BAB 2 TINJAUAN PUSTAKA

2.1 Kecerdasan Buatan

2.1.1 Definisi Kecerdasan Buatan

Ada beberapa definisi (*Artificial Intelligence*) AI atau kecerdasan buatan, antara lain [4]:

- **H. A. Simon (1987):**

  "Kecerdasan buatan merupakan kawasan penelitian, aplikasi dan instruksi yang terkait dengan pemrograman komputer untuk melakukan sesuatu hal yang -dalam pandangan manusia adalah- cerdas".

- **Rich and Knight (1991):**

  "Kecerdasan Buatan merupakan sebuah studi tentang bagaimana membuat komputer melakukan hal-hal yang pada saat ini dapat dilakukan lebih baik oleh manusia."

- **Suparman (1991):**

  "*Artificial Intelligence* merupakan sub-bidang pengetahuan komputer yang khusus ditujukan untuk membuat software dan hardware yang sepenuhnya bisa menirukan beberapa fungsi otak manusia".

- **Jogiyanto H. M (2000):**

  "Kecerdasan buatan didefinisikan sebagai suatu mesin atau alat pintar (biasanya adalah suatu komputer) yang dapat melakukan suatu tugas yang bilamana tugas tersebut dilakukan oleh manusia akan dibutuhkan suatu kepintaran untuk melakukannya".

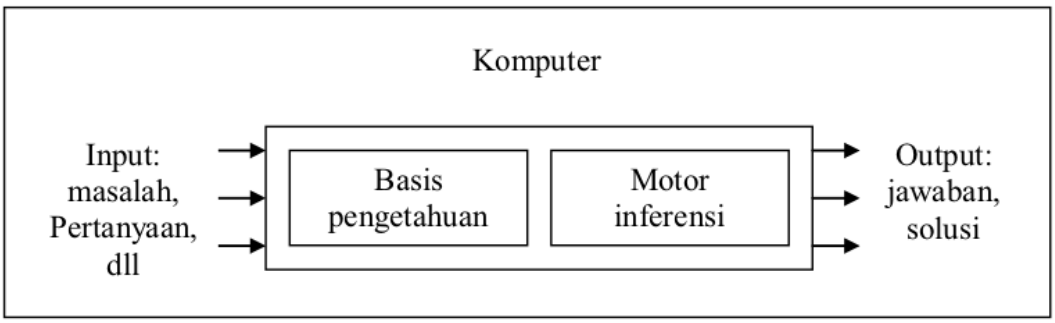
- **Sri Kusumadewi (2003):**

  “Kecerdasan buatan merupakan salah satu bagian ilmu komputer yang membuat agar mesin (komputer) dapat melakukan pekerjaan seperti dan sebaik yang dilakukan oleh manusia“.

- **Anita Desiani & Muhammad Arhami (2006):**

  ”Kecerdasan buatan merupakan cabang dari ilmu komputer yang konsern dengan pengautomatisasi tingkah laku cerdas ”.

Dari beberapa pengertian di atas, dapat ditarik kesimpulan bahwa kecerdasan buatan merupakan bagian dari ilmu komputer yang menitikberatkan pada perancangan otomatisasi tingkah laku cerdas. Beberapa macam bidang yang menggunakan kecerdasan buatan antara lain sistem pakar, permainan komputer (*games*), *logika fuzzy*, jaringan syaraf tiruan dan robotika.



Gambar 2.1. Penerapan konsep kecerdasan buatan di komputer [3]

Tujuan dari kecerdasan buatan menurut Winston dan Prendergast (1984):

1. Membuat mesin menjadi lebih pintar (tujuan utama)
2. Memahami apa itu kecerdasan (tujuan ilmiah)
3. Membuat mesin lebih bermanfaat (tujuan entrepreneurial)

AI dapat dipandang dalam berbagai perspektif:

1. Dari perspektif kecerdasan, Kecerdasan buatan akan membuat mesin menjadi ‘cerdas’ (mampu berbuat seperti apa yang dilakukan oleh manusia).

2. Dari perspektif pemrograman, AI meliputi studi tentang pemrograman simbolik, pemecahan masalah *(problem solving),* dan pencarian (*searching*).

3. Dari perspektif bisnis, AI adalah kumpulan peralatan *(tools)* yang sangat *powerful* dan metodologis dalam menyelesaikan masalah-masalah bisnis.

4. Dari perspektif penelitian, Kecerdasan buatan adalah suatu studi bagaimana membuat agar komputer dapat melakukan sesuatu sebaik yang dikerjakan oleh manusia.

Domain penelitian dalam kecerdasan buatan yaitu:

a) Mundane task

- Persepsi *(vision & speech)*
- Bahasa alami (*Understanding, generation & translation*)
- Pemikiran yang bersifat *commonsense*.
- Robot *control*.

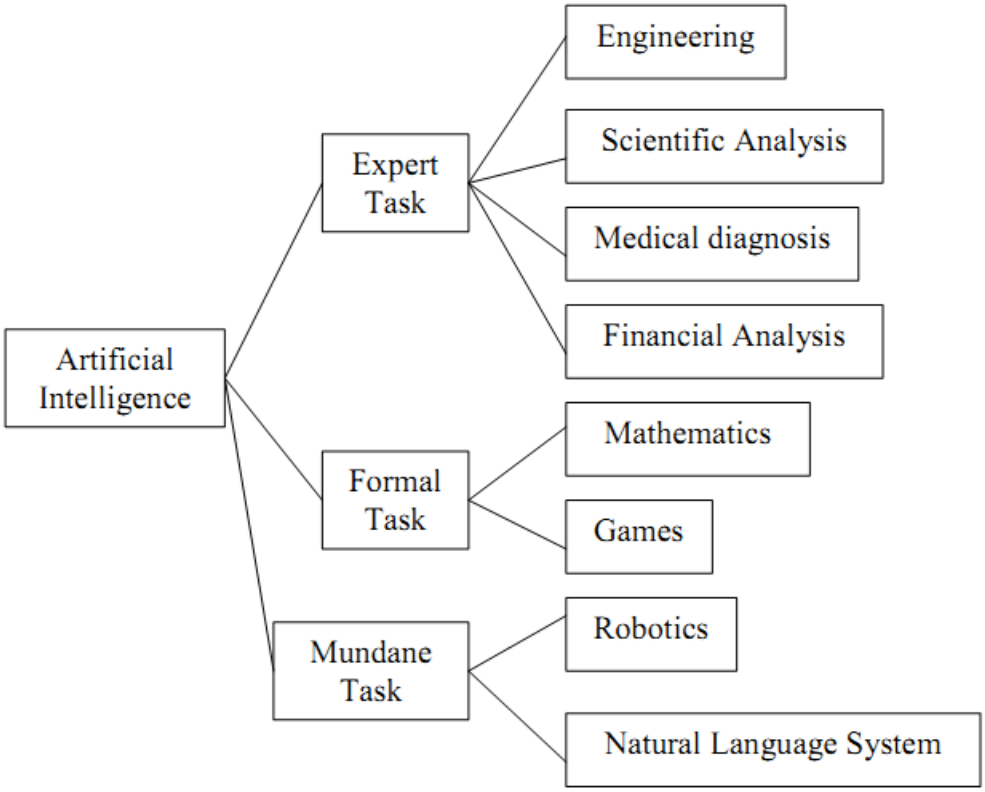
b) Formal task

- Permainan / *games.*
- Matematika (geometri, logika, kalkulus integral, pembuktian).

c) Expert task

- Analisis finansial.
- Analisis medikal.
- Analisis ilmu pengetahuan.

- Rekayasa (desain, pencarian kegagalan, perencanaan manufaktur).



Gambar 2.2. Bidang-bidang tugas (*task domains*) dari AI [2]

2.2 Teknik-Teknik Dasar Pencarian

Pencarian merupakan salah satu teknik untuk menyelesaikan permasalahan AI. Keberhasilan suatu sistem salah satunya ditentukan oleh kesuksesan dalam pencarian dan pencocokan. Teknik dasar pencarian memberikan suatu kunci bagi banyak sejarah penyelesaian yang penting dalam bidang AI. Ada beberapa aplikasi yang menggunakan teknik pencarian ini, yaitu [3]:

1. Papan game dan puzzle (tic-tac-toe, catur, menara hanoi).
2. Penjadwalan dan masalah routing (*travelling salesman problem*).
3. Parsing bahasa dan inteprestasinya (pencarian struktur dan arti).

4. Logika pemrograman (pencarian fakta dan implikasinya).
5. Computer vision dan pengenalan pola.
6. Sistem pakar bebasis kaidah (*rule based expert system*).

Pencarian adalah proses mencari solusi dari suatu permasalahan melalui sekumpulan kemungkinan ruang keadaan (*state space*). Ruang keadaan merupakan suatu ruang yang berisi semua keadaan yang mungkin. Kondisi suatu pencarian meliputi [1]:

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

Solusi merupakan suatu lintasan dari keadaan awal sampai keadaan tujuan. Secara umum, proses pencarian dapat dilakukan seperti berikut [6]:

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

2.2.1 Masalah Pencarian

Masalah pencarian merupakan proses pencarian solusi yang direncanakan, yang mencari lintasan dari keadaan sekarang sampai keadaan tujuan. Suatu masalah pencarian direpresentasikan sebagai graf. Keadaan direpresentasikan

sebagai simpul (*node*), sedangkan langkah yang dibolehkan atau aksi direpresentasikan dengan busur *(arc).* Dengan demikian, secara khusus masalah pencarian didefinisikan sebagai [1]:

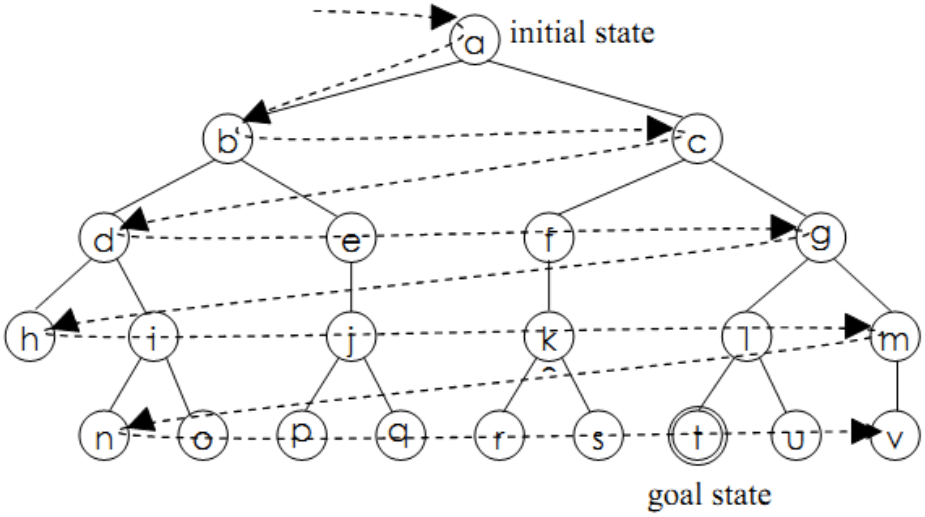
1. State space (ruang keadaan).
2. Start node (permukaan simpul).
3. Kondisi tujuan dan uji untuk mengecek apakah kondisi tujuan ditemukan atau tidak.
4. Kaidah yang memberikan bagaimana mengubah keadaan.

2.2.2 Strategi Pencarian Mendalam

Pencarian boleh jadi merupakan hasil dari suatu solusi ruang keadaan yang mungkin telah terkunjungi semua, tetapi tanpa penyelesaian. Pencarian yang mendalam (*Exchausting Search Strategy*) mungkin dilakukan dengan menggunakan strategi *Breadth First Search* atau *Depth First Search* (*Iterative Deepening*). Kedua pencarian ini merupakan pencarian buta (*blind search*).

2.2.2.1 *Breadth First Search*

Prosedur *Breadth First Search* merupakan pencarian yang dilakukan dengan mengamati setiap node di setiap tingkat graf sebelum bergerak menuju ruang yang lebih dalam. Atau dengan kata lain, penulusuran yang dilakukan adalah dengan mengunjungi tiap-tiap node pada level yang sama hingga ditemukan *goal state*-nya [1]. Untuk lebih jelasnya, perhatikan ilustrasi dari *Breadth First Search* pada gambar 2.3.



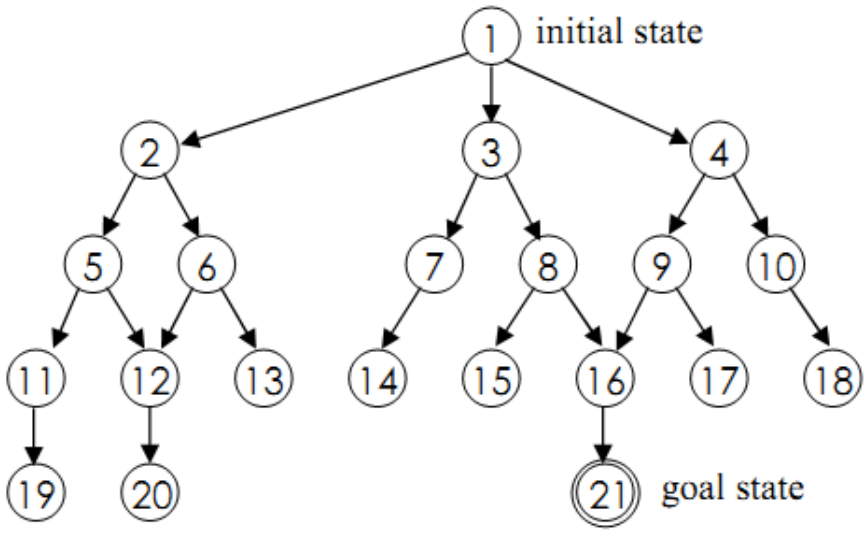
Gambar 2.3. Teknik pencarian *Breadth First Search*

Pengimplementasian *Breadth First Search* dapat ditelusuri dengan menggunakan daftar (*list*), *open* dan *closed*, untuk menelusuri gerakan pencarian di dalam ruang keadaan. Prosedur untuk *Breadth First Search* dapat dituliskan sebagai berikut:

```
1  procedure BFS (G,v): [7]
2      create a queue Q
3      enqueue v onto Q
4      mark v
5      while Q is not empty:
6          t ← Q.dequeue()
7          if t is what we are looking for:
8              return t
9          for all edges e in G.incidentEdges(t) do
10             o ← G.opposite(t,e)
11             if o is not marked:
12                 mark o
13                 enqueue o onto Q
```

Gambar 2.4. Algoritma BFS

Untuk lebih jelasnya, simak contoh graf pada gambar 2.5.



Gambar 2.5. Contoh graf yang akan ditelusuri

Pada gambar 2.5, state 21 merupakan tujuannya (goal) sehingga bila ditelusuri menggunakan prosedur *Breadth First Search*, diperoleh:

1. Open = [1]; closed = [].
2. Open = [2, 3, 4]; closed = [1].
3. Open = [3, 4, 5, 6]; closd = [2, 1].
4. Open = [4, 5, 6, 7, 8]; closed = [3, 2, 1].
5. Open = [5, 6, 7, 8, 9, 10]; closed = [4, 3, 2, 1].
6. Open = [6, 7, 8, 9, 10, 11, 12]; closed = [5, 4, 3, 2, 1].
7. Open = [7, 8, 9, 10, 11, 12, 13] (karena 12 telah di-open); closed = [6, 5, 4, 3, 2, 1].
8. Open = [8, 9, 10, 11, 12, 13, 14]; closed = [7, 6, 5, 4, 3, 2, 1].
9. Dan seterusnya sampai state 21 diperoleh atau open = [].

Ada beberapa keuntungan menggunakan algoritma *Breadth First Search*

ini, di antaranya adalah tidak akan menemui jalan buntu dan jika ada satu solusi maka *Breadth First Search* akan menemukannya, dan jika ada lebih dari satu solusi maka solusi optimal akan ditemukan.

Namun ada tiga persoalan utama berkenaan dengan *Breadth First Search* ini yaitu:

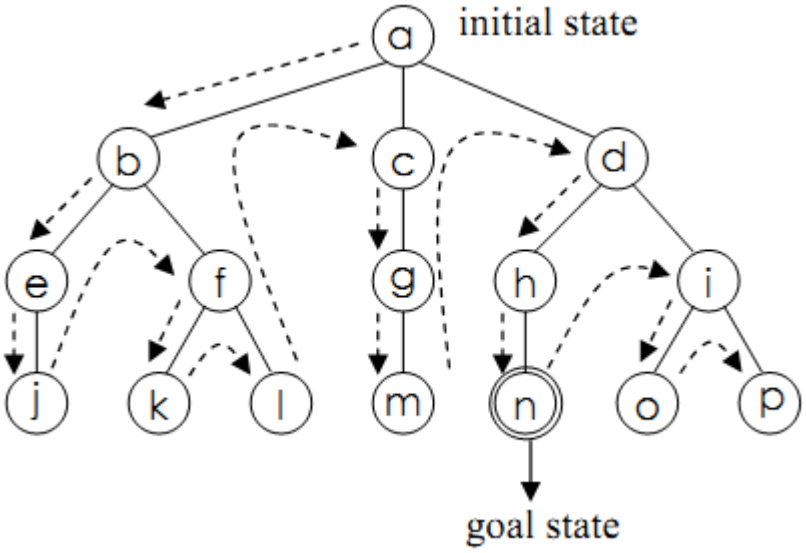
1. Membutuhkan memori yang lebih besar, karena menyimpan semua node dalam satu pohon.
2. Membutuhkan sejumlah besar pekerjaan, khususnya jika lintasan solusi terpendek cukup panjang, karena jumlah node yang perlu diperiksa bertambah secara eksponensial terhadap panjang lintasan.
3. Tidak relevannya operator akan menambah jumlah node yang harus diperiksa.

Oleh karena proses *Breadth First Search* mengamati *node* di setiap level graf sebelum bergerak menuju ruang yang lebih dalam maka mula-mula semua keadaan akan dicapai lewat lintasan yang terpendek dari keadaan awal. Oleh sebab itu, proses ini menjamin ditemukannya lintasan terpendek dari keadaan awal ke keadaan tujuan (akhir). Lebih jauh karena mula-mula semua keadaan ditemukan melalui lintasan terpendek sehingga setiap keadaan yang ditemui pada kali kedua didapati pada sepanjang sebuah lintasan yang sama atau lebih panjang. Kemudian, jika tidak ada kesempatan ditemukannya keadaan yang identik pada sepanjang lintasan yang lebih baik maka algoritma akan menghapusnya [2].

2.2.2.2 *Depth First Search*

Pencarian dengan metode ini dilakukan dari node awal secara mendalam

hingga yang paling akhir (*dead-end*) atau sampai ditemukan. Dengan kata lain, simpul cabang atau anak yang terlebih dahulu dikunjungi. Sebagai ilustrasinya dapat dilihat gambar 2.6.



Gambar 2.6. Teknik pencarian *Depth First Search*

Berdasarkan gambar 2.6, proses pencarian dilakukan dengan mengunjungi cabang terlebih dahulu hingga tiba di simpul terakhir. Jika tujuan yang diinginkan belum tercapai maka pencarian dilanjutkan ke cabang sebelumnya, turun ke bawah jika memang masih ada cabangnya. Begitu seterusnya hingga diperoleh tujuan akhir (*goal*).

Depth First Search, seperti halnya *Breadth First Search*, juga memiliki kelebihan di antaranya adalah cepat mencapai kedalaman ruang pencarian. Jika diketahui bahwa lintasan solusi permasalahan akan panjang maka *Depth First Search* tidak akan memboroskan waktu untuk melakukan sejumlah besar keadaan dangkal dalam permasalahan graf. *Depth First Search* jauh lebih efisien untuk ruang pencarian dengan banyak cabang karena tidak perlu mengeksekusi semua

simpul pada suatu level tertentu pada daftar open. Selain itu, *Depth First Search* memerlukan memori yang relatif kecil karena banyak node pada lintasan yang aktif saja yang disimpan [1].

Selain kelebihan, *Depth First Search* juga memiliki kelemahan di antaranya adalah memungkinkan tidak ditemukannya tujuan yang diharapkan dan hanya akan mendapatkan satu solusi pada setiap pencarian.

Prosedur *Depth First Search* dapat diimplementasikan dengan melakukan modifikasi proses *Breadth First Search* menjadi:

```
1  procedure DFS(G,v): [8]
2     label v as explored
3     for all edges e in G.incidentEdges(v) do
4        if edge e is unexplored then
5           w ← G.opposite(v,e)
6           if vertex w is unexplored then
7              label e as a discovery edge
8              recursively call DFS(G,w)
9        else
10          label e as a back edge
```

Gambar 2.7 Algoritma DFS

Untuk lebih jelasnya, dapat dilihat aplikasi algoritma tersebut untuk gambar 2.5 dengan state 21 diasumsikan sebagai tujuannya. Langkah-langkah penelusuran tersebut adalah:

1. Open = [1]; closed = [].
2. Open = [2, 3, 4]; closed = [1].
3. Open = [5, 6, 3, 4]; closed = [2, 1].
4. Open = [11, 12, 6, 3, 4]; closed = [5, 2, 1].

5. Open = [19, 12, 6, 3, 4]; closed = [11, 5, 2, 1].
6. Open = [12, 6, 3, 4]; closed = [19, 11, 5, 2, 1].
7. Open = [20, 6, 3, 4]; closed = [12, 19, 11, 5, 2, 1].
8. Open = [6, 3, 4]; closed = [20, 12, 19, 11, 5, 2, 1].
9. Open = [13, 3, 4] (karena 12 telah di-closed); closed = [6, 20, 12, 19, 11, 5, 2, 1].
10. Open = [3, 4]; closed = [13, 6, 20, 12, 19, 11, 5, 2, 1].
11. Open = [7, 8, 4]; closed = [3, 13, 6, 20, 12, 19, 11, 5, 2, 1].
12. Dan seterusnya sampai state 21 diperoleh atau open = [].

Penentuan teknik pencarian yang sesuai atau yang tepat untuk sebuah kasus khusus penganalisaan suatu ruang permasalahan, menjadi sangat penting dan biasanya dengan melakukan konsultasi dengan para pakar di bidangnya untuk mendapatkan dan mengetahui tingkah laku ruang permasalahan tersebut [2].

2.3 Pengertian *Game*

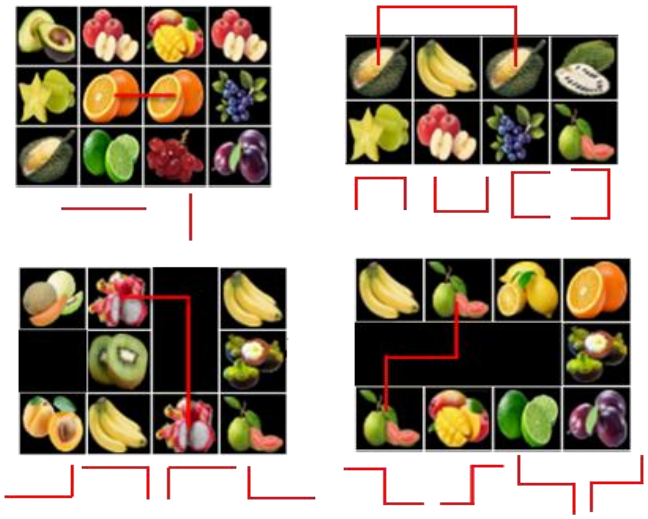
Game adalah permainan komputer yang dibuat dengan teknik dan metode animasi. *Game* juga merupakan bidang AI yang sangat populer berupa permainan antara manusia melawan mesin yang mempunyai intelektual untuk berpikir. Komputer dapat bereaksi dan menjawab tindakan-tindakan yang diberikan oleh lawan mainnya.

Salah satu komputer yang ditanamkan AI untuk *game* bernama *Deep Blue***.** *Deep Blue* adalah sebuah komputer catur buatan IBM pertama yang memenangkan sebuah permainan catur melawan seorang juara dunia (Garry Kasparov) dalam waktu standar sebuah turnamen catur. Kemenangan pertamanya

(dalam pertandingan atau babak pertama) terjadi pada 10 Februari 1996, dan merupakan permainan yang sangat terkenal. Kini telah banyak berkembang game AI yang semakin menarik, interaktif, dan dengan grafis yang sangat bagus.

2.3.1 *Game* Onet Fruit

Onet fruit merupakan permainan dengan mencocokan gambar yang sama dengan syarat kedua gambar itu harus dihubungkan maksimal 3 garis maya tanpa terhalang gambar-gambar yang lain. Umumnya orang yang memainkan onet fruit ini butuh waktu lama dalam menyelesaikanya. Hal ini disebabkan karena setiap gambarnya serupa tapi tak sama, dan dibatasi waktu untuk menyelesaikannya.



Gambar 2.8 Langkah-langkah dalam permainan Onet Fruit.

2.4 Kecerdasan buatan dalam *Game*

Salah satu unsur yang berperan penting dalam sebuah *game* adalah kecerdasan buatan. Dengan kecerdasan buatan, elemen-elemen dalam *game* dapat berperilaku sealami mungkin layaknya manusia.

Game AI adalah aplikasi untuk memodelkan karakter yang terlibat dalam permainan baik sebagai lawan, ataupun karakter pendukung yang merupakan bagian dari permainan tetapi tidak ikut bermain (NPC = *Non Playable Character*). Peranan kecerdasan buatan dalam hal interaksi pemain dengan permainan adalah pada penggunaan interaksi yang bersifat alami yaitu yang biasa digunakan manusia untuk berinteraksi dengan sesama manusia. Contoh media interaksi ialah:

- Penglihatan (*vision*)
- Suara (*voice*), ucapan (*speech*)
- Gerakan anggota badan (*gesture*)

Untuk pembentukan *Artificial Intelligence* pada *game* ternyata digunakan pula algoritma, yaitu jenis pohon n-ary untuk suatu struktur. Implementasi pohon (*tree*) ini biasa disebut *game tree*. Berdasarkan *game tree* inilah sebuah game disusun algoritma kecerdasan buatannya. *Artificial intellegence* yang disematkan dalam sebuah game yang membentuk analisis *game tree* biasanya merepresentasikan kondisi atau posisi permainan dari *game* sebagai suatu node, dan merepresentasikan langkah yang mungkin dilakukan sebagai sisi berarah yang menghubungkan node kondisi tersebut ke anak (*child*) sebagaimana representasi suatu pohon (*tree*).

Namun, biasanya representasi langsung tersebut mempunyai kelemahan, yaitu representasi data pohon akan menjadi sangat lebar dan banyak. Mungkin bagi sebuah mesin komputer mampu melakukan kalkulasi sebanyak apapun masalah, namun *game tree* yang lebar dan besar memberikan beberapa masalah, antara lain konsumsi proses memori, kapasitas penyimpanan yang cukup besar

dan kinerja yang kurang pada konsol game berspesifikasi rendah. Karena itu dibentuklah beberapa algoritma dan penyederhanaan bagi sebuah *game tree*.

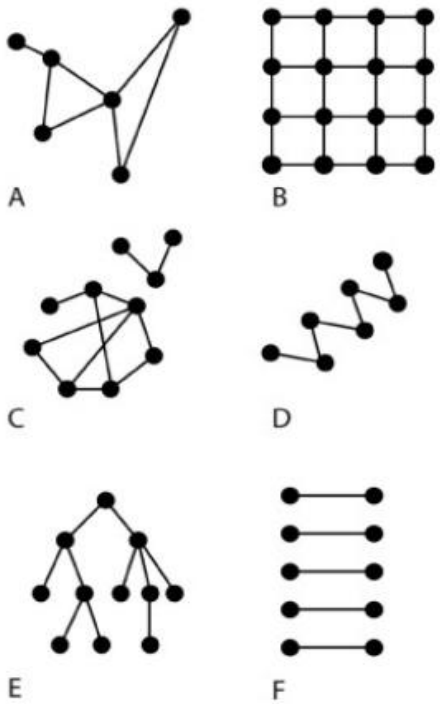
2.4.1 Penerapan Kecerdasan Buatan dalam *Game* dengan Graf

Dalam pengembangan game, banyak ditemukan penggunaan teori-teori di bidang ilmu komputer seperti logika, matriks, ilmu peluang, dan juga graf. Graf secara formal didefinisikan sebagai himpunan pasangan (V, E). Dituliskan sebagai:

G = {V, E} dimana,

- V adalah himpunan tidak kosong dari simpul-simpul (*vertices* atau *node*),
- E adalah himpunan sisi (*edges* atau *arcs*) yang menghubungkan sepasang simpul.

Gambar di bawah ini memperlihatkan berbagai macam contoh graf.



Gambar 2.9. Contoh *graf*

Setiap node dalam graf biasanya diberikan sebuah nama, dapat berupa huruf atau angka. Graf dapat memiliki sisi yang memiliki bobot. Salah satu bentuk graf yang yang digunakan dalam penelitian ini adalah graf pohon. Pohon merupakan sebuah graf yang tidak memiliki sirkuit di dalamnya, artinya ia tidak memiliki koneksi yang dapat membuat lintasan kembali ke node awalnya. Struktur data pohon banyak digunakan dalam *game* dengan kecerdasan buatan untuk pengambilan keputusan.

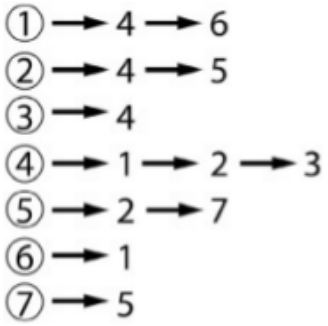
2.4.2 Representasi Graf

Representasi graf ada bermacam-macam, di sini akan diberikan dua contoh representasi graf dalam program, yaitu dengan *matriks* dan *adjacency list*.

	①	②	③	④	⑤	⑥	⑦
①	0	0	0	1	0	1	0
②	0	0	0	1	1	0	0
③	0	0	0	1	0	0	0
④	1	1	1	0	0	0	0
⑤	0	1	0	0	0	0	1
⑥	1	0	0	0	0	0	0
⑦	0	0	0	0	1	0	0

Gambar 2.10. Representasi graf dengan matriks

Setiap angka 1 merepresentasikan adanya koneksi antara node-node yang terdapat pada graf. Setiap angka 0 merepresentasikan tidak adanya koneksi antar kedua node dalam graf.



Gambar 2.11. Representasi graf dengan adjacency list

Representasi ini memperlihatkan setiap node yang terkoneksi dengan node-node lainnya. Misalnya, node 1 terkoneksi dengan node 4 dan node 6, node 2 terkoneksi dengan node 4 dan node 5, dan seterusnya.

2.5 Android

Android adalah sistem operasi yang digunakan di smartphone dan juga tablet PC. Android pertama kali dikembangkan oleh perusahaan bernama Android Inc., dan pada tahun 2005 di akuisisi oleh raksasa Internet Google. Android dibuat dengan basis kernel Linux yang telah dimodifikasi, dan untuk setiap release-nya diberi kode nama berdasarkan nama hidangan makanan. Keunggulan utama Android adalah gratis dan *open source*. Keuntungan open source, banyak pengembang software yang bisa melihat dan memanfaatkan kode itu serta bisa membuat aplikasi baru di dalamnya. Berbagai aplikasi android diwadahi dalam sebuah portal, yaitu Android Market, sehingga pengguna tinggal meng-*install* aplikasi pilihannya.

2.5.1 Fitur-fitur yang ada di android

- **Framework Aplikasi** , pengembang memiliki akses penuh framework API

(*Application Programming Interface*) yang sama yang digunakan oleh aplikasi inti.

- **Mesin virtual Dalvik** dioptimalkan untuk perangkat mobile.
- **Integrated browser** berdasarkan *engine open source* WebKit
- **Grafis** yang dioptimalkan dan didukung oleh perpustakaan grafis 2D, grafis 3D berdasarkan spesifikasi opengl ES 1,0
- **SQLite** untuk penyimpanan data
- **Media Support** yang mendukung audio, video, dan gambar (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF)
- **GSM Telephony** (tergantung hardware)
- **Bluetooth, EDGE, 3G, dan WiFi**
- **Kamera, GPS, kompas, dan accelerometer**
- **Lingkungan Development** yang lengkap: perangkat emulator, tools untuk debugging, profil dan kinerja memori, dan plugin untuk Eclipse IDE.

2.5.2 Versi Android

Android Versi 1.1

Dirilis oleh Google pada bulan Maret 2009.

***Cupcake (*1.5)**

Hanya selang 1 bulan Google merilis **Android 1.5 (***Cupcake***)** dengan SDK (*Software Development Kit*).

***Donut (*1.6)**

Versi ini dirilis pada September dengan menampilkan proses pencarian yang lebih baik, penggunaan baterai indikator dan kontrol *applet* VPN (*Virtual Private Network*) atau jaringan pribadi.

Eclair (2.0 / 2.1)

Versi Android awal yang mulai dipakai oleh banyak smartphone, fitur utama Eclair yaitu perubahan total struktur dan tampilan user interface dan merupakan versi Android yang pertama kali mendukung format HTML5.

Froyo / *Frozen Yoghurt* (2.2)

Android 2.2 dirilis dengan 20 fitur baru, antara lain peningkatan kecepatan, fitur Wi-Fi hotspot tethering dan dukungan terhadap Adobe Flash.

Gingerbread (2.3)

Perubahan utama di versi 2.3 ini termasuk update UI, peningkatan fitur *soft keyboard* & *copy/paste*, *power management*, dan support *Near Field Communication*.

Honeycomb (3.0, 3.1 dan 3.2)

Merupakan versi Android yang ditujukan untuk gadget / device dengan layar besar seperti Tablet PC; Fitur baru Honeycomb yaitu dukungan terhadap prosessor multicore dan grafis dengan hardware acceleration.

Ice Cream Sandwich (4.0)

Anroid 4.0 Ice Cream Sandwich diumumkan pada 10 Mei 2011 di ajang *Google*

I/O Developer Conference (San Francisco) dan resmi dirilis pada tanggal 19 Oktober 2011 di Hongkong. Android versi 4.0 ini dapat digunakan di smartphone ataupun tablet. Fitur utama yang ditambahkan di Android 4.0 ialah *Face Unlock*, *Android Beam*, perubahan major *User Interface*, dan ukuran layar standar (*native screen*) beresolusi 720p *(high definition)*.

2.5.3 *Tools* Pendukung Pembuatan Aplikasi Android

2.5.3.1 Eclipse

Eclipse adalah sebuah IDE (*Integrated Development Environment*) untuk mengembangkan perangkat lunak dan dapat dijalankan di semua platform (*platform-independent*). Berikut ini adalah sifat dari Eclipse:

- **Multi-platform**
- **Mulit-language**: Eclipse dikembangkan dengan bahasa pemrograman Java, akan tetapi Eclipse mendukung pengembangan aplikasi berbasis bahasa pemrograman lainnya, seperti C/C++, Cobol, Python, Perl, PHP, dan lain sebagainya.
- **Multi-role**: Selain sebagai IDE untuk pengembangan aplikasi, Eclipse pun bisa digunakan untuk aktivitas seperti dokumentasi, test perangkat lunak, pengembangan web, dan lain sebagainya.

Secara standar Eclipse selalu dilengkapi dengan JDT (*Java Development Tools*), *plug-in* yang membuat Eclipse kompatibel untuk mengembangkan program Java, dan PDE (*Plug-in Development Environment*) untuk

mengembangkan *plug-in* baru. Eclipse beserta *plug-in*-nya diimplementasikan dalam bahasa pemrograman Java. Konsep Eclipse adalah IDE yang terbuka (*open*), mudah diperluas (*extensible*) untuk apa saja, dan tidak untuk sesuatu yang spesifik. Apabila ingin mengembangkan program C/C++ terdapat *plug-in* CDT (*C/C++ Development Tools*).

2.5.3.2 Android SDK *(Software Development Kit)*

Android SDK merupakan library yang berisi kumpulan *tools* / alat bantu yang dibutuhkan dalam membangun / mengembangkan sebuah aplikasi di android dengan menggunakan bahasa pemrograman Java.

Setelah selesai menginstal SDK, selanjutnya menambahkan Platform Android yakni:

1. Buka Android SDK and AVD Manager.
2. Pada bagian Available Package, centang **SDK Platform Android 2.2 API 8**.
3. Klik ***Install Selected*** untuk mengunduh dan menginstall.

Tahap selanjutnya membuat AVD *(Android Virtual Device).* AVD dapat juga disebut sebagai emulator android.

Tahapan membuat AVD:

- Pada kotak dialog Android SDKdan AVD Manager, pilih **New**
- Isikan seperti berikut:

Tabel 2.1 Tahapan membuat AVD

Name	Froyo
Target	Android 2.2 (atau yang lain)
SDCard	Biarkan saja
Snapshot	Biarkan saja

Skin	HVGA
Hardware	Biarkan saja

- Klik tombol **Create AVD**.

Setelah selesai, AVD yang telah kita buat akan tampil di list AVD. Pilih AVD Froyo tadi lalu klik tombol start yang ada disebelah kanan, maka akan tampil seperti berikut:



Gambar 2.12 Emulator Android 2.2 Froyo

2.5.3.3 ADT (*Android Development Tools*)

ADT merupakan sebuah *plugin* untuk eclipse. Sebelum menggunakan atau menginstal ADT, pastikan bahwa eclipse nya sudah terinstal dan kompatibel dengan ADT dan android SDK.

Tahapan menginstal ADT:

- Jalankan eclipse, lalu klik menu **Help** > **Install New Software**
- Klik tombol **Add**, di pojok kanan atas
- Maka akan ada pop up, silahkan anda beri nama **ADT**
- Pada kolom location masukkan link kode berikut

  `https://dl-ssl.google.com/android/eclipse/`

- Klik Ok.
- Dalam dialog **Available Software**, centang pada bagian **Developer Tools** lalu klik **Next**.
- Pada tampilan berikutnya, Anda akan melihat daftar tools yang akan di download lebih detail. Pilih **Next**.
- Anda akan melihat license agreement. Pilih **Accept**, kemudian klik **Finish**.
- Setelah proses instalasi selesai , restart eclipse.

Konfigurasi Plugin ADT:

- Jalankan Eclipse, pilih **window > Preference**
- Pada panel sebelah kiri, pilih **Android**
- Pada SDK Location, klik **Browse**, cari kemudian pilih SDK File yang sudah anda miliki sebelumya .
- Pilih salah satu platform android, kmeudian klik **Apply**.
- Klik **Ok**.

2.6 *Tools* yang digunakan

2.6.1 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. Dengan menggunakan OOP maka dalam melakukan pemecahan suatu masalah kita tidak melihat bagaimana cara menyelesaikan suatu masalah tersebut (terstruktur) tetapi objek-objek apa yang dapat melakukan pemecahan masalah tersebut.

Pemrograman orientasi-objek menekankan konsep berikut:

- Kelas (*Class*) - kumpulan atas definisi data dan fungsi-fungsi dalam suatu unit untuk suatu tujuan tertentu.
- Objek (*Object*) - membungkus data dan fungsi bersama menjadi suatu unit dalam sebuah program komputer. Objek merupakan dasar dari modularitas dan struktur dalam sebuah program komputer berorientasi objek.
- Abstraksi (*Abstract*) - kemampuan sebuah program untuk melewati aspek informasi yang diproses olehnya, yaitu kemampuan untuk fokus pada inti.
- Enkapsulasi (*Encapsulation*) - Memastikan pengguna sebuah objek tidak dapat mengganti keadaan dalam dari sebuah objek; hanya metode dalam objek tersebut yang diberi ijin untuk mengakses keadaannya.

- Polimorfisme (*Polimorfism*) melalui pengiriman pesan.
- Inheritas (*Inheritance*) - Mengatur polimorfisme dan enkapsulasi dengan mengijinkan objek didefinisikan dan diciptakan dengan jenis khusus dari objek yang sudah ada.

2.6.2 UML

UML (*Unified Modeling Language*) merupakan bahasa standar yang bekerja dalam *object-oriented* untuk menentukan, memvisualisasikan, merancang, dan mendokumentasikan elemen-elemen informasi yang terdapat dalam sistem *software*. UML mulai diperkenalkan oleh *Object Management Group*, sebuah organisasi yang telah mengembangkan model, teknologi, dan standar OOP sejak tahun 1980-an. UML merupakan dasar bagi perangkat (*tool*) desain berorientasi objek dari IBM.



Gambar 2.13 Logo UML

Dalam UML terdapat beberapa diagram untuk memodelkan aplikasi berorientasi objek, yaitu:

1. *Use Case diagram* , untuk memodelkan proses bisnis dan merepresentasikan sebuah interaksi antara aktor dengan sistem.

2. *Class diagram* , untuk menggambarkan keadaan (atribut/properti) suatu sistem, sekaligus menggambarkan struktur dan deskripsi class, package dan objek beserta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi, dan lain-lain. Class memiliki tiga area pokok yakni nama (dan stereotype) , atribut , dan metoda .
3. *Statechart diagram* , untuk memodelkan perilaku objects di dalam sistem atau menggambarkan transisi dan perubahan keadaan (dari satu state ke state lainnya) suatu objek pada sistem. Pada umumnya statechart diagram menggambarkan class tertentu .
4. *Activity diagram* , untuk memodelkan perilaku *Use Cases* dan *objects* di dalam system.
5. *Sequence diagram* , untuk memodelkan pengiriman pesan (*message*) antar objects dan juga digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respons dari sebuah event untuk menghasilkan output tertentu.
6. *Collaboration diagram* , untuk memodelkan interaksi antar objects seperti sequence diagram, tetapi lebih menekankan pada peran masing-masing objek dan bukan pada waktu penyampaian *message*.
7. *Component diagram* , untuk memodelkan komponen object atau menggambarkan struktur dan hubungan antar modul yang berisi code, termasuk ketergantungan (*dependency*) di antaranya.
8. *deployment diagram*, untuk memodelkan distribusi aplikasi.

2.6.3 Java

Java adalah bahasa pemrograman yang dapat diimplementasikan pada berbagai lingkungan, seperti internet, telepon genggam maupun komputer. Java disusun oleh James Gosling yang dibantu oleh rekan-rekannya seperti Patrick Naugton, Chris Warth, Ed Frank, dan Mike Sheridan di suatu perusahaan perangkat lunak yang bernama Sun Microsystems, pada tahun 1991. Saat ini java merupakan bagian dari Oracle dan dirilis tahun 1995. Bahasa pemrograman java mula-mula diinisialisasi dengan nama “Oak”, namun pada tahun 1995 diganti namanya menjadi “Java”.

Alasan utama pembentukan bahasa Java adalah untuk membuat aplikasi-aplikasi yang dapat diletakkan diberbagai macam perangkat elektronik, seperti microwave oven dan remote control, sehingga java harus bersifat portabel atau yang sering disebut dengan platform independent (tidak bergantung pada platform). Itulah yang menyebabkan dalam dunia pemrograman Java, dikenal adanya istilah ‘*write once, run everywhere*’, yang berarti kode program hanya ditulis sekali, namun dapat dijalankan di bawah platform manapun, tanpa harus melakuka:n perubahan kode program.

Bahasa pemrograman Java sendiri secara garis besar dikelompokan menjadi tiga edisi, yaitu:

- Java 2 Standar Edition (J2SE), untuk mengembangkan aplikasi-aplikasi desktop dan applet.
- Java 2 Enterprise Edition (J2EE), untuk mengembangkan aplikasi-aplikasi berbasis client/server berskala enterprise.

- Java 2 Micro Edition (J2ME), untuk mengembangkan aplikasi pada berbagai perangkat kecil dengan jumlah memori, kapasitas penyimpan dan user interface terbatas, seperti ponsel dan PDA.

2.6.3.1 Kelebihan dan Kekurangan Java

➔ Kelebihan java:

- *Multiplatform*.
- OOP (*Object Oriented Programming*)
- *Library* yang lengkap
- Bergaya C++

➔ **Kekurangan Java:**

- Mudah didekompilasi.
- Penggunaan memori yang banyak.

2.6.4 SQLite

SQLite merupakan sebuah sistem manajemen basisdata relasional yang bersifat ACID-compliant dan memiliki ukuran pustaka kode yang relatif kecil, ditulis dalam bahasa C. SQLite merupakan sebuah library proses yang menerapkan serverless (mandiri tanpa server), *zero configuration*, *database* SQL transaksional dan merupakan proyek yang bersifat public domain yang dikerjakan oleh D. Richard Hipp. Dengan demikian bebas untuk digunakan untuk tujuan apa pun, komersial atau swasta.

SQLite adalah mesin database SQL embedded. Tidak seperti kebanyakan database SQL lainnya, SQLite tidak memiliki proses server yang terpisah. SQLite membaca dan menulis secara langsung ke disk. Database SQL lengkap dengan *multiple* tabel, *indices*, *triggers*, dan *views*, semua terdapat dalam sebuah disk file tunggal. Format file database adalah cross-platform yaitu kita bebas mengcopy database antara 32-bit dan sistem 64-bit atau antara arsitektur *big-endian* dan *little-endian*. Fitur-fitur ini membuat SQLite menjadi pilihan populer sebagai *Application File Format*.