atau *Futebol* dari

bahasa Spanyol atau Portugal yang berarti permainan sepakbola dan *SALon* atau *SALa* dari bahasa Prancis atau spanyol yang berarti dalam ruangan. Sehinga dapat diartikan futsal yaitu permainan sepakbola dalam ruangan yang berjumlah lima orang tiap tim nya

2.2.1 Sejarah Futsal

Secara resmi, badan sepakbla dunia FIFA menyebutkan futsal pertama kali dimainkan di Montevideo, Uruguay tahun 1930. Saat itu, Juan Carlos Certani memperkenalkan pertandingan sepakbola lima lawan lima untuk suatu kompetisi bagi remaja. Pertandingan itu dilakukan di lapangan basket. Pertandingan itu tidak menggunakan dinding pembatas, artinya ada kesempatan bola keluar lapangan dan terjadi tendangan ke dalam. Saat itu pertandingan dilakukan di dalam ruangan maupun di luar.

Mulanya, Juan Carlos Ceriani yang berasal dari argentina menjadi pelatih di Montevideo. Hujan yang sering mengguyur Montevideo membuat kesal. Semua proses latihan dari jadwal yang sudah disusun berantakan. Kalau hujan gerimis, mungkin ia masih akan melanjutkan latihan. Namun, hujan yang mengguyur amat deras sehingga membuat lapangan tergenang air. Jadwal latihan berantakan, latih tanding dibatalkan dengan sebab yang sama. Mulanya ceriani memikirkan penyelesaian masalah dengan memindahkan tempat latihan aman dari hujan, tidak becek, dan bebas banjir. Mulanya ia tetap menggunakan aturan pemain tiap tim, yakni 11. Namun, permainan dalam ruangan ini diubah sedikit demi sedikit. Karena lapangan sempit, ia mengurangi jumlah pemain menjadi lima untuk setiap tim. Karena menarik ia member tantangan baru, permainan itu digemari masyarakat Montevideo. Penggemar sepakbola di kota itu kemudian beramai-ramai mencoba permainan itu. Jadilah futsal sebagai permainan yang tersebar meluas. Sejarah futsal versi FIFA itu tidak diterima begitu saja. Versi lain mengatakan bahwa tahun 1854 permainan sejenis sudah dilakukan di kanada. Masyarakat Berasil juga tidak setuju dengan sejarah versi FIFA. Mereka mengklaim bahwa pada saat yang bersamaan dengan cerita tentang Ceriani, masyarakat berasil juga melakukan permainan yang sama. Hanya saja mereka

bermain di jalan-jalan. Permain di Brasil juga tidak memakai aturan permainan yang jelas. Ada yang tetap menggunakan peraturan sepakbola. Ada juga yang mengombinasikan dengan basket. Bahkan ada juga yang tidak pakai peraturan, artinya terserah saja. Di masing-masing daerah Brasil terdapat peraturan yang berbeda-beda. Keadaan ini membuat petinggi sepakbola brasil merasa kesulitan dalam mengembangkan futsal. Pada tahun 1935, dibuatlah kesepakatan dan penetapan aturan tentang cara futsal dimainkan. Paling tidak, harus ada perbedaan antara futsal dengan sepakbola, begitu kesepakatan mereka. Akhirnya, pada pertengahan 1936 mereka mengeluarkan peraturan futsal yang tidak jauh berbeda dengan peraturan permainan futsal yang berlaku saat ini. Peraturan itu memang bukan peraturan resmi. Hanya saja, dengan peraturan yang ada, futsal semakin menyebar ke mana-mana. Futsal semakin digemari.

Kira-kira dua puluh tahun kemudian, tepatnya 1954, peraturan futsal dibakukan. Perkembangan futsal di Brasil amat pesat. Klub-klub futsal bermunculan dan membentuk kompetisi futsal pertama di dunia. Tak heran prestasi timnas futsal Brasil amat hebat. Sejak itu , futsal menyebar ke seluruh penjuru dunia. Amerika latin menjadi kawasan yang terjangkit virus futsal. Semua orang di sana bermain futsal. Pada tahun 1965 di Sao Paulo diselenggarakan turnamen antarnegara Amerika Latin. Tuan rumah Brasil yang diunggulkan secara mengejutkan tumbang di tangan Paraguay. Tak lama kemudian, futsal berkembang di Eropa. Di italia perkembangan futsal dimulai tahun 1960-an. Saat itu, futsal dibawa oleh pemain impor dari Amerika Ltin yang berlaga di seri A. Saat-saat senggang, para pemain itu memainkan futsal. Akhirnya. Futsal digemari di Itali.

Lain halnya dengan inggris, futsal mengalami sedikit modifikasi. Klub-klub Premier Leageu seing mangadakan latih tanding enam lawan enam dilapangan rumput. Akhirnya diselenggarakan turnamen semacam itu dengan sponsor London Express, harian terkenal di kota London. Spanyol paling cepat menerima futsal. Gara-gara budaya mereka memang mirip dengan Amerika Latin. Perkembangan futsal juga merambah ke Eropa Timur. Saat itu, penduduk di Uni Soviet terlibat asik dengan futsal. Pada tahun 1974, berkumpulah perwakilan

futsal dari berbagai Negara. Pertemuan yang diadakan di Sao Paulo itu menyepakati pembentuakan FIFUSA(*the Federaco international de Futebol de salao)* sebagai organisasi resmi yang mewadahi futsal. Saat itu, Joao Havelange menjadi ketua umumnya. Setelah terbentuk FIFUSA, fytsal semakin cepat menyebar ke seluruh penjuru dunia. Penyebaran ke Asia, Afrika, Amerika Utara amat pesat pada tahun 1980-an. Akan tetapi FIFUSA menjadi vakum setelah tahun 1989 FIFA mengambil alih futsal dan mengganti peraturan yang telah ada. Tidak Cuma itu, nama futsal diciptakan utuk menyeragamkan nama yang ada. Saat belum terbentuk nama futsal, memang permainan lima lawan lima ini memiliki banyak nama. Ada yang menyebut *five-a-side-game.* Karena memang masing-masing tim terdiri dari lima orang. Ada juga menyebut *mini soccer* karena seperti sepakbola mini. Yang memainkan di jalanan menyebutkan *street soccer.* Ada juga yang menyebutkan *indoor soccer* atau *beach soccer.*

Sejak saat itu, aturan baru ditetapkan oleh FIFA. Sebagai missal, bola yang digunakan berukuran lebih besar daripada sebelumnya, tetapi dengan berat yang tetap. Wasit kedua juga diperkenalkan untuk menggantik hakim garis. Peraturan lain adalah tidak adanya sistem *offside*. Lemparan ke dalam saat bola keluar juga diganti dengan tendangan ke dalam. Selain itu, jumlah pergantian pemain juga dibebaskan. Peraturan-peraturan itu mempercepat penyebaran futsal secara resmi ke berbagai penjuru dunia. FIFA lalu menyelenggarakan Piala Dunia Futsal pertama di belanda tahun 1989. Pada piala dunia pertama itu Brasil itu berasil merebut juara. Penyelenggaraan berikutnya diadakan di Hong kong pada tahun 1992. Di Hong kong, tim nas Brasil kembali menjadi juara dunia. Lagi-lagi Brasil menjadi juara saat Piala Dunia ketiga di Spanyol tahun 1996. Tak mau kalah Spanyol merebut juara saat Piala Dunia Futsal diselenggarkan di GGuetamala tahun 200. Piala Dunia Futsal kelima di Taiwan tahun 2004 lalu, Spanyol masih mempertahankan gelar juara. Tak mau kalah dengan Piala Dunia Futsal, komite eksekutif UEFA yang mewadahi organisasi sepakbola di Eropa memperkenalkan UEFA *Europan Championship* sebagai kompetisi resmi di Eropa [5].

2.2.2 Formasi Futsal

Untuk mengefektifkan suatu *teamwork*, pelatih atau pemain harus menetapkan formasi, taktik dan strategi yang tepat. Ada beragam formasi yang digunakan untuk menangani pertandingan, tetapi dengan beberapa pola formasi permainan pada umum nya yaitu, (2-2),(4-0),(1-3) [6].

1. Formasi (2-2).

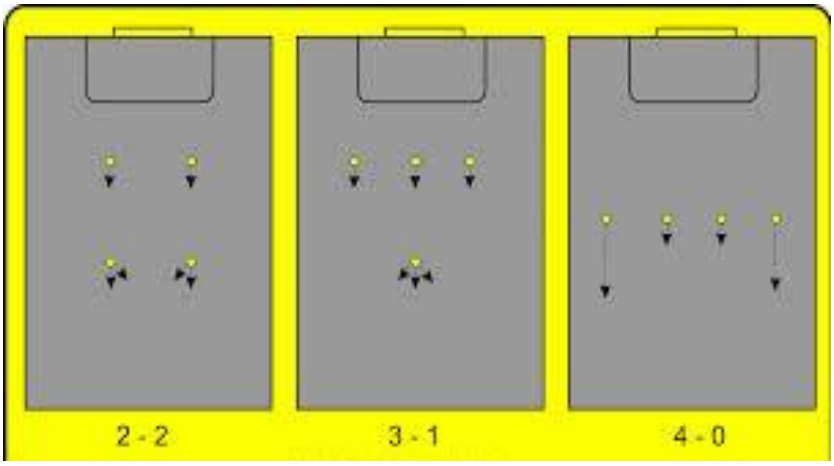
   formasi ini adalah formasi dengan 2 pemain belakang dan 2 pemain depan, gunannya adalah untuk penyeimbang permainan. saat menyerang, pemain belakang bisa maju 1 orang untuk membantu serangan, demikian sebaliknya, saat bertahan, 1 orang pemain depan dapat mundur untuk membantu pertahanan

2. Formasi (4-0).

   Formasi ini adalah formasi dengan 4 main di belakang atau bertahan *man to man marking*.

3. Formasi (1-3).

   Formasi ini dengan 1 pemain belakang dan 3 pemain depan, gunanya untuk penyerangan.



Gambar 2.2 Formasi Futsal Umum

2.3 Game

Game adalah kegiatan yang memiliki tujuan untuk dicapai dan dibatasi dengan peraturan bagi para pemainnya yang bersifat menyenangkan dan untuk membuat *game* maka haruslah memahami teknik dan metode animasi [7]. *Game*

bertujuan untuk menghibur, biasanya *game* banyak disukai oleh anak – anak hingga orang dewasa. *Game* sebenarnya penting untuk perkembangan otak, untuk meningkatkan konsentrasi dan melatih untuk memecahkan masalah dengan tepat dan cepat karena dalam *game* terdapat berbagai konflik atau masalah yang menuntut untuk dilesaikan dengan cepat dan tepat.

Game merupakan salah satu teknologi yang perkembangannya sangat pesat pada masa kini di mulai dari *game* sederhana seperti permainan kartu, catur sampai permainan yang di mainkan di beberapa sarana atau perangkat teknologi seperti Playstation 3, Xbox 360, Nintendo Wii, PSP, Nintendo DS, maupun PC dari yang berbasis individu ataupun *multiplayer* yang lebih kompleks.

2.4 Object Oriented Analysis and Design (OOAD)

Object Oriented Analysis (OOA) adalah metode analisis yang memeriksa requirement (syarat/keperluan yang harus dipenuhi suatu sistem) dan sudut pandang kelas-kelas dan objek-objek yang ditemui dalam ruang lingkup permasalahan. Yang menjadi sasaran OOA adalah mengembangkan sederatan model yang menggambarkan perangkat lunak komputer pada saat perangkat itu bekerja untuk memenuhi serangkaian persyaratan yang ditentukan oleh pelanggan. Seperti metode analisis konvensional lainnya .

Object Oriented Design (OOD) adalah metode untuk mengarahkan arsitektur software yang didasarkan pada manipulasi objek-objek sistem atau subsistem. OOD mnerjemahkan model OOA dari dunia nyata kedalam model implementasi spesifik yang dapat direalisasikan di dalam perangkat lunak. Proses OOD dapat digambarkan sebagai piramida yang terdiri dari 4 (empat) lapisan. Lapisan dasar berfokus pada design subsistem yang mengimplementasikan fungsi-fungsi sitem mayor, lapisan kelas mengkhususkan keseluruhan arsitektur objek dan hirarki kelas yang diperlukan untuk mengimplementasi suatu sistem, lapisan pesan menunjukan bagaimana kolaborasi antar objek akan direalisasikan, dan lapisan tanggung jawab mengidentifikasikan atribut dan operasi yang menandai masing-masing kelas [10].

2.4.1 Konsep-konsep OOAD

OOAD mencakup analisis dan desain sebuah sistem dengan pendekatan objek, yaiut analisis berorientasi objek (OOA) dan desain berorientasi objek (OOD). OOA adalah metode analisis yang memerika *requirement* (syarat/keperluan) yang harus dipenuhi sebuah sistem) dari sudut pandang kelas-kelas dan objek-objek yang ditemui dalam ruang lingkup perusahaan. Sedangkan OOD adalah metode untuk mengarahkan arsitektur software yang didasarkan pada manipulasi objek-objek sistem atau subsistem. Terdapat beberapa konsep dalam OOAD [10], yaitu :

1. *Object*

   Objek adalah suatu benda, entitas, biasanya berupa kata benda (*noun*), sesuatu yang dapat diangkat maupun ditendang atau pukul. Beberapa objek berupa benda hidup, dan beberapa tidak. Contoh objek dalam kehidupan nyata sehari-hari adalah sebuah mobil, orang, rumah, meja, seekor anjing, kucing ataupun beruang. Objek memiliki atribut (*attributes*), misalnya: sebuah mobil memiliki atribut manufaktur, nomor model, warna, dan harga, seekor anjing memiliki atribut umur, warna, tinggi maupun berat. Objek juga memiliki kelakuan (*behavior*), misalnya: sebuah mobil dapat bergerak dari satu tempat ke tempat lainnya dan seekor anjing dapat menggongong.

2. Enkapsulasi

   Enkapsulasi merujuk pada sebuah objek yang menyembunyikan atau mengenkapsulasi atribut-atributnya terhadap operasi yang dikenakan (analoginya objek tersimpan dan terkunci dalam sebuah kapsul, dan operasi berada di tepian luar kapsul tersebut). Atribut yang tersembunyi ini disebut dengan *private*. Beberapa bahasa pemograman (sebagai contoh, *Smalltalk*) secara otomatis membuat atribut-atributnya sebagai *private* dan beberapa tidak (sebagai contoh, Java) membiarkan programmernya menentukan sendiri.

3. Asosiasi dan Agregasi

   Setiap objek memiliki kaitan dengan objek lainnya, baik itu secara langsung maupun tidak langsung, kuat maupun lemah. Dengan mengaitkan tiap objek,

membuat objek semakin kuat. Kaitan antar objek mengijinkan untuk membantu menemukan informasi dan kelakuan (*behavior*) ekstra. Ketika melakukan pemodelan objek, dapat mengaitkan ke dalam 2 prinsip: Asosiasi atau Agregasi. Terkadang memang sulit untuk mencari letak perbedaan antara asosiasi dan agregasi ini. Berikut sedikit penjelasannya :

a. Asosiasi merupakan kaitan yang lemah, objek dapat berupa grup atau kelompok, namun objek ini tidak secara komplit tergantung satu dengan lainnya. Sebagai contoh, bayangkanlah sebuah mobil, seorang supir, seorang penumpang dan satu penumpang lainnya. Ketika supir dan dua penumpang berada dalam mobil, mereka berasosiasi dan mereka semua menuju ke arah atau alamat yang sama, mereka menempati tempat yang sama, yakni di dalam mobil. Namun kaitan asosiasi ini sifatnya lemah, supir dapat menurunkan salah satu atau kedua penumpang pada arah atau alamat yang terpisah, jadi penumpang tersebut tidak lagi berasosiasi dengan objek lainnya.
b. Agregasi berarti menempatkan objek bersama-sama untuk menjadikannya objek yang lebih besar. Barang-barang atau benda-benda manufaktur biasanya merupakan contoh dari bentuk agregasi. Sebagai contoh, sebuah *microwave* terbentuk dari sebuah *cabinet* (semacam lemari kaca), sebuah pintu, sebuah panel indikator, sejumlah tombol, sebuah alat pemanas dan lain sebagainya. Agregasi mengimplikasikan kedekatan ketergantungan.

4. *Class*

Sebuah kelas mengenkaspsulasi karakteristik bawaan ke dalam sebuah kelompok objek (sekumpulan objek). Beberapa programmer mengatakan “objek merupakan *instance* (diumpamakan) dari sebuah kelas”, oleh karena itu istilah *instance* bisa dikatakan merupakan persamaan kata (*synonym*) dari objek. Kelas merupakan sebuah model, pola, atau *blueprint* (cetak biru) yang digunakan untuk menciptakan sebuah objek. Ambil contoh sebuah rumah yang dibuat oleh seorang arsitek. Ada banyak sekali rumah di dunia ini, kesemuanya memiliki karakteristik yang sama. Sebuah cetak biru

menggambarkan semua karakteristik yang dimiliki oleh sebuah rumah, misalnya dinding, jendela, saluran listrik, dan lain sebagainya. Dengan menggunakan cetak biru tersebut, maka dapat dibuat banyak rumah. Rumah-rumah yang dibangun dari cetak biru itu disebut dengan *instance* dari kelas. *Instance* dari kelas adalah objek yang dibentuk dengan menggunakan kelas sebagai dasar pembentukannya. Rumah-rumah tersebut yang terbentuk dari cetak biru yang ada dapat berbeda satu dengan lainnya, misalnya: lokasi, bahan yang digunakan untuk membangun rumah, bentuk dan ukuran, dan lain sebagainya. Tetepi mereka dapat dikatakan memiliki karakteristik yang sama.

5. Pewarisan *(Inheritance)*

   Pewarisan mengijinkan untuk menspesifikasikan atau mengkhususkan bahwa sebuah kelas bisa memiliki atau mendapatkan beberapa karakteristik dari kelas induknya (*parent class*) dan menambahkan fitur atau kegunaan khusus atas dirinya sendiri. Kelas anak (*subclass*) mewariskan (*inherits*) semua *fields, messages, dan methods* dari kelas induk utama (*superclass*). Pewarisan menjadikan kelas dapat di-grupkan atau dikelompokkan ke dalam kelas yang lebih general (umum), oleh sebab itu pewarisan sering disebut juga sebagai bentuk spesialisasi atau generalisasi atas objek. Dalam perspektif pemograman, pewarisan diperlukan karena:

   a. Pewarisan mendukung perluasan dan kekuatan yang lebih dalam proses pemodelan perangkat lunak.
   b. Pewarisan mengijinkan untuk mendefenisikan informasi atau atribut dan kelakuan (*behavior*) pada sebuah kelas dan membagikannya (*share*) kepada kelas anak (*subclass*) yang berhubungan, sehingga *code* yang dimiliki akan lebih efisien dan *reuseable*.
   c. Pewarisan merupakan hal yang natural, yang gampang dimengerti dan didefenisikan dalam kehidupan nyata

6. Polimorfisme *(Polymorphism)*

   Polimorfisme diambil dari bahasa Yunani yang diturunkan dari kata *poly*, yang berarti banyak, dan *morph* yang berarti bentuk. Maka dari itu polimorfisme berarti ‘memiliki banyak bentuk/rupa’. Dalam kehidupan

sehari-hari, antara satu objek dengan objek lainnya sering terjadi komunikasi. Satu objek mengirmkan pesan ke objek lainnya untuk mendapatkan sesuatu yang dinginkan, namun tanggapan dari objek-objek tersebut dapat beragam walaupun pesan yang disampaikan adalah pesan yang sama.

2.4.2 OOP (*Object* Oriented Programming)

OOP (*Object Oriented Programming*) atau yang dikenal dengan Pemrograman Berorientasi Objek merupakan paradigma pemrograman yang berorientasikan kepada objek. Semua data dan fungsi di dalam paradigma ini dibungkus ke dalam kelas-kelas atau objek-objek.

Model data berorientasi objek dikatakan dapat memberi fleksibilitas yang lebih, kemudahan mengubah program dan digunakan luas dalam teknik piranti lunak skala besar. Lebih jauh lagi, pendukung OOP mengklaim bahwa OOP lebih mudah dipelajari bagi pemula dibanding dengan pendekatan sebelumnya, dan pendekatan OOP lebih mudah dikembangkan dan dirawat.

Dengan menggunakan OOP maka dalam melakukan pemecahan suatu masalah tidak harus melihat bagaimana cara menyelesaikan suatu masalah tersebut (terstruktur) tetapi objek-objek apa yang dapat melakukan pemecahan masalah tersebut. Sebagai contoh suatu departemen yang memiliki *manager*, sekretaris, petugas administrasi data dan lainnya. Misal manager tersebut ingin memperoleh data dari bagian administrasi maka *manager* tersebut tidak harus mengambilnya langsung tetapi dapat menyuruh petugas bagian administrasi untuk mengambilnya. Pada kasus tersebut seorang *manager* tidak harus mengetahui bagaimana cara mengambil data tersebut tetapi manager bisa mendapatkan data tersebut melalui objek petugas administrasi. Jadi untuk menyelesaikan suatu masalah dengan kolaborasi antar objek-objek yang ada karena setiap objek memiliki deskripsi tugasnya sendiri **[11]**.

2.4.2.1 Istilah-istilah 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. Contoh Objek : Volcano Robot (*a volcanic exploration vehicle*), mempunyai atribut sebagai berikut :

   Status → exploring, moving, returning home

   Speed → in miles per hour

   Temperature → in Fahrenheit degrees

   Dalam *class*, atribut sering disebut sebagai variabel. Atribut dibedakan menjadi dua jenis yaitu *Instance Variable* dan *Class Variable*. *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. *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.

   a. *Behavior*

Behavior/tingkah laku adalah hal-hal yang bisa dilakukan oleh objek dari suatu *class*. *Behavior* dapat digunakan untuk mengubah nilai atribut suatu objek, menerima informasi dari objek lain, dan mengirim informasi ke objek lain untuk melakukan suatu tugas (*task*).

b. Abstraksi

Abstraksi adalah kemampuan sebuah program untuk melewati aspek informasi yang diproses olehnya, yaitu kemampuan untuk memfokus pada inti. Setiap objek dalam sistem melayani sebagai model dari "pelaku" abstrak yang dapat melakukan kerja, laporan dan perubahan keadaannya dan berkomunikasi dengan objek lainnya dalam sistem, tanpa mengungkapkan bagaimana kelebihan ini diterapkan. Proses, fungsi atau metode dapat juga dibuat abstrak dan beberapa teknik digunakan untuk mengembangkan sebuah pengabstrakan.

2.4.2.2 Konsep-konsep OOP

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:

a. *Subclass* menyediakan *state/behaviour* yang spesifik yang membedakan dengan *superclass*, sehingga memungkinkan *programmer* untuk menggunakan ulang *source code* dari *superclass* yang telah ada.
b. *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

Poolimorfisme 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.

2.5 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 Modelling Language* (UML). Adapun pengertian dari UML adalah salah satu standar bahasa yang banyak digunakan di dunia industri untuk mendefinisikan *requirement*, membuat analisis dan desain, serta menggambarkan arsitektur dalam pemrograman berorientasi objek [12].

UML muncul karena adanya kebutuhan pemodelan visual untuk menspesifikasikan, menggambarkan, membangun, dan dokumentasi dari sistem perangkat lunak. Dalam hal ini UML merupakan suatu bahasa visual untuk

melakukan pemodelan dan komunikasi mengenai sebuah sistem dengan menggunakan diagram dan teks-teks pendukung.

UML menggunakan berbagai macam diagram dengan fungsi masing-masing untuk menggambarkan setiap proses dari sistem berorientasi objek. UML memiliki 13 jenis diagram. Dalam pembangunan *game* ini dipakai 4 diagram UML diantaranya :

1. *Use Case Diagram*

Use Case atau diagram *use case* merupakan pemodelan yang digunakan untuk menggambarkan kelakuan (*behavior*) dari sistem yang akan dibuat. *Use case* mendeskripsikan sebuah interaksi antara satu atau lebih aktor dengan sistem yang akan dibuat. Secara kasar, *use case* digunakan untuk mengetahui fungsi apa saja yang ada di dalam sebuah sistem dan siapa saja yang berhak menggunakan fungsi-fungsi tersebut. Syarat penamaan pada *use case* adalah nama didefinisikan sesimpel mungkin dan dapat dipahami. Ada dua hal utama pada *use case* yaitu pendefinisian apa yang disebut aktor dan *use case* [12].

a. Aktor merupakan orang, proses, atau sistem lain yang berinteraksi dengan sistem yang akan dibuat diluar sistem yang akan dibuat itu sendiri, jadi walaupun simbol dari aktor adalah gambar orang, tapi aktor belum tentu merupakan orang.
b. *Use case* merupakan fungsionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesan antarunit atau aktor.

Sebuah *use case* dapat meninclude fungsionalitas *use case* lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa *use case* yang di-*include* akan dipanggil setiap kali *use case* yang me-*include* dieksekusi secara normal. *Use case* dapat di-*include* oleh lebih dari satu *use case* lain, sehingga duplikasi fungsionalitas dapat dihindari dengan cara menarik keluar fungsionalitas yang *common*. *Use case* juga dapat me-e*xtend use case* lain dengan *behaviour*-nya sendiri. Sementara hubungan generalisasi antar *use case* menunjukkan bahwa *use case* yang satu merupakan spesialisasi dari yang lain. Contoh dari *use case diagram* dapat dilihat pada gambar berikut.

2. *Activity Diagram*

Activity diagram menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi.

Activity diagram merupakan *state diagram* khusus, di mana sebagian besar *state* adalah *action* dan sebagian besar transisi di-*trigger* oleh selesainya *state* sebelumnya (*internal processing*). Oleh karena itu *activity diagram* tidak menggambarkan *behaviour* internal sebuah sistem (dan interaksi antar subsistem) secara eksak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum.

Sebuah aktivitas dapat direalisasikan oleh satu *use case* atau lebih. Aktivitas menggambarkan proses yang berjalan, sementara *use case* menggambarkan bagaimana aktor menggunakan sistem untuk melakukan aktivitas. Sama seperti *state*, standar UML menggunakan segiempat dengan sudut membulat untuk menggambarkan aktivitas. *Decision* digunakan untuk menggambarkan *behaviour* pada kondisi tertentu. Untuk mengilustrasikan proses-proses paralel (*fork* dan *join*) digunakan titik sinkronisasi yang dapat berupa titik, garis horizontal atau vertical [12].

3. *Sequence Diagram*

Diagram *sequence* adalah diagram yang menggambarkan kelakuan objek pada *use case* dengan mendeskripsikan waktu hidup objek dan *message* yang dikirimkan dan diterima antar objek [12]. Oleh karena itu untuk menggambarkan diagram *sequence* maka harus diketahui objek-objek yang terlibat dalam sebuah *use case* beserta metode-metode yang dimiliki kelas yang diinstansiasi menjadi objek itu.

Banyaknya diagram *sequence* yang harus digambarkan adalah sebanyak pendefinisian *use case* yang memiliki proses itu sendiri atau yang penting semua *use case* yang telah didefinisikan interaksi jalannya pesan sudah dicakup pada diagram *sequence* sehingga semakin banyak *use case* yang didefinisikan maka diagram *sequence* yang harus dibuat juga semakin banyak.

4. *Class Diagram*

Diagram kelas atau *class diagram* menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang akan dibuat untuk membangun sistem. Kelas memiliki apa yang disebut atribut dan metode atau operasi.

a. Atribut merupakan variabel-variabel yang dimiliki oleh suatu kelas
b. Operasi atau metode adalah fungsi-fungsi yang dimiliki oleh suatu kelas

2.6 Tools Yang Digunakan Untuk Pembuatan *Game*

Tools yang digunakan dalam penelitian *Game* Penerapan Metode N-gram Untuk Non player character Prediksi Formasi agar Menggunakan Formasi Yang Tepat Pada Game Futsal diantaranya adalah *ActionScript* 3.0 dan Adobe Flash.

2.6.1 ActionScript

ActionScript adalah bahasa pemrograman yang dibuat berdasarkan *ECMAScript,* yang digunakan dalam pengembangan situs web dan perangkat lunak menggunakan *platform* Adobe Flash Player. *ActionScript* juga dipakai pada beberapa aplikasi basis data, seperti *Alpha Five*. Bahasa ini awalnya dikembangkan oleh Macromedia, tapi kini sudah dimiliki dan dilanjutkan perkembangannya oleh Adobe, yang membeli Macromedia pada tahun 2005.

ActionScript terbaru saat ini adalah *ActionScript* 3.0 dari edisi yang sebelumnya dikenal dengan *ActionScript* 2.0. *ActionScript* 3.0 adalah bahasa berorientasi objek untuk membuat aplikasi dan konten multimedia ber-*script* untuk pemutaran di *runtime* Flash *client* (seperti Flash Player dan Adobe AIR). Dengan sintaks yang mirip dengan Java dan C#. *ActionScript* 3.0 memiliki beberapa kelebihan dibanding pendahulunya, antara lain fitur yang ditawarkan adalah file pada *ActionScript* 3.0 dapat dibuat terpisah saat *runtime* [13].

2.6.1 Adobe Flash

Adobe Flash merupakan sebuah program yang didesain khusus oleh Adobe dan program aplikasi standar authoring tool professional yang digunakan untuk membuat animasi dan bitmap yang sangat menarik untuk keperluan pembangunan situs web yang interaktif dan dinamis. Flash didesain dengan kemampuan untuk

membuat animasi 2 dimensi yang handal dan ringan sehingga flash banyak digunakan untuk membangun dan memberikan efek animasi pada *website*, CD Interaktif dan yang lainnya. Selain itu aplikasi ini juga dapat digunakan untuk membuat animasi logo, *movie*, *game*, pembuatan navigasi pada situs web, tombol animasi, banner, menu interaktif, interaktif form isian, *e-card*, *screen saver* dan pembuatan aplikasi-aplikasi web lainnya.

Dalam Flash, terdapat teknik-teknik membuat animasi, fasilitas *action script, filter, custom easing* dan dapat memasukkan video lengkap dengan fasilitas *playback* FLV. Keunggulan yang dimiliki oleh Flash ini adalah ia mampu diberikan sedikit *code* pemograman baik yang berjalan sendiri untuk mengatur animasi yang ada didalamnya atau digunakan untuk berkomunikasi dengan program lain seperti HTML, PHP, dan Database dengan pendekatan XML, dapat dikolaborasikan dengan web, karena mempunyai keunggulan antara lain kecil dalam ukuran *file output*-nya.

Flash bekerja dengan bahasa pemprograman yang dinamakan *ActionScript*. Bahasa ini telah mengalami beberapa kali perkembangan sejak pertama kali diperkenalkan. Versi terakhir adalah *ActionScript* 3.0 atau disingkat AS3, dengan struktur bahasa yang sangat menyerupai *javascript*.

Berkas yang dihasilkan dari perangkat lunak ini mempunyai *file extension* .swF. SWF adalah kependekan dari "*Shock Wave Flash*", kata lain yang biasanya memiliki ekstensi .swf dipergunakan untuk menyebutkan file swf adalah *Flash Movie*. *File Flash Movie* dan dapat diputar di penjelajah web yang telah dipasangi Adobe Flash Player. *File* swf juga didapatkan pada *file game* yang dibuat oleh flash. Dalam Adobe Flash terbaru terdapat fungsi *export* yang dapat meng-*export file* kedalam bentuk .apk eksensi yang biasa digunakan dalam android

Flash juga dapat digunakan untuk mengembangkan secara cepat aplikasi-aplikasi web yang kaya dengan pembuatan *script* tingkat lanjut. Di dalam aplikasinya juga tersedia sebuah alat untuk men-*debug script*, dengan menggunakan *code hint* untuk mempermudah dan mempercepat pembuatan dan pengembangan isi *ActionScript* secara otomatis [14].

2.7 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 *Whitebox* dan Pengujian *Blackbox* [15].

2.7.1 Pengujian *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

2.7.2 Pengujian *Blackbox*

Pengujian yang dilakukan untuk antarmuka perangkat lunak, pengujian ini dilakukan untuk memperlihatkan bahwa fungsi-fungsi bekerja dengan baik dalam arti masukan yang diterima dengan benar dan keluaran yang dihasilkan benar-benar tepat, pengintegrasian dari eksternal data berjalan dengan baik.

Metode pengujian *blackbox* 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.

Pengujian *blackbox* merupakan pendekatan komplementer dari teknik white box, karena pengujian *blackbox* diharapkan mampu mengungkap kelas kesalahan yang lebih luas dibandingkan teknik *whitebox*. Pengujian *blackbox* berfokus pada pengujian persyaratan fungsional perangkat lunak, untuk mendapatkan serangkaian kondisi input yang sesuai dengan persyaratan fungsional suatu program.

Pengujian *blackbox* adalah pengujian aspek fundamental sistem tanpa memperhatikan struktur logika internal perangkat lunak. Metode ini digunakan untuk mengetahui apakah perangkat lunak berfungsi dengan benar. Pengujian *blackbox* merupakan metode perancangan data uji yang didasarkan pada spesifikasi perangkat lunak. Data uji dibangkitkan, dieksekusi pada perangkat lunak dan kemudian keluaran dari perangkat lunak dicek apakah telah sesuai dengan yang diharapkan berikut adalah ciri-ciri *blackbox testing*.

1. Black box testing berfokus pada kebutuhan fungsional pada software, berdasarkan pada spesifikasi kebutuhan dari software.
2. Black box testing bukan teknik alternatif daripada white box testing. Lebih daripada itu, ia merupakan pendekatan pelengkap dalam mencakup error dengan kelas yang berbeda dari metode white box testing.
3. Black box testing melakukan pengujian tanpa pengetahuan detil struktur internal dari sistem atau komponen yang dites. juga disebut sebagai behavioral testing, specification-based testing, input/output testing atau functional testing

Pada *blackbox testing* terdapat jenis teknik disain tes yang dapat dipilih berdasarkan pada tipe testing yang akan digunakan, yang diantaranya :

1. Equivalence Class Partitioning
2. Boundary Value Analysis
3. State Transitions Testing
4. Cause-Effect Graphing