BAB II

LANDASAN TEORI

2.1. Konsep Dasar Kecerdasan Buatan

Kecerdasan buatan atau lebih dikenal sebagai *Artificial Intelligence*, memiliki beberapa defenisi, antara lain :

a. *Artificial intelligence* adalah ilmu yang mengembangkan komputer supaya dapat bekerja dan berpikir serta mengambil keputusan seperti layaknya manusia.

b. *Artificial intelligence* merupakan salah satu bagian ilmu komputer yang membuat agar mesin (komputer) dapat melakukan pekerjaan seperti dan sebaik yang dilakukan oleh manusia.

c. *Artificial intelligence* merupakan software yang memungkinkan komputer digital bisa meniru beberapa fungsi otak manusia yang terbatas.

Layaknya Manusia yang memiliki otak, komputer juga dapat memiliki perangkat lunak yang bekerja sebagai otak. Manusia dapat menyelesaikan berbagai masalah bukan hanya karena manusia memiliki otak yang mampu menalar dan menganalisa, tapi manusia juga memiliki basis data, pengetahuan, kumpulan informasi, yang semuanya itu diperoleh dari pengalaman, dan belajar. Semuanya itu kemudian diproses secara otomatis dan sering tidak disadari oleh manusia itu sendiri. Beberapa lingkup utama dalam kecerdasan buatan adalah :

a. Sistem Pakar (*Expert System*)

Sistem Pakar adalah suatu perangkat lunak yang menyimpan pengetahuan seorang pakar, dengan demikian komputer tersebut akan memiliki keahlian layaknya seorang pakar.

b. Pengolahan Bahasa Alami (*Natural Language Processing*)

Dengan Pengolahan Bahasa Alami ini, diharapkan dikemudian hari manusia dapat berkomunikasi dengan komputer dengan menggunakan bahasa sehari-hari.

c. Pengenalan Ucapan (*Speech Recognition*)

Dengan teknologi ini, diharapkan manusia nantinya dapat berbincang-bincang dengan komputer.

d. *Computer Vision*

Teknologi ini merupakan upaya pengenalan citra atau objek visual pada komputer. Dengan mengunakan alat sensor atau *scanner* sebagai indra, maka komputer dapat mengenali objek apa yang ditangkap oleh indranya.

e. *Game Playing*

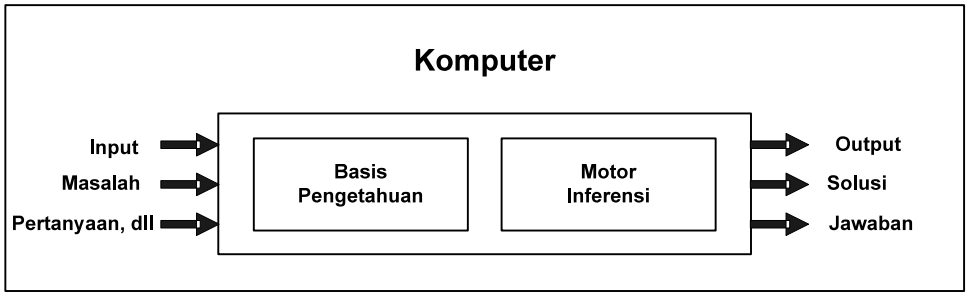
Mengembangkan berbagai macam bentuk permainan interaktif yang cerdas. Diasumsikan dengan teknologi tersebut dapat menarik minat para penggemar games.

2.1.1. Komponen Kecerdasan Buatan

Untuk dapat membangun aplikasi kecerdasan buatan ada 2 bagian utama yang sangat dibutuhkan, yaitu:

a. **Basis Pengetahuan (*Knowledge Base*),** berisi fakta-fakta, teori, pikiran dan hubungan antara satu dengan lainnya

b. **Motor Inferensi (*Inferensi Engine*)**, yaitu kemampuan menarik kesimpulan berdasarkan pengalaman



Gambar 2.1. Penerapan Konsep Kecerdasan Buatan di Komputer

2.1.2. Keuntungan Kecerdasan Buatan

Jika dibandingkan dengan kecerdasan alami (kecerdasan yang dimiliki oleh manusia), kecerdasan buatan memiliki beberapa keuntungan secara komersial antara lain :

a. Kecerdasan buatan lebih bersifat permanen. Kecerdasan alami akan cepat mengalami perubahan. Hal ini dimungkinkan karena sifat manuasia yang pelupa. Kecerdasan buatan tidak akan berubah sepanjang sistem komputer dan program tidak mengubahnya.

b. Kecerdasan buatan lebih mudah diduplikasi dan disebarkan. Mentransfer pengetahuan manusia dari satu orang ke orang lain membutuhkan proses yang sangat lama, dan juga suatu keahlian itu tidak akan pernah dapat diduplikasi dengan lengkap. Oleh karena itu, jika pengetahuan terletak pada suatu sistem komputer, pengetahuan

tersebut dapat disalin dari komputer tersebut dan dapat dipindahkan dengan mudah ke komputer lain.

2.2. Konsep Dasar Algoritma

Dalam bahasa Indonesia, kata *algorithm* diserap menjadi **algoritma**. Mulanya kata *algorism* berasal dari nama penulis buku arab yang terkenal, yaitu **Abu Ja'far Muhammad ibnu Musa Al-Khuwarizmi**. *Al-Khuwarizmi* oleh orang barat dibaca menjadi al-gorism. Al-Khuwarizmi adalah penulis buku yang berjudul **Kitab Al-Jabar Wal-Muqabala** (*The Book of Restoration and Reduction*). Pada akhirnya algorism berubah menjadi algoritm, sehingga akhiran -*sm* berubah menjadi –*thm*.

Defenisi Algoritma, "*Algoritma adalah urutan langkah-langkah logis penyelesaian masalah yang disusun secara sistematis"*. Sedangkan menurut Kamus Besar Bahasa Indonesia, terbitan Balai Pustaka, 1988. "*Algoritma adalah urutan logis pengambilan putusan untuk pemecahan masalah"*.

Kata **logis** merupakan kata kunci. Langkah-langkah yang disusun tersebut harus logis, artinya nilai kebenarannya harus dapat ditentukan, benar atau salah. Berikut ini adalah contoh menuliskan algoritma TUKAR ISI BEJANA;

Algoritma **TUKAR_ISI_BEJANA**

Diberikan dua buah bejana, A dan B; bejana A berisi larutan berwarna merah, bejana B berisi larutan berwarna biru. Pertukarkan isi kedua bejana itu sedemikian sehingga bejana A berisi larutan berwarna biru dan bejana B berisi larutan merah.

Deskripsi

1. Tuangkan larutan dari bejana A ke dalam bejana C
2. Tuangkan larutan dari bejana B ke dalam bejana A
3. Tuangkan larutan dari bejana C ke dalam bejana B

2.3. Algoritma Pencarian *Artificial Intelligence*

Pencarian dan pelacakan merupakan salah satu hal penting yang menentukan keberhasilan dan kesuksesan sebuah kecerdasan buatan. Pada dasarnya terdapat 2 teknik pencarian dan pelacakan yang digunakan dalam kecerdasan buatan, yaitu Pencarian Buta (*Blind Search*) dan Pencarian Heuristik (*Heuristic Search*).

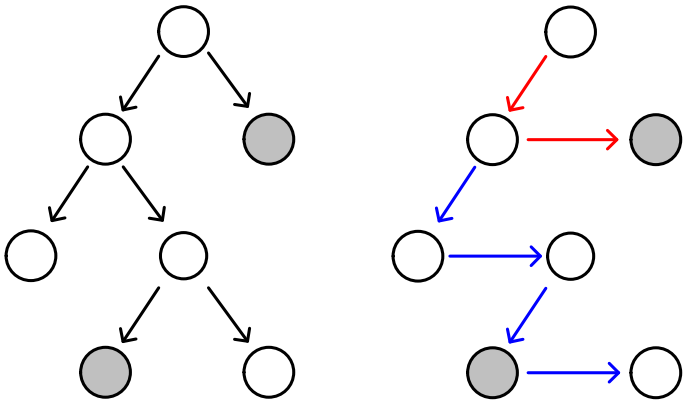
2.3.1. Konsep Dasar Algoritma *Depth-First Search*

Algoritma pencarian *blind search* merupakan algoritma pencarian yang secara umum dibagi menjadi 2 yaitu :

1. Pencarian Melebar Pertama (*Breadth-First Search*)

   Pada metode *Breadth-First Search*, semua node pada level n akan dikunjungi terlebih dahulu sebelum mengunjungi node-node pada level n+1. Untuk lebih jelasnya dapat dilihat pada gambar 2.2. Dimulai dari *root* atau akar kemudian dilanjutkan pada node level n paling kiri ke arah node paling kanan. Kemudian dilanjutkan kembali pada node level n+1 paling kiri kearah node paling kanan, demikian seterusnya hingga ditemukan solusi yang diinginkan. Tanda panah berwarna merah merupakan alur pencarian yang dilakukan pada tahap

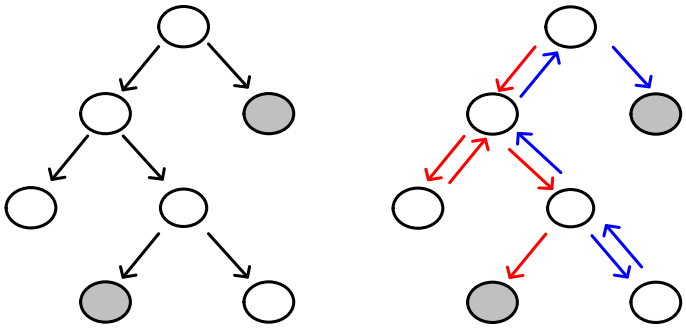
pertama, kemudian dilanjutkan dengan alur yang ditandai dengan tanda panah berwarna biru.



Gambar 2.2. Algoritma Breadth -First Search

2. Pencarian Mendalam Pertama (*Depth-First Search*)

   Pada metode *Depth-First Search*, pencarian dilakukan mulai dari root atau akar ke node dengan level n paling kiri, dilanjutkan ke semua anaknya sebelum dilanjutkan pencarian ke node-node selevel. Untuk lebih jelasnya dapat dilihat pada gambar 2.3.



Gambar 2.3. Algoritma Depth-First Search

Gambar 2.3. telah menampilkan dengan jelas bagaimana teknik pencarian pada *depth-first search*. Tanda panah berwarna merah merupakan alur pencarian

yang dilakukan pada tahap pertama, kemudian dilanjutkan dengan alur yang ditandai dengan tanda panah berwarna biru.

Secara derkriptif, algoritma *depth-first search* dapat disajikan dalam bentuk sebagai berikut :

Algoritma *depth-first search* :

1. Jika keadaan awal merupakan tujuan, keluar (sukses).
2. Jika tidak demikian, kerjakan langkah-langkah berikut ini sampai tercapai keadaan sukses atau gagal.
   a. Bangkitkan *succesor* E dari keadaan awal. Jika tidak ada *succesor*, maka akan terjadi kegagalan.
   b. Panggil *depth-first search* dengan E sebagai keadaan awal.
   c. Jika sukses berikan tanda sukses. Namun jika tidak, ulangi langkah-2.

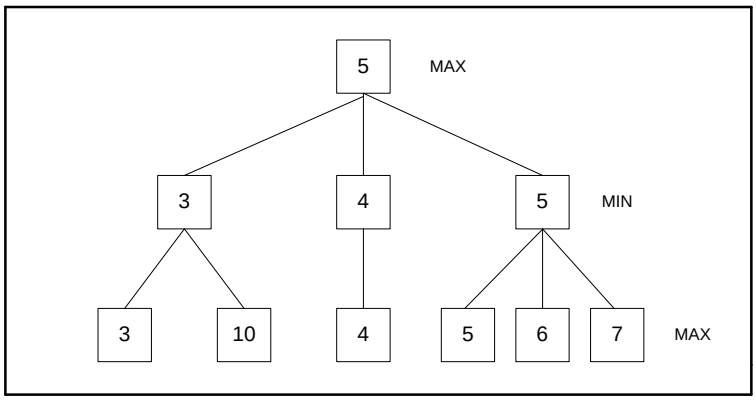
2.3.2. Konsep Dasar Algoritma MINIMAX

Ada banyak bidang dalam penerapan kecerdasan buatan, namun Permainan atau *Game Playing* adalah salah satu yang paling digemari. Ada banyak jenis aplikasi yang dibangun berdasarkan kecerdasan buatan, dan games (permainan) merupakan aplikasi yang paling menarik perhatian masa atau publik. Saat ini banyak vendor-vendor sistem operasi yang mendistribusikan OS atau *Operating System*-nya bersamaan dengan games. Maka bukan hal yang mengejutkan apabila telah banyak algoritma-algoritma kecerdasan buatan yang telah dipikirkan bersamaan games tersebut.

Algoritma *minimax* diterapkan pada permainan dengan dua orang pemain, seperti *Tic-Tac-Toe, Chess, Go,* dan lain-lain. Persamaan atara semua permainan

tersebut yaitu semua merupakan permainan logika. Ini berarti bahwa permainan merupakan sekumpulan aturan main, dan dasar pemikiran yang logis. Adanya aturan main dan dasar pemikiran yang logis tersebut, maka nantinya dapat diketahui, apa langkah selanjutnya yang akan dilakukan. Bahkan sebagian pemain mengetahui semua langkah yang mungkin akan diambil oleh lawan.

Minimax adalah teknik pelacakan mendalam. Ide dasar dari *minimax* adalah untuk memulai pada posisi yang terakhir dan menggunakan pembangkit langkah yang mungkin untuk menghasilkan sekumpulan posisi yang paling baik. kita dapat mempergunakan fungsi penaksiran statis pada kumpulan posisi tersebut dan memilih yang terbaik dengan sederhana. Setelah melakukannya, kita dapat mengembalikan nilai tersebut ke posisi awal untuk menggambarkan evaluasi dari hasil tersebut. Fungsi dari penaksiran statis adalah mengembalikan nilai yang besar untuk mengindikasikan situasi yang baik, jadi tujuannya adalah untuk memaksimalkan nilai dari fungsi penaksiran statis untuk posisi selanjutnya. Prosedur *minimax* merupakan prosedur rekursi yang didasarkan pada dua prosedur pembantu.



Gambar 2.4. Contoh Pohon Pencarian

Minimax merupakan salah satu teknik permainan yang terkenal. Minimax menggunakan teknik pencarian *Depth-First Search* dengan kedalaman terbatas, dan fungsi evaluasi yang digunakan adalah fungsi evaluasi statis, dengan mengansumsikan bahwa lawan akan membuat langkah terbaiknya, yang dapat dilakukan. Pada Minimax ada 2 prosedur yang dijalankan, yaitu ; **Maksimasi** yang dilakukan oleh "Pemain Pertama", dan **Minimasi** yang dilakukan oleh "Pemain Kedua". Dan dalam hal ini diperlukan suatu fungsi evaluasi statis yang menyatakan nilai yang mungkin didapat oleh pemain. Berikut ini adalah tahapan-tahapan dari algoritma minimax :

1. Mencari langkah-langkah terbaik yang mungkin akan diambil oleh lawan, sebagai prioritas.
2. Menentukan range nilai yang akan menentukan nilai yang diberikan pada tiap-tiap langkah.
3. Pada tiap-tiap langkah diberikan nilai secara acak sesuai range yang telah ditentukan.
4. Pemain pertama selalu melakukan maksimasi, dan pemain kedua melakukan minimasi.

5. Langkah terbaik dicari atau dipilih sesuai nilai dari masing-masing langkah. Bila prosedur yang sedang dijalankan adalah maksimasi, maka langkah terbaik yang dipilih adalah langkah yang memiliki nilai langkah tertinggi. Begitu juga sebaliknya, bila prosedur yang sedang dijalankan adalah minimasi, maka langkah terbaik yang dipilih adalah angkah yang memiliki nilai langkah terendah.
6. Melakukan evaluasi kondisi jumlah tanda pada tiap-tiap papan atau kotak, baik secara horisontal, vertikal atau diagonal.
7. Status pemenang bernilai +1 untuk kemenangan pemain yang melakukan prosedur maksimasi, status pemenang bernilai -1 untuk kemenangan pemain yang melakukan prosedur minimasi, dan status pemenang bernilai 0 untuk kondisi seri.

Berikut ini adalah *Pseudocode* dari algoritma minimax :

```
Minimax (StatusSaya,kedalaman,pemain)
IF (kedalaman= =max)
      RETURN static (StatusSaya,pemain)
Bangkitkan successor S(1..n)
IF(pemain= =saya)
      RETURN max of Minimax (S(i),kedalaman+1,lawan)
ELSE
      RETURN max of Minimax (S(i),kedalaman+1,Saya)
```

2.4. Pemograman Berorientasi Objek (PBO)

OOP (*Object Oriented Programming*) atau disebut juga dengan PBO (Pemograman Berorientasi Objek) merupakan suatu kelas bahasa pemrograman

dan teknik yang didasarkan pada suatu objek, dimana struktur data terenkapsulasi dengan rutin yang mengoperasikan data tersebut.

Keunggulan utama pemrograman berorientasi objek adalah kemampuannya untuk mengembangkan objek-objek sekali saja dan menggunakannya lagi (*reusable*) berulang-ulang.

Terdapat Karakteristik-karakteristik dalam Pemograman Berorientasi Objek yaitu :

a. Pembungkusan (*Encapsulation*)

Dalam sistem berorientasi objek kita menggabungkan potongan-potongan informasi dan perilaku-perilaku spesifik yang bekerja pada informasi tersebut, kemudian mengemasnya menjadi apa yang disebut sebagai objek. Hal ini dirujuk dari kata pembungkusan (*encapsulation*). Cara lain umtuk memahami pembungkusan adalah kita membagi aplikasi-aplikasi menjadi bagian-bagian kecil yang secara fungsional berhubungan.

b. Pewarisan (*Inheritance*)

Pewarisan adalah konsep dasar pemograman berorientasi objek yang lainnya. Pada system berorientasi objek, pewarisan adalah mekanisme yang memungkinkan kita menciptakan objek-objek baru berdasarkan objek lain yang sudah ada: Objek *Anak* mewarisi segala sesuatunya dari objek *Induk*.

Salah satu keuntungan dari pewarisan (*inheritance*) adalah kemudahannya untuk dipelihara. Saat sesuatu perubahan untuk informasi atau perilaku yang sama perlu terjadi pada seluruh objek.

c. *Polimorfisme*

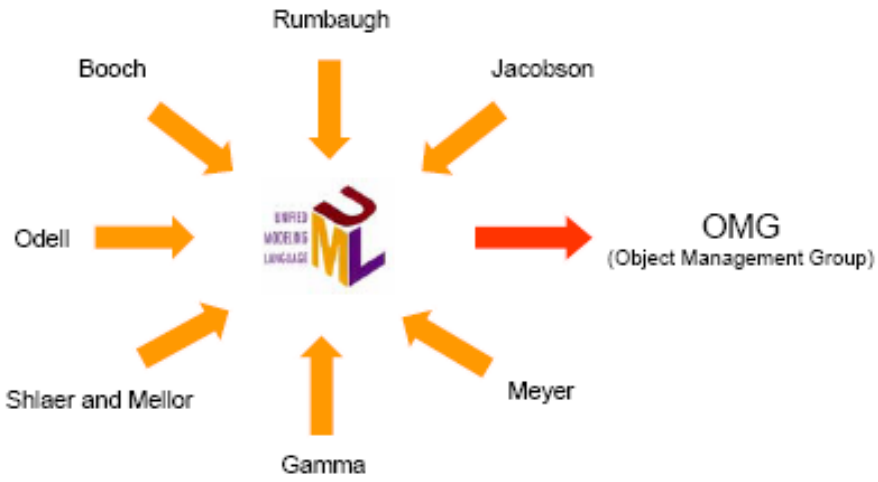
Polimorfisme adalah prinsip penting yang ketiga dari pemograman berorientasi objek. *Polimorfisme* berarti suatu fungsionalitas yang diimplementasikan dengan berbagai cara yang berbeda. Pada terminologi berorientasi objek, ini berarti kita dapat memiliki berbagai implementasi untuk sebagian fungsionalitas tertentu. Salah satu keuntungan dari *polimorfisme* adalah kemudahan pemeliharaannya.

2.5. UML (*Unified Modeling Language*)

*Unified Modelling Langua*ge (UML) adalah sebuah "bahasa" yg telah menjadi standar dalam industri untuk visualisasi, merancang dan mendokumentasikan sistem piranti lunak. UML menawarkan sebuah standar untuk merancang model sebuah sistem.

2.5.1. Sejarah UML

Sejarah UML sendiri cukup panjang. Sampai era tahun 1990 seperti kita ketahui puluhan metodologi pemodelan berorientasi objek telah bermunculan di dunia. Diantaranya adalah: metodologi booch, metodologi coad, metodologi OOSE, metodologi OMT, metodologi shlaer-mellor, metodologi wirfs-brock, dsb. Masa itu terkenal dengan masa perang metodologi (*method war*) dalam pendesainan berorientasi objek. Masing-masing metodologi membawa notasi sendiri-sendiri, yang mengakibatkan timbul masalah baru apabila kita bekerjasama dengan group/perusahaan lain yang menggunakan metodologi yang berlainan.



Gambar 2.5. Sejarah UML

Dimulai pada bulan Oktober 1994 Booch, Rumbaugh dan Jacobson, yang merupakan tiga tokoh yang boleh dikata metodologinya banyak digunakan mempelopori usaha untuk penyatuan metodologi pendesainan berorientasi objek. Pada tahun 1995 direlease draft pertama dari UML (versi 0.8). Sejak tahun 1996 pengembangan tersebut dikoordinasikan oleh *Object Management Group* (OMG – http://www.omg.org). Tahun 1997 UML versi 1.1 muncul, dan saat ini versi terbaru adalah versi 1.5 yang dirilis bulan Maret 2003. Booch, Rumbaugh dan Jacobson menyusun tiga buku serial tentang UML pada tahun 1999. Sejak saat itulah UML telah menjelma menjadi standar bahasa pemodelan untuk aplikasi berorientasi objek.

2.5.2. Tujuan Penggunaan UML

Memodelkan suatu sistem (bukan hanya perangkat lunak) yang menggunakan konsep berorientasi *object*. Dan menciptakan suatu bahasa pemodelan yang dapat digunakan baik oleh manusia maupun mesin.

2.5.3. Gambaran dari UML

UML dapat dimanfaatkan untuk berbagai macam kebutuhan, diantaranya untuk melakukan pemodelan, visualisasi, spesifikasi, konstruksi dan dokumentasi.

2.5.4. Notasi dalam UML

UML menyediakan beberapa notasi dan artifak standar yang bisa digunakan sebagai alat komunikasi bagi para pelaku dalam proses analisis dan desain. Artifak didalam UML didefinisikan sebagai informasi dalam berbagai bentuk yang digunakan atau dihasilkan dalam proses pengembangan perangkat lunak. Contohnya adalah *sourcecode* yang dihasilkan oleh proses pemograman.

Yang perlu diperhatikan untuk menjaga konsistensi antar artifak selama proses analisis dan desain adalah bahwa setiap perubahan yang terjadi pada satu artifak harus dilakukan pada artifak sebelumnya. Misalnya ditemukan satu cara yang lebih efisien waktu membuat *Sequence* diagram, maka perbaikan tersebut perlu diverifikasikan terhadap *Use Case* diagram, apakah perlu dilakukan perubahan juga pada *Use Case* spesifikasinya.

a. Actor

Actor adalah segala sesuatu yang berinteraksi dengan sistem aplikasi komputer. Jadi actor ini bisa berupa orang, perangkat keras, atau mungkin objek lain dalam sistem yang sama. Biasanya yang dilakukan oleh *actor*

adalah memberikan informasi pada sistem atau memerintahkan sistem untuk melakukan sesuatu.

b. Class

Class merupakan pembentuk utama dari sistem berorientasi objek karena class menunjukkan kumpulan objek yang memiliki atribut dan operasi yang sama. *Class* digunakan untuk mengimplementasikan *interface. Class* digunakan untuk mengabstraksikan elemen-elemen dari sistem yang sedang dibangun. *Class* bisa untuk merepresentasikan baik perangkat lunak maupun perangkat keras, baik konsep maupun benda nyata.

Notasi class berbentuk persegi panjang berisi tiga bagian yaitu :

1) Persegi panjang atas untuk nama class;
2) Persegi panjang bawah untuk operasi;
3) Persegi panjang tengah untuk atribut.

Atribut digunakan untuk menyimpan informasi. Nama atribut menggunakan kata benda yang bisa dengan jelas merepresentasikan informasiyang disimpan didalamnya. Operasi menunjukkan sesuatu yang bisa dilakukan oleh objek, dan menggunakan kata kerja.

c. Interface

Interface merupakan kumpulan operasi tanpa implementasi dari suatu *class*. Implementasi operasi dalam interface dijabarkan oleh operasi dalam *class*. Oleh karena itu, keberadaan interface selalu disertai oleh *class* yang mengimplementasikan operasinya. Interface ini merupakan salah satu cara mewujudkan prinsip enkapsulasi dalam objek.

d. Use Case

Use case menjelaskan urutan kegiatan yang dilakukan *actor* dan sistem untuk mencapai suatu tujuan tertentu. Walaupun menjelaskan kegiatan namun use case hanya menjelaskan apa yang dilakukan oleh *actor* dan sistem, bukan bagaimana *actor* dan sistem melakukan kegiatan tersebut.

e. Interaction

Interaction digunakan untuk menunjukkan aliran pesan atau informasi antar objek maupun hubungan antar objek. Biasanya *interaction* ini dilengkapi juga dengan teks bernama operation signature yang tersusun dari nama operasi, parameter yang dikirim dan tipe parameter yang dikembalikan.

f. Package

Package adalah kontainer atau wadah konseptual yang digunakan untuk mengelompokkan elemen-elemen dari sistem yang sedang dibangun, sehingga bisa dibuat model yang lebih sederhana. Tujuannya adalah untuk mempermudah penglihatan (*visibility*) dari model yang sudah dibangun.

g. Note

Note digunakan untuk memberikan keterangan dan komentar tambahan dari suatu elemen sehingga bisa langsung terlampir dalam model. *Note* ini bisa ditempatkan kesemua elemen notasi yang lain.

h. Dependency

Dependency merupakan relasi yang menunjukkan bahwa perubahan pada salah satu elemen memberi pengaruh pada elemen lain. Elemen yang ada

di bagian tanda panah adalah elemen yang tergantung pada elemen yang ada dibagian tanda panah.

Terdapat dua *stereotype* dari dependency, yaitu *Include* dan *Extend*. *Include* menunjukkan bahwa suatu bagian dari elemen (yang ada digaris tanpa panah) memicu eksekusi bagian dari elemen lain (yang ada digaris dengan panah), misalnya untuk notasi A - - - > B operasi yang ada di class A memicu di eksekusinya operasi yang ada di class B. Extended menunjukkan bahwa suatu bagian elemen digaris tanpa panah bisa disisipkan kedalam elemen yang ada digaris dengan panah, misalnya untuk notasi A - - - > B suatu fungsi dari use case A bisa disisipkan kedalam use case B atau dengan kata lain A optional untuk B.

Kedua *stereotype* ini direpresentasikan dengan menambahkan *text include* atau *extends* di notasi *dependency*.

i. Boundary Class

Boundary Class adalah *class* yang menghubungkan *user* dengan sistem. Oleh karena itu sering disebut juga sebagai *user interface class*.

j. Control Class

Control Class adalah *class* yang mengkoordinasi aktivitas dalam sistem. *Class* ini menghubungkan *boundary class* dengan *entity class*.

k. Entity Class

Entity class adalah *class* yang berhubungan dengan data atau informasi yang digunakan oleh sistem. *Entity class* ini adalah *class* yang menyimpan dan mengelola data.

2.5.5. Diagram Grafis

Diagram berbentuk grafik yang menunjukkan simbol elemen model yang disusun untuk mengilustrasikan bagian atau aspek tertentu dari sistem. Sebuah diagram merupakan bagian dari suatu *view* tertentu dan ketika digambarkan biasanya dialokasikan untuk v*iew* tertentu. Adapun jenis diagram antara lain :

a. Use Case Diagram

Menggambarkan sejumlah *external actors* dan hubungannya ke *use case* yang diberikan oleh sistem. Use case adalah deskripsi fungsi yang disediakan oleh sistem dalam bentuk teks sebagai dokumentasi dari use case symbol namun dapat juga dilakukan dalam *activity diagrams*. *Use case* digambarkan hanya yang dilihat dari luar oleh *actor* (keadaan lingkungan sistem yang dilihat user) dan bukan bagaimana fungsi yang ada di dalam sistem.

b. Class Diagram

Menggambarkan struktur statis *class* di dalam sistem. *Class* merepresentasikan sesuatu yang ditangani oleh sistem.

Class dapat berhubungan dengan yang lain melalui berbagai cara: *associated* (terhubung satu sama lain), *dependent* (satu class tergantung/menggunakan class yang lain), *specialed* (satu class merupakan spesialisasi dari class lainnya), atau *package* (grup bersama sebagai satu unit). Sebuah sistem biasanya mempunyai beberapa *class diagram*.

c. State Diagram

Menggambarkan semua *state* (kondisi) yang dimiliki oleh suatu *object* dari suatu *class* dan keadaan yang menyebabkan *state* berubah. Kejadian dapat berupa *object* lain yang mengirim pesan. *State class* tidak digambarkan untuk semua *class*, hanya yang mempunyai sejumlah *state* yang terdefinisi dengan baik dan kondisi *class* berubah oleh *state* yang berbeda.

d. Activity Diagram

Menggambarkan rangkaian aliran dari aktivitas, digunakan untuk mendeskripsikan aktifitas yang dibentuk dalam suatu operasi sehingga dapat juga digunakan untuk aktifitas lainnya seperti *use case* atau interaksi.

e. Sequence Diagram

Menggambarkan kolaborasi dinamis antara sejumlah *object*. Kegunaanya untuk menunjukkan rangkaian pesan yang dikirim antara *object* juga interaksi antara *object*, sesuatu yang terjadi pada titik tertentu dalam eksekusi sistem.

f. Collaboration Diagram

Menggambarkan kolaborasi dinamis seperti *sequence diagrams*. Dalam menunjukkan pertukaran pesan, *collaboration diagrams* menggambarkan *object* dan hubungannya (mengacu ke konteks). Jika penekannya pada waktu atau urutan gunakan *sequence diagrams*, tapi jika penekanannya pada konteks gunakan *collaboration diagram*.

g. Component Diagram

Menggambarkan struktur fisik kode dari komponent. Komponent dapat berupa *source code*, komponent *biner*, atau *executable component*. Sebuah komponent berisi informasi tentang *logic class* atau *class* yang diimplementasikan sehingga membuat pemetaan dari *logical view* ke *component view*.

h. Deployment Diagram

Menggambarkan arsitektur fisik dari perangkat keras dan perangkat lunak sistem, menunjukkan hubungan komputer dengan perangkat (*nodes*) satu sama lain dan jenis hubungannya. Di dalam *nodes*, *executeable component* dan *object* yang dialokasikan untuk memperlihatkan unit perangkat lunak yang dieksekusi oleh node tertentu dan ketergantungan komponen.

2.6. Sejarah *Tic-Tac-Toe*

Bicara tentang *game*, banyak hal yang menarik darinya. Perkembangan *game* komputer dan *video game* yang kian memanjakan para pemainnya dengan teknologi-teknologi tinggi merupakan poin menarik untuk dicermati. Meski sejarah *video game* dan *game* komputer mencakup rentang waktu sekira lima dekade, keduanya meraih popularitas sebagai bagian dari peradaban manusia modern di akhir tahun 1970.

Tic-Tac-Toe adalah sebuah permainan masa lampau sejak 1300 BC di Mesir. Yang juga dikenal sebagai “Nol-nol dan Silang-silang”, permainan *Tic-Tac-Toe* adalah salah satu *game* legendaris yang pernah ada, bahkan orang Jawa

mengklaim dan menyebutnya catur jawa. Kita bisa memainkan *Tic-Tac-Toe* melawan seorang teman dengan papan cukup dari kertas yang telah digaris kotak-kotak bujur sangkar seperti buku tulis matematika. Siapa mampu membuat garis vertikal, horizontal, atau diagonal yang terbentuk dari lima tanda '0' atau 'X', dialah pemenangnya.

Awal perkembangan Tahun 1952, A.S. Douglas membuat OXO, *game* grafis *noughts and crosses* (nol dan silang), di University of Cambridge untuk mendemonstrasikan tesisnya tentang interaksi komputer dan manusia. Permainan ini bekerja pada komputer besar yang menggunakan *CRT display*. Bahkan, perangkat *game portable* genggam yang pertama dibuat adalah *Tic-Tac-Toe* di tahun 1972 oleh Waco Company. Sampai sekarang *game* ini masih populer di internet.

Meski mempertimbangkan suatu hiburan yang sederhana, itu mengusulkan satu masalah yang menarik untuk dipertimbangkan mengenai satu strategi optimal. Sepanjang suatu pertandingan dirmainkan secara relatif sedikit gerakan-gerakan, secara optimal dalam kaitan dengan menggunakan istilah Tic-Tac-Toe mengambil sebuah arti lebih bertahan yang dibandingkan dengan permainan-permainan yang lain. Yakni, suatu strategi optimal hanya dijamin untuk menghindari kerugian pemain, sedangkan itu tidak bisa menjamin suatu kemenangan melawan setiap lawan. Sebaliknya, semakin kecil nomor dari situasi-situasi permainan terpisah jelas juga menyatakan bahwa strategi seperti itu bisa berpotensi ditemukan dan disandikan dengan metode pencarian komputasi.

Sementara di dalam skala yang lebih kecil, Strategi *Tic-Tac-Toe* masih menyajikan satu ruang pencarian yang besar, dan meski mereka Membuat kemungkinan situasi-situasi game yang dapat dikurangi, itu sisa pada ukuran tidak praktis untuk satu eksplorasi yang menyeluruh. Tantangan kemudian untuk memikirkan satu metode pencarian lokal yang efektif untuk masalah ini, dan berdasar pada catatan sukses mereka yang diberi banyak permasalahan yang serupa, algoritma-algoritma genetik terpilih untuk percobaan ini.

Sebagai bagian dari keluarga yang lebih besar dari metode pencarian yang evolusiner, algoritma-algoritma genetik kombinasikan masalah spesifik menyandi dan metoda-metoda serupa dengan seleksi alami untuk meningkatkan suatu solusi. Pada umumnya, suatu implementasi algoritma seperti itu memelihara suatu kolam dari individu (populasi), masing-masing yang mana membawa satu pengkodean bahwa menerjemahkan ke dalam beberapa jawaban memberi masalah pencarian, seperti ketentuan khusus yang lengkap semua parameter untuk membangun kebanyakan melaksanakan dan antena satelit sedikit mahal. Semakin baik jawaban bahwa itu membawa, yang lebih tinggi kebugaran satu individu miliki. Dengan masing-masing mencari langkah, suatu populasi yang baru diciptakan dari yang tua di mana individu terbaik adalah yang terpilih untuk menukar spesifikasi mereka dan reproduksi. Dari waktu ke waktu lalu, fitur menguntungkan diharapkan untuk muncul dari populasi kolam dan untuk berpotensi menghasilkan perorangan dengan satu jawaban optimal. Pembaca itu disebut Koza (1992) untuk suatu pengenalan yang saksama kepada algoritma-algoritma genetik seperti yang berikut diskusi mengasumsikan keakraban dengan domain.

Haystack Genetik: Eksplorasi Ruang Pencarian

Suatu algoritma genetik (GA) yang paling efektif di dalam situasi-situasi, di mana suatu masalah yang dirumuskan dengan baik menawarkan suatu pengkodean padat dari semua parameter-parameter solusi yang perlu. Jika pengkodean ini bertumbuh terlalu besar dan kompleks, algoritma menghadapi pembatasan-pembatasan dari yang lain metode pencarian lokal dan tidak bisa diharapkan untuk menemukan suatu optimum global. Di dalam mempertimbangkan masalah strategi *Tic-Tac-Toe*, itu adalah yang pertama penting bagi menemukan suatu matriks S yang pantas dan untuk memastikan bahwa suatu GA dapat secara efektif diterapkan.

Permainan-permainan *Tic-Tac-Toe* dan Situasi-situasi

Pada. umumnya, suatu strategi *Tic-Tac-Toe* adalah paling dengan mudah diwakili sebagai suatu pemetaan untuk semua kemungkinan situasi-situasi permainan pada suatu gerak yang sesuai bahwa harus diambil seperti ke (sedikitnya) hindari suatu kerugian. Ukuran dari ruang strategi kemudian yang digambarkan oleh banyaknya semua kemungkinan situasi-situasi permainan yang mengikuti dari pertanyaan tentang berapa banyak pertandingan-pertandingan yang terpisah dapat dimainkan.

Sementara mungkin kelihatannya tidak produktif untuk memainkan *Tic-Tac-Toe* secara terus-menerus, banyak waktu dapat sesungguhnya digunakan dengan hitungan *Tic-Tac-Toe*.

2.7. Aturan Permainan *Tic-Tac-Toe*

1. Besar papan permainan 3x3 atau lebih dengan deret kemenangan 3 atau disesuaikan dengan papan permainan.
2. Pemain dalam permainan ini dibatasi untuk satu lawan satu, dilakukan bergiliran antara pemain pertama dan pemain kedua
3. Untuk memberikan langkah, pemain harus mengisi kotak dengan simbol masing-masing, biasanya X atau O.
4. Kotak yang sudah terisi tidak bisa ditimpa oleh langkah berikutnya.
5. Langkah yang sudah diambil tidak dapat dibatalkan atau diganti dengan langkah yang lain.
6. Tujuan dari permainan ini adalah untuk mendapatkan deret dengan tiga simbol atau lebih yang sama secara horizontal, vertikal atau diagonal.
7. Pemenang ditentukan oleh siapa yang pertama kali menyusun deret tersebut.

2.7.1. Strategi yang Biasa Dilakukan

1. Langkah pertama adalah mengambil posisi tengah bila belum diambil oleh lawan jika posisi tengah sudah diambil maka pilih diantara empat pojok.
2. Memblok lawan ketika akan membuat deret, dan membuat deret jika memungkinkan

2.7.2. Struktur Data

Sembilan elemen vektor yang merepresentasikan papan permainan, dimana setiap elemen dari vektor dicocokkan pada posisi papan seperti berikut:

1	**2**	**3**
4	**5**	**6**
7	**8**	**9**

Gambar 2.6. Contoh Papan Permainan *Tic-Tac-Toe*

Setiap elemen mengandung nilai 2(menunjukan kosong), 3(menunjukan X), 5 (menunjukan O).

2.7.3. Strategi

Strategi yang dilakukan adalah dengan membuat tiga prosedur.

berikan nilai 5 jika kotak tengah kosong, itu berarti if kotak[5]=2. sebaliknya, fungsi ini mengembalikan pada kotak selain pojok (2, 4, 6, atau 8).

berikan nilai 0 jika pemain p tidak memenangkan permainan pada langkah berikutnya; sebaliknya, mengembalikan nomor kotak yang merupakan langkah kemenangan. Fungsi ini akan memungkinkan program untuk menang dan menghalangi lawan menang. Posswin dioperasikan dengan pengecekan, sekali dalam sewaktu, masing-masing tiap baris, tiap kolom, dan diagonal. Karena tiap nilai mempunyai nomor, maka dapat dilakukan pengecekan keseluruh baris, kolom atau diagonal untuk melihat apakah mungkin menang dengan mengalikan nilai pada kotak.

Jika hasilnya 18(3x3x2), maka X dapat menang. Jika hasil 50(5x5x2), maka O dapat menang. Jika kita menemukan baris kemenangan, kita memutuskan elemen mana yang kosong, dan mengembalikan nilai kotak tersebut.

buat langkah pada kotak n. prosedur ini meletakan papan[n} ke 3 jika giliran pemain, atau 5 jika giliran komputer.

Dari strategi diatas kita dapat melihat bahwa komputer harus melakukan pengecekan pada beberapa kondisi sebelum membuat setiap langkah. Strategi diatas mudah untuk dipahami. Tapi kesalahan pada kemampuan pemrogram permainan *Tic-Tac-Toe* akan kelihatan jika program dimainkan. Dan kita masih belum dapat menyamaratakan setiap pengetahuan program dengan program yang lainnya, seperti program yang lebih kompleks.