MODUL PRAKTIKUM

KECERDASAN BUATAN

Program Studi Matematika
Fakultas Sains dan Teknologi
UIN Maulana Malik Ibrahim Malang

2022

MODUL 1
INTELLIGENT AGENT

1.1. Tujuan Praktikum

Mahasiswa diharapkan memahami konsep *agent*, *relational agent*, dan cara merancang sebuah struktur agent. Pada praktikum ini, mahasiswa akan berlatih membuat sebuah struktur agent Robot Pembersih *(Vacuum Cleaner)*.

1.2. Dasar Teori

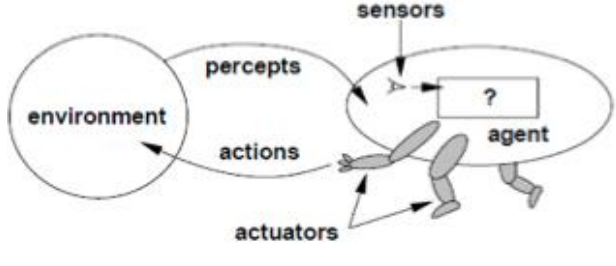
1.2.1. Definisi *Agent, Relational Agent,* dan *Task Environment*

Agen *(Agent)* merupakan sesuatu yang dapat menerima inputan dari lingkungan sekitar *(enviroment)* melalui pendeteksi *(sensor)* dan merespon hasil inputan dengan sebuah aksi/tindakan melalui suatu penggerak *(actuator)*. Agen dapat berupa manusia, robot, dan software. Manusia mempunyai mata, telinga, lidah, hidung, dan kulit sebagai *sensor* dan kaki, tangan, dan gigi sebagai *actuator*. Pada robot, kamera, sinar infra red, pendeteksi sidik jari merupakan sensor sedangkan alat-alat penggerak (hidrolik) adalah *actuator*. Adapun sensor pada agen software adalah keyboard, file contents, dan network packets sedangkan actuator nya adalah proses penampakan objek pada screen, penulisan ke dalam file, dan pengiriman paket data ke dalam jaringan.

Sebuah agen tidak hanya bertindak untuk mencapai tujuannya saja, namun dia juga harus mampu memilih tindakan yang tepat sehingga tindakan yang dilakukannya itu bermanfaat. Jika sebuah agen bertindak secara efektif dengan memaksimumkan ukuran kinerja, merekam semua hal yang diamatinya, dan bertindak dengan tepat, maka agen ini disebut *Relational Agent*. Relational egent dapat diukur melalui kinerja dari agent (performance measure). Sebagai contoh, mahasiswa harus mempunyai Indek Prestasi Komulatif (IPK) yang bagus untuk memperoleh gelar sarjana. Pegawai harus mempunyai gaji bulanan untuk menjadi orang kaya.

Menurut Russel dan Norvig, ketika sebuah agent dirancang maka hal pertama yang harus didefinisikan adalah *Task Environment* yaitu ***P****ercept* (inputan indera si agen), A*ction* (tindakan yang dilakukan oleh agen), ***G****oal* (tujuan si agen), dan ***E****nvironment* (lingkungan dimana si agen berada). Ini sering disingkat dengan PAGE.

Konsep agent, relational agent, dan task environment dapat dijelaskan melalui Gambar 1.1.



Gambar 1.1 : Agent dan Task Environment

Berikut adalah contoh Agen Robot Pembersih Lantai (Vacuum Cleaner) bertugas membersihkan lantai yang penuh dengan sampah pada dua lokasi A dan B.

- **Percept**: Alat Pembersih
- **Action**: membersihkan sampah, memindahkan agen ke kiri, memindahkan agen ke kanan, dan tidak istirahat (agen tidak melakukan apa-apa).
- **Goal**: membersihkan sampai pada kedua lokasi.
- **Environment**: lokasi dengan sampah dan lokasi yang sudah bersih.

1.2.2. Perancangan dan Struktur Agent

Perancangan sebuah agent dapat dilakukan melalui dua langkah. Langkah-langkah tersebut adalah *Agent Function* dan *Agent Program*. *Agent program* tidak dapat diimplementasikan sebelum *Agent Function* dirancang.

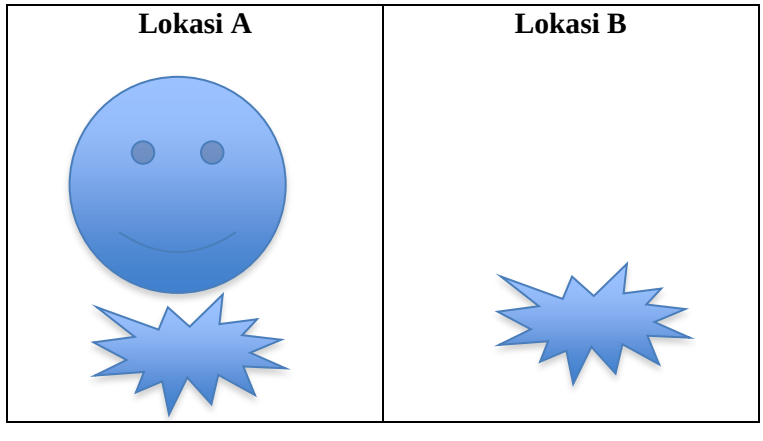
Definisi: Agent Function adalah sebuah fungsi yang memetakan semua urutan inputan *(percept sequence)* terhadap tindakan *(action)* yang dilakukan.

$$\mathcal{F}: \rho * \rightarrow \text{A}$$

Definisi: Agent Program adalah sebuah program yang mengimplementasikan fungsi *F* terhadap perancangan agent.

1.3. Perancangan Agent Robot Pembersih (Vacuum Cleaner).

Pada pertemuan praktikum ini, mahasiswa dimintakan membuat perancangan agent Robot Pembersih. Gambar 1.2 menunjukkan posisi robot dan lokasi yang akan dibersihkan.



Gambar 1.2. Robot Vacuum Cleaner

Langkah-langkah untuk merancang struktur agent Robot Vacuum Cleaner adalah:

1. Mendefinisikan *Task Environment*:
   - Percepts: lokasi dan status, misal: [*A,Kotor*]

     Contoh: *Percept Sequence* (urutan inputan)

     {[*A,Kotor*], [*A,Bersih*], [*B, Kotor*], [*B, Bersih*],....}

     {[*A,Kotor*], [*A,Kotor*], [*A, Kotor]*, [*A,Bersih*],....}
   - Acton: *DoKekiri, DoKekanan, DoSedot, DoSantai*
   - Goal: membersihkan kotoran pada kedua lokasi
   - Environment: lokasi A dan B beserta kotorannya
2. Membuat *Agent Function* **RobotPembersih**

$$\mathcal{F}(\{\dots,[*,Kotor]\}) \rightarrow doSedot$$
$$\mathcal{F}(\{\dots,[A,Bersih]\}) \rightarrow doKeKanan$$
$$\mathcal{F}(\{\dots,[B,Bersih]\}) \rightarrow doKeKiri$$

3. Mengimplementasikan Agent Program **RobotPembersih**

```
function RobotPembersih (status,lokasi) return action
    If status := kotor then return doSedot
      else if lokasi := A then return doKeKanan
   else return doKeKiri
end function
```

1.4. Mengimplementasikan Agent Menggunakan Java Applet.

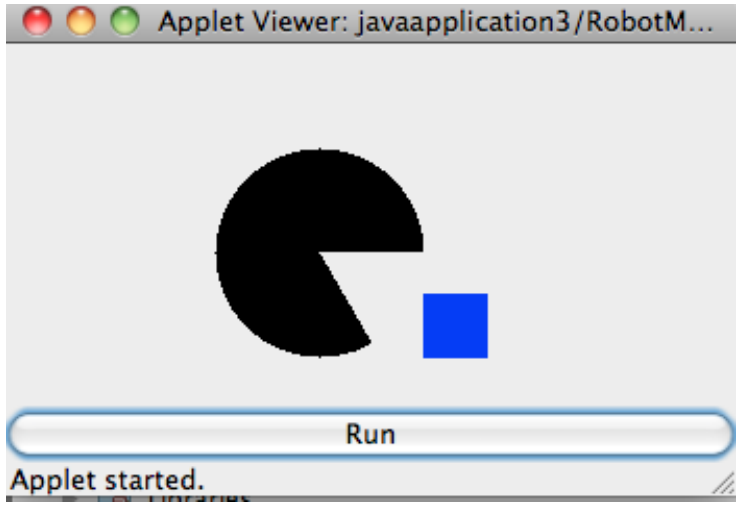
1.4.1. Membuat GUI Robot Vacuum Cleaner dan doSedot()

```
/*
 * Mengimplementasikan Robot Vacuum Cleaner dengan method doSedot()
 * menggunakan metode Simple Based Agent.
 */
package javaapplication3;

import java.applet.Applet;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**
 *
 * @author Irvanizam Zamanhuri
 * Date : 01 Oktober 2013
 */
public class RobotMakan extends Applet implements ActionListener{

    private Image robot_image;
    Button b;
    int x = 30, y = 30;           //posisi sampah
    int statusMulutRobot = 300;   //robot buka mulut
    /**
     * Initialization method that will be called after
     * the applet is loaded into the browser.
```

```
     */
    public void init() {
    // TODO start asynchronous download of heavy resources
        b = new Button("Run");
        setLayout(new BorderLayout());
         add("South",b);

         b.addActionListener(this);
    }
    //TODO overwrite start(), stop() and destroy() methods
    public void paint( Graphics g )
    {
        super.paint( g );

        // menggambar robot dengan mulut terbuka
        g.fillArc( 100, 50, 100, 100, 0, statusMulutRobot );

        //menggambar sampah dalam bentuk kotak berwarna biru
        g.setColor(Color.blue);
        g.fillRect( 200, 120, x, y);
        g.drawRect( 200, 120, x, y);

    }

    public void actionPerformed(ActionEvent e) {
            if (e.getSource() == b) {
            System.out.println("Button 1 was pressed");
                  doSedot();
            }
            else
            System.out.println("Button 2 was pressed");
    }

    public void doSedot()
    {
        if(x == 0 && y==0)
        {
            statusMulutRobot = 300; // robot buka mulut
            x = 30;
            y = 30;
        } else {
            x = 0;   //posisi sampah sudah bersih
            y = 0;   //posisi sampah sudah bersih
            statusMulutRobot = 360; // probot tutup mulut
        }
        repaint();
    }
}
```

Program 1.1. Implementasi GUI Robot Vacuum Clleaner dan Method doSedot()

Jika program 1 diimplementasikan menggunakan text editor Netbeans 7.1 maka program tersebut dapat di-compile dan dijalankan dengan memilih menu **Run**, lalu pilih submenu **Run File**. Output dari program 1 adalah seperti tampilan pada Gambar 1.3.



Gambar 1.3. Applet Robot Vacuum Cleaner

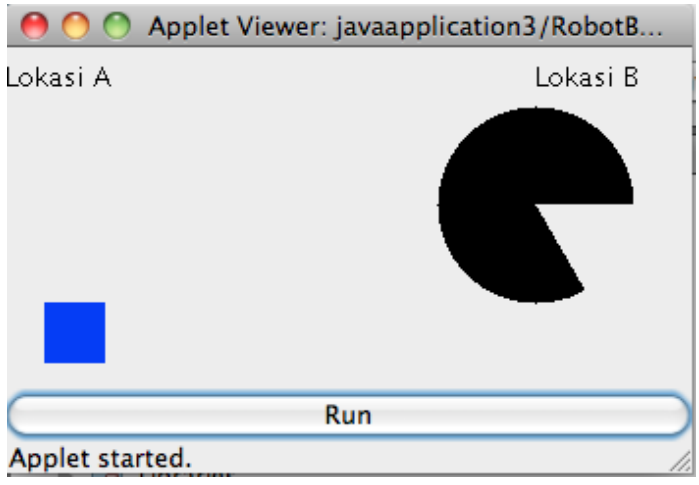
1.4.2. Membuat method doKeKiri() dan doKeKanan()

```
/*
 * Mengimplementasikan Robot Vacuum Cleaner dengan method doKeKiri()
 * dan doKeKanan() menggunakan metode Simple Based Agent.
 */
package javaapplication3;
import java.applet.Applet;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
/**
 *
 * @author Irvanizam Zamanhuri
 */
public class RobotBerjalan extends Applet implements ActionListener{

    private Image robot_image;
    Button b;

    int x = 30, y = 30;     //posisi sampah
    int z = 300;      //posisi robot sedang membuka mulut

    int xPosRobot = 20;    //posisi awal robot
    int yPosRobot = 30;    //posisi awal robot

    int xPosSampah = 20;   //posisi awal sampah
    int yPosSampah = 130;    //posisi awal sampah

    /**
     * Initialization method that will be called after
     * the applet is loaded into the browser.
     */
    public void init() {
    // TODO start asynchronous download of heavy resources
        b = new Button("Run");

        setLayout(new BorderLayout());
```

```
        add("South",b);

        b.addActionListener(this);
    }
    //TODO overwrite start(), stop() and destroy() methods
    public void paint( Graphics g )
    {
        super.paint( g );
        g.drawString("Lokasi A", 0, 20);
        g.drawString("Lokasi B", 270, 20);
        //menggambar robot
        g.fillArc( xPosRobot, yPosRobot, 100, 100, 0, z );

        //menggambar sampah dalam bentuk kotak berwarna biru
        g.setColor(Color.blue);
        g.fillRect( xPosSampah, yPosSampah, x, y);
        g.drawRect( xPosSampah, yPosSampah, x, y);

    }
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == b) {
            System.out.println("Button 1 was pressed");
                  if(xPosRobot == 20) {
                      doKeKanan();
                  } else {
                      doKeKiri();
                  }
            }
            else
            System.out.println("Button 2 was pressed");
    }
    public void doKeKanan()
    {
        z = 300;
        xPosRobot = 220;   // posisi robot pindah
        repaint();
    }
    public void doKeKiri()
    {
        z = 300;
        xPosRobot = 20;
        repaint();
    }
}
```

Program 1.2. Implementasi Method doKeKiri() dan doKeKanan()

Program 1.2 dapat di-compile dan dijalankan melalui pilihan menu **Run**, lalu pilih submenu **Run File**. Kemudian output dari program 2 ditampilkan seperti Gambar 1.4.



Gambar 1.4. Applet Robot Vacuum Cleaner

Latihan:

Robot Vacuum Cleaner merupakan robot yang membersihkan sampah pada Lokasi A dan B. Jika posisi Robot pada suatu lokasi dimana ada sampah pada lokasi itu, maka Robot akan membersihkan lokasi dengan memakan sampah yang ada. Sebaliknya, robot dengan senang hati akan berpindah ke lokasi yang lain. Potongan Program 2 belum secara sempurna diimplementasikan. Silakan lengkapi potongan program tersebut sehingga Robot mampu mendeteksi sampah dan kemudian membersihkan sampah pada kedua lokasi A dan B.

MODUL 2
ROBOT PACMAN

2.1. Tujuan Praktikum

Mahasiswa mampu mengimplementasikan Agent Robot Pacman sederhana dengan menggunakan bahasa pemrograman Java Applet. Implementasi agent ini menggunakan metode Simple Reflex Agents.

2.2. Dasar Teori

2.2.1. Jenis-Jenis Agent Program

Dalam Kecerdasan Buatan, tingkat kesulitan untuk menyelesaikan permasalahan (problem solving) tergantung dari model agent program yang diimplementasikan. Menurut Russel dan Norvig, terdapat 5 jenis model agent program.

1. Simple Reflex Agents :

Merupakan agent yang bekerja berdasarkan reflex. Contohnya, sebuah driver agent(supir taxi otomatis), harus memberikan reflex mengerem ketika terdapat mobil yang berhenti didepanya.

2. Model Based Reflex Agents

Merupakan agent yang bekerja berdasarkan model reflex.

3. Goal Based Agents

Merupakan sebuah agent yang mendasarkan setiap tindakannya untuk mencapai tujuan yang telah ditentukan. Setiap agent akan mempertimbangkan setiap kemungkinan yang akan terjadi pada mesa depan berdasarkan tindakan yang akan/telah dilakukanya.

4. Utility Based Agents

Merupakan sebuah fungsi yang memetakan suatu keadaan kedalam bilangan real, yang menggambarkan derajak kesenangan/kepuasan. Sedikit berbeda dengan Goal Based Agent, tipe ini tidak mengutamakan semua tujua, tetapi akan mengutamakan tujuan mana yang mungkin tercapai berdasarkan kondisi tertentu(tujuan kepuasan, kenyamanan, keefisienan).

5. Learning Agents

Merupakan agent yang tetap melakukan pengecekan terhadap keadaan lingkungan, sehingga dapat memberikan respon yang tepat.

2.3. Mengimplementasikan Agent Pacman dengan Java Applet.

Applet ini terdiri dari tiga buah class, RobotPacMan, RobotAksi, dan MyPoint. Class RobotPacMan digunakan untuk merancang posisi dari panel, button “Start” dan “Stop”.

```
/*
* Class RobotPacMan digunakan untuk mendesain letak Panel,
* Button "Start" dan "Stop". Pada class ini,
* Robot Pacman mencari sampah secara horizontal
* dan membersihkan sampah tersebut.
*/
import java.applet.Applet;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
*
* @author Irvanizam Zamanhuri
*/
public class RobotPacMan extends Applet implements ActionListener {

      Button start;
      Button stop;
      Panel papanTombol;
      RobotAksi papanAksi;

      /**
      * Initialization method that will be called after
      * the applet is loaded into the browser.
      */
      public void init() {
            start = new Button("Start");
            start.addActionListener(this);

            stop = new Button("Stop");
            stop.addActionListener(this);

            papanTombol = new Panel();
            papanTombol.setLayout(new GridLayout());
            papanTombol.add(start);
            papanTombol.add(stop);

            papanAksi = new RobotAksi();
            papanAksi.setBackground(Color.black);

            setLayout(new BorderLayout());
            add("South",papanTombol);
            add("Center",papanAksi);

      }

      @Override
```

```
    public void actionPerformed(ActionEvent ae) {

            if(ae.getSource()==start)
            {
                papanAksi.jalan();
            }
            if(ae.getSource()==stop)
            {
                papanAksi.berhenti();
            }
    }
}
```

Class RobotAksi memperlihatkan robot Pacman berjalan secara horizontal dan kembali membalik ke arah semua. Pada class ini letak semua sampah didefinisikan melalui class MyPoint yang bertipe data array.

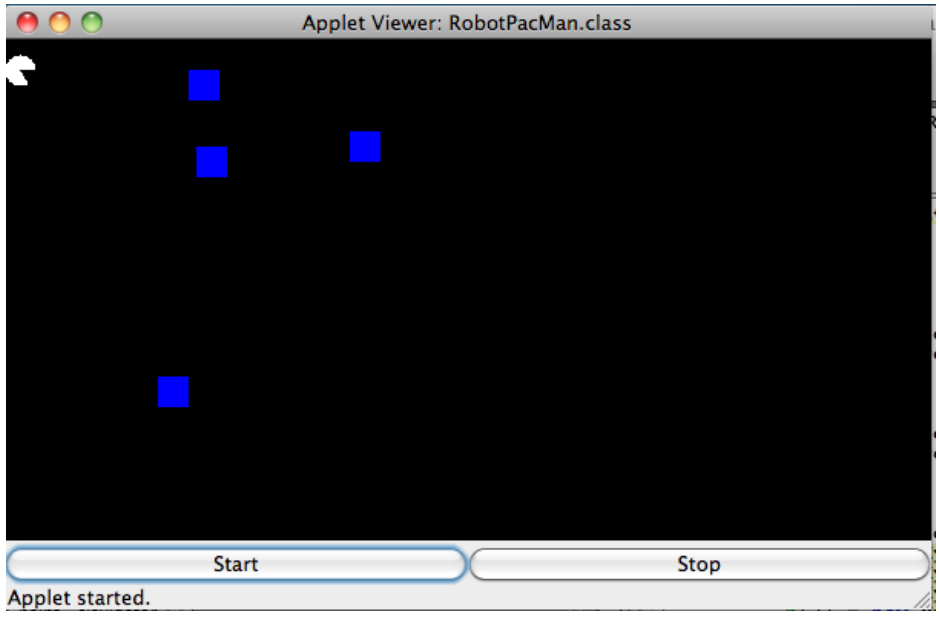
```
class RobotAksi extends RobotPacMan implements Runnable
{
    Thread runner = null;
    Boolean keepRunning;

    MyPoint[] p = new MyPoint[4];

    int x = 0;
    int y = 10;
    int g = 5;
    int incr = 5;

    int arahMulut = 0;

    boolean black=true;

    @Override
    public void run() {
        Dimension d = getSize();
        System.out.println(x);
        while(keepRunning)
        {
           if((x + g) > d.width)
           {
               incr = -incr;  // ubah posisi
               arahMulut = 240; //arah mulut ke kanan
           }
           if (x < 0)
           {
               incr = -incr;
               arahMulut = 0;  // arah mulut ke kiri
           }
           x += incr;
           repaint();
           try {
                runner.sleep(90);
           } catch (InterruptedException ex) {
```

```
Logger.getLogger(RobotAksi.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
        throw new UnsupportedOperationException("Not supported yet.");
    }

    public void jalan()
    {
        if (runner == null) {
            runner = new Thread(this);
            keepRunning = true;
            runner.start();
        }
    }
    public void berhenti()
    {
        if (runner != null) {
            keepRunning = false;
            runner = null;
        }
    }

    public void paint(Graphics g)
    {
        int i;
        if(black) {
            g.setColor(Color.white);
             // robot tutup mulut
            g.fillArc(x,y,20,20,arahMulut,360);
        }
        else {
            g.setColor(Color.white);
            //menggambar robot
            g.fillArc(x,y,20,20,arahMulut,300);
        }

        g.setColor(Color.blue);
        p[0] = new MyPoint(120,20);
        p[1] = new MyPoint(100,220);
        p[2] = new MyPoint(225,60);
        p[3] = new MyPoint(125,70);

        for (i = 0; i < p.length; i++)
                        g.fillRect(p[i].getX(),p[i].getY(),20,20);
        black = !black;
    }
}
```

Class MyPoint merupakan class untuk mendeklarasikan koordinat sampah-sampah. Dengan menggunakan konstraktor *MyPoint(x,y)*, letak koordinat dari sampah dapat terekam dan dengan mudah dipanggil kembali dengan menggunakan method *getX()* dan *getY()*.

```
class MyPoint {

    private int x;
    private int y;

    MyPoint(int x, int y)
    {
        this.x = x;
        this.y = y;
    }

    public int getX()
    {
        return this.x;
    }

    public int getY()
    {
        return this.y;
    }

}
```

Gabungkan ketiga class di atas dan simpan ke dalam satu file RobotPacMan.java. Jalankan file java itu menggunakan perangkat lunak Netbeans 7.1. Class Applet RobotPacMan dapat dilihat seperti Gambar 2.1. Class RobotPacMan juga dapat dijalankan melalui web browser dengan menempelkan class tersebut pada file HTML.



Gambar 2.1. Visualisasi Robot PacMan

Setelah menjalankan class RobotPacMan melalui perangkat lunak Netbeans (compiler java), masukkan class nya ke dalam sebuah file HTML dengan nama misalnya RobotPacMan.Java.

Penempelan class ini dapat dilihat pada Gambar 2.2. Jalankan script ini melalui salah satu web browser. Pastikan bahwa web browsernya telah diinstall plug-in Java Applet.

```
<HTML>
      <HEAD>
            <TITLE>Applet HTML Page</TITLE>
      </HEAD>
<BODY>
<H3><HR WIDTH="100%">Applet HTML Page<HR WIDTH="100%"></H3>
<P>
<APPLET codebase="classes" code="RobotPacMan.class" width=350
height=200></APPLET>
</P>
<HR WIDTH="100%"><FONT SIZE=-1><I>Generated by NetBeans IDE</I></FONT>
</BODY>
</HTML>
```

Gambar 2.2. Program Script RobotPacMan HTML

Tugas:

Lengkapi program di atas sehingga Robot PacMan dapat berjalan baik secara horizontal maupun vertikal. Ketika robot menjumpai sampah, maka robot akan membersihkan sampah-sampah tersebut dengan memakannya.

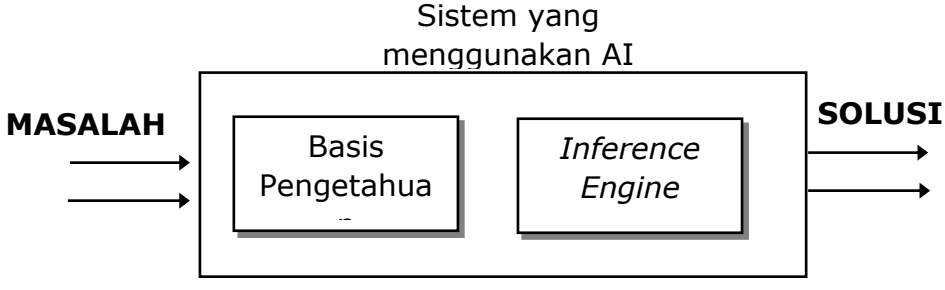
MODUL 3
MASALAH DAN RUANG KEADAAN

3.1. Tujuan

Mahasiswa diharapkan mampu merepresentasikan suatu masalah ke dalam bentuk solusi. Meningkatkan pemahaman mahasiswa terhadap topologi tree sebagai salah satu solusi dalam memecahkan permasalahan Kecerdasan Buatan.

3.2. Dasar Teori

- Pada sistem yang menggunakan kecerdasan buatan, akan mencoba untuk memberikan output berupa solusi dari suatu masalah berdasarkan kumpulan pengetahuan yang ada (Gambar 3.1).



Gambar 3.1. Sistem yang Menggunakan Kecerdasan Buatan

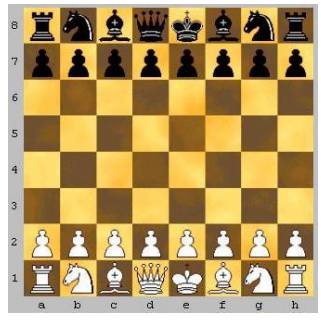
- Pada sistem harus dilengkapi dengan sekumpulan pengetahuan yang ada pada basis pengetahuan. Sistem harus memiliki *inference engine* agar mampu mengambil kesimpulan berdasarkan fakta atau pengetahuan. Output yang diberikan berupa solusi masalah sebagai hasil dari inferensi.
- Secara umum, untuk membangun suatu sistem yang mampu menyelesaikan masalah, perlu dipertimbangkan 4 hal:
  1. Mendefinisikan masalah dengan tepat. Pendefinisian ini mencakup spesifikasi yang tepat mengenai keadaan awal dan solusi yang diharapkan.
  2. Menganalisis masalah tersebut serta mencari beberapa teknik penyelesaian masalah yang sesuai.
  3. Merepresentasikan pengetahuan yang perlu untuk menyelesaikan masalah tersebut.
  4. Memilih teknik penyelesaian masalah yang terbaik.

3.3 Mendefinisikan Masalah Sebagai Suatu Ruang Keadaan

- **Ruang Keadaan (*State Space*),** yaitu suatu ruang yang berisi semua keadaan yang mungkin. Kita dapat memulai bermain catur dengan menempatkan diri pada keadaan awal, kemudian bergerak dari satu keadaan ke keadaan yang lain sesuai dengan aturan yang ada, dan mengakhiri permainan jika salah satu telah mencapai tujuan.
- Pada Permainan Catur, harus ditentukan :

1. Posisi awal pada papan catur;

   Semua bidak diletakkan di atas papan catur dalam 2 sisi, yaitu kubu putih dan kubu hitam (Gambar 3.2).



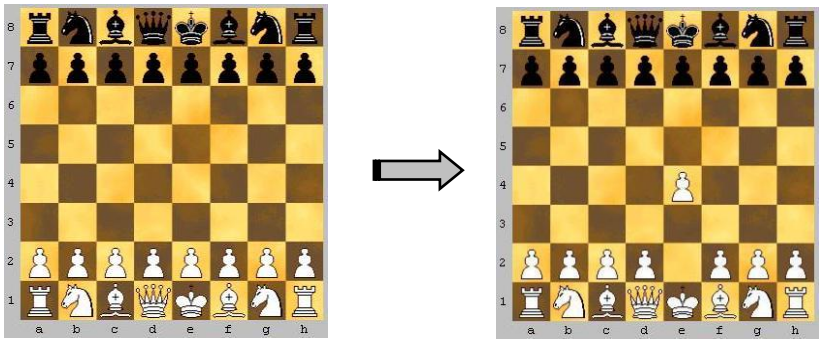
Gambar 3.2. Keadaan Awal Permainan Catur

2. Aturan-aturan untuk melakukan gerakan secara legal:

   Aturan-aturan ini sangat berguna untuk menentukan gerakan suatu bidak, yaitu melangkah dari satu keadaan ke keadaan lain. Misalkan suatu aturan untuk menggerakkan bidak dari posisi (e,2) ke (e,4), dapat ditunjukkan dengan aturan:

   IF **Bidak putih pada Kotak(e,2),**
   And Kotak(e,3) Kosong,
   And Kotak(e,4) Kosong
   Then **Gerakkan bidak dari (e,2) ke (e,4)**

   Seperti terlihat pada Gambar 3.3.



Gambar 3.3. Gerakan bidak catur

3. Tujuan (Goal)

   Tujuan yang ingin dicapai adalah posisi pada papan catur yang menunjukkan kemenangan seseorang terhadap lawannya. yaitu posisi Raja yang sudah tidak dapat bergerak lagi. Gambar 3.4 merupakan salah satu contoh tujuan telah tercapai, yaitu Raja pada bidak hitam sudah tidak dapat bergerak lagi.

Gambar 3.4. Salah Satu Raja Mati

- Sehingga secara umum, untuk mendeskripsikan masalah dengan baik, harus:
  1. Mendefinisikan suatu ruang keadaan;
  2. Menetapkan satu atau lebih keadaan awal;
  3. Menetapkan satu atau lebih tujuan;
  4. Menetapkan kumpulan aturan.

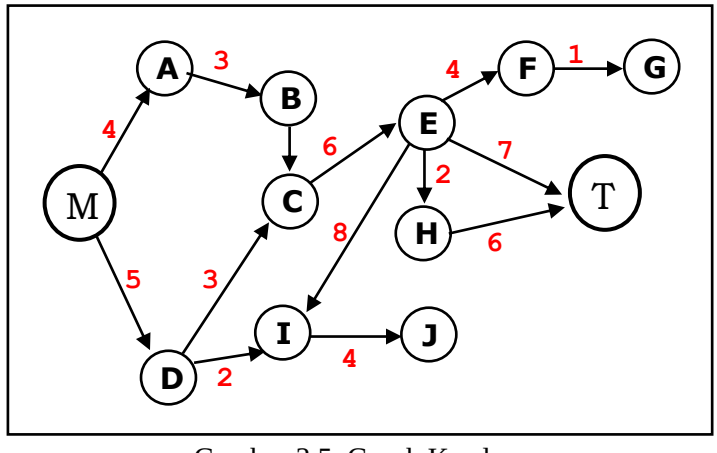
3.4 Representasi Ruang Keadaan

- Ada beberapa cara untuk merepresentasikan Ruang Keadaan, antara lain:

  1. Graph Keadaan

  - Graph terdiri-dari node-node yang menunjukkan keadaan yaitu keadaan awal dan keadaan baru yang akan dicapai dengan menggunakan operator.
  - Node-node dalam graph keadaan saling dihubungkan dengan menggunakan *arc* (busur) yang diberi panah untuk menunjukkan arah dari suatu keadaan ke keadaan berikutnya.
  - Pada Gambar 2.5 menunjukkan graph berarah dengan node M menunjukkan keadaan awal, dan node T adalah tujuan. Pada Gambar 2.5 tersebut, kita dapat melihat ada lintasan 4 dari M ke T, yaitu:
    - ✓ M-A-B-C-E-T
    - ✓ M-A-B-C-E-H-T
    - ✓ M-D-C-E-T
    - ✓ M-D-C-E-H-T

- Pada graph ini, ada juga lintasan yang tidak sampai ke tujuan atau menemui jalan buntu, yaitu:
  - ✓ M-A-B-C-E-F-G
  - ✓ M-A-B-C-E-I-J
  - ✓ M-D-C-E-F-G
  - ✓ M-D-C-E-I-J
  - ✓ M-D-I-J
- Gambar 3.5, tanpa mempertimbangkan arah, akan didapat siklus: D-C-E-I-D, node-node ini akan selalu berulang.

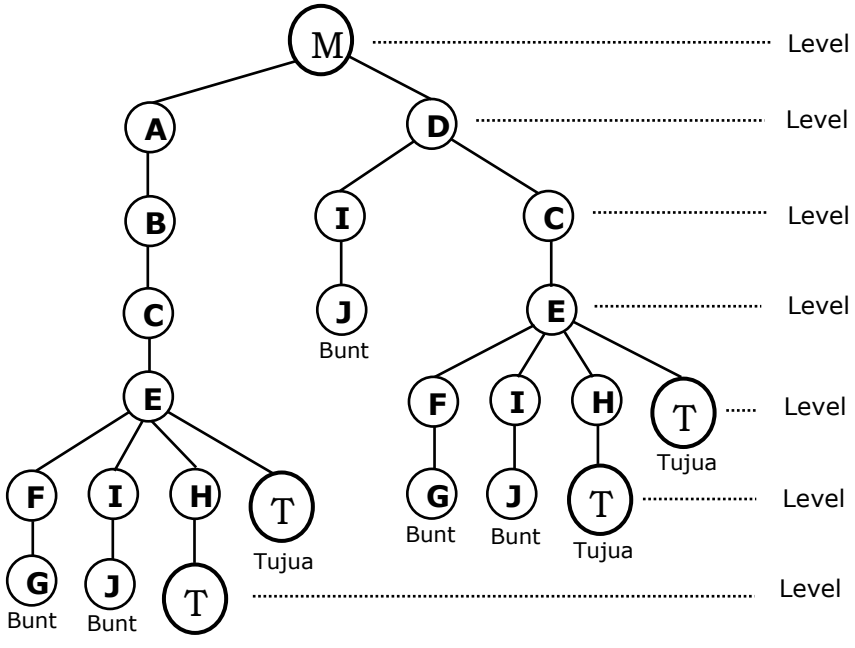


Gambar 3.5. Graph Keadaan

2. Pohon Pelacakan

- Untuk menghindari kemungkinan adanya proses pelacakan suatu node secara berulang, maka digunakan struktur pohon.
- Struktur pohon digunakan untuk menggambarkan keadaan secara hirarkis.
- Pohon juga terdiri-dari beberapa node. Node yang terletak pada level-0 disebut dengan nama "akar". Node akar menunjukkan keadaan awal yang biasanya merupakan topik atau obyek. Node akar ini terletak pada level ke nol.
- Node akar memiliki beberapa percabangan yang terdiri-atas beberapa node successor yang sering disebut dengan nama "anak" dan merupakan node-node perantara. Namun jika dilakukan pencarian mundur, maka dapat dikatakan bahwa node tersebut memiliki predecessor.
- Node-node yang tidak memiliki anak sering disebut dengan nama node "daun" yang menunjukkan akhir dari suatu pencarian, dapat berupa tujuan yang diharapkan (*goal*) atau jalan buntu (*dead end*). Gambar 3.6 menunjukkan pohon pencarian untuk graph pada gambar 3.5 dengan 6 level.

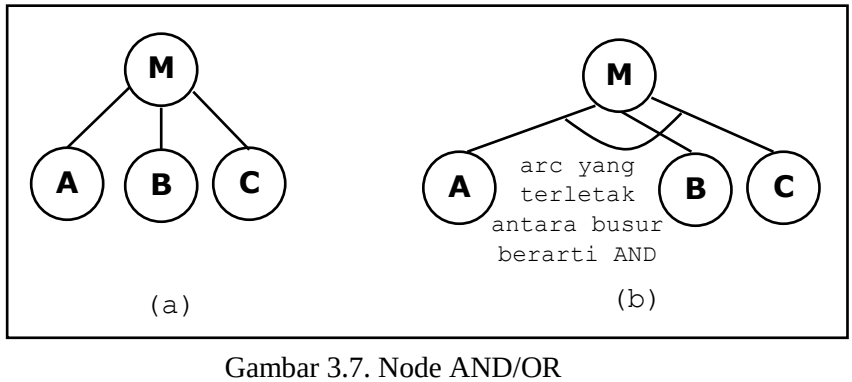
- Pada Gambar 3.6 di bawah ini, sudah tidak terlihat lagi adanya siklus, karena setiap node tidak diperbolehkan memiliki cabang kembali ke node dengan level yang lebih rendah



Gambar 3.6. Struktur Pohon

3. Pohon AND/OR

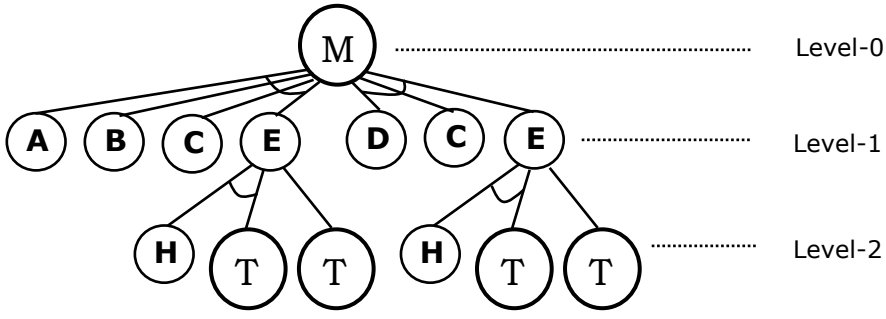
- Pada Gambar 3.7a terlihat ada suatu masalah M yang hendak dicari solusinya dengan 3 kemungkinan yaitu A, B atau C. Artinya, masalah M bisa diselesaikan jika salah satu dari subgoal A, B, atau C tidak terpecahkan.
- Lain halnya dengan gambar 3.7b, masalah M hanya dapat diselesaikan dengan A AND B AND C. Dengan kata lain, untuk memecahkan masalah M, maka harus dipecahkan subgoal A, B dan C terlebih dahulu. Pohon semacam ini disebut dengan Pohon AND/OR.



Gambar 3.7. Node AND/OR

- Gambar 3.8 memperlihatkan pencapaian tujuan pada graph Gambar 2.5 dengan menggunakan Pohon AND/OR. Dengan mengunakan pohon AND/OR, tujuan yang

dicapai pada pohon (Gambar 3.6) sampai pada level-6 bisa dipersingkat hanya sampai pada level-2 saja.



Gambar 3.8. Pohon AND/OR

3.5. Contoh Kasus

- Seorang petani akan menyeberangkan seekor kambing, seekor serigala, dan sayur-sayuran dengan sebuah boat yang melalui sungai. Boat hanya bisa memuat petani dan satu penumpang yang lain (kambing, serigala atau sayur-sayuran). Jika ditinggalkan oleh petani tersebut, maka sayur-sayuran akan dimakan oleh kambing, dan kambing akan dimakan oleh serigala.
- Penyelesaian :

1. Identifikasi ruang keadaan

   Permasalahan ini dapat dilambangkan dengan (JumlahKambing, JumlahSerigala, JumlahSayuran, JumlahBoat). Sebagai contoh: Daerah asal (0,1,1,1) berarti pada daerah asal tidak ada kambing, ada serigala, ada sayuran, dan ada boat.

2. Keadaan awal & tujuan

   Keadaan awal, pada kedua daerah:

   a. Daerah asal: (1,1,1,1)
   b. Daerah seberang: (0,0,0,0)

   Tujuan, pada kedua daerah:

   a. Daerah asal: (0,0,0,0)
   b. Daerah seberang: (1,1,1,1)

3. Aturan-aturan

   Aturan-aturan dapat digambarkan seperti pada Tabel 3.1.

Tabel 3.1. Aturan-aturan masalah teko air

Aturan ke-	Aturan
1.	Kambing menyeberang
2.	Sayuran menyeberang
3.	Serigala menyeberang
4.	Kambing kembali
5.	Sayuran kembali
6.	Serigala kembali
7.	Boat kembali

4. Solusi

   Salah satu solusi yang bisa ditemukan dapat dilihat pada Tabel 3.2.

Tabel 3.2. Contoh solusi masalah petani, kambing, sayuran, dan serigala

Daerah Asal	Daerah Seberang	Aturan yang dipakai
(1,1,1,1)	(0,0,0,0)	1
(0,1,1,0)	(1,0,0,1)	7
(0,1,1,1)	(1,0,0,0)	3
(0,0,1,0)	(1,1,0,1)	4
(1,0,1,1)	(0,1,0,0)	2
(1,0,0,0)	(0,1,1,1)	7
(1,0,0,1)	(0,1,1,0)	1
(0,0,0,0)	(1,1,1,1)	solusi

Tugas

Pelajari dan analisa kasus petani, kambing, sayuran, dan serigala. Buatkan mekanisme pembuatan struktur agent.

MODUL 4
TEKNIK PENCARIAN BLIND SEARCH

4.1. Tujuan Praktikum

Mahasiswa mampu memahami konsep blind search dan dapat mengimplementasikan program salah satu algoritma blind search pada kasus tree. Program ini dibuat dengan menggunakan bahasa pemrograman Java.

4.2. Dasar Teori

4.2.1. Teknik Pencarian Blind Search

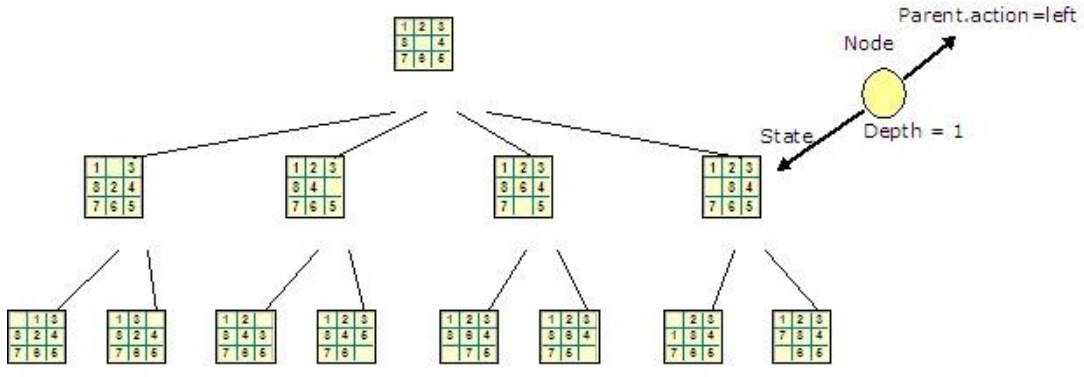
Ada beberapa algoritma yang dikategorikan ke dalam teknik pencarian blind search.

- Breadth First Search (BFS)
- Uniform Cost Search (UCS)
- Depth First Search (DFS)
- Depth Limited Search (DLS)
- Iterative Deepening Search (IDS)
- Bidirectional Search (BS)

Pada pertemuan praktikum ini, akan diperlihatkan salah satu algoritma blind search untuk mencari goal dari initial state yang diberikan. Terdapat 4 komponen yang harus didefinisikan ketika mencari solusi dari sebuah permasalahan.

1. Initial state
2. Goal State
3. Menentukan/menemukan urutan untuk mencapai Goal State
4. Biaya (cost) menemukan solusi.

Kesemua algoritma di atas mempunyai strategi dengan mencari goal yang dimulai dari initial state. Terminologi tree dipilih sebagai salah satu teknik pencarian untuk mencapai goal. Untuk lebih jelas, Gambar 4.1 memperlihatkan contoh kasus penyelesaian permainan 8-puzzle dengan mentransformasikan solusi permainan ke dalam topologi tree.



Gambar 4.1. Topologi Tree untuk permainan 8-puzzle[1]

Kumpulan node-node yang dibentuk tetapi belum disambungkan dengan node yang lain dinamakan dengan *fringe*. Setiap element dari fringe merupakan node left dari tree. Berikut akan dijelaskan algoritma-algoritma yang dikategorikan ke dalam kelas blind search.

- **Breadth First Search (BFS)**: adalah algoritma yang menjelajah node root pertama sekali, kemudian menjelajah semua successor dari node root, kemudian menjelajah semua successor dari successor, dan seterusnya sampai successor yang terakhir. Fringe merupakan struktur data queue First In First Out (FIFO).

- **Uniform Cost Search (UCS)**: merupakan modifikasi dari BFS dengan selalu menjelajah node yang paling sedikit cost-nya pada fringe menggunakan *path cost function g(n)* (misalnya biaya (banyaknya langkah) dari initial state ke node n). Node disusun dengan algoritma queue untuk menentukan jumlah (biaya) untuk mencapai node n.

- **Depth First Search (DFS)**: selalu menjelajah node yang paling dalam pada tree. Fringe merupakan struktur data queue (stack) Last In First Out (LIFO).

- **Depth Limited Search (DLS)**: Kegagalan algoritma DFS dalam menyediakan space (memory) dapat diatasi dengan menentukan terlebih dahulu depth limit l, yaitu node pada depth l diperlakukan seolah-olah mereka tidak memiliki successors.

- **Iterative Deepening Depth First Search (IDS)**: secara umum strategi algoritma ini biasanya digunakan dengan mengkombinasikan algoritma depth first tree search yang mencari *the best depth limit*. Ini dilakukan dengan menambahkan limit dari 0, kemudian 1, kemudian 2, and dan seterusnya sampai goal-nya ditemukan.

[1] (Sumber: http://www.codeproject.com/Articles/203828/AI-Simple-Implementation-of-Uninformed-Search-Stra)

- **Bidirectional Search (BS)**: Ide dari algoritma ini adalah untuk mencari secara bersamaan baik dari goal ke initial state dan dari the initial state ke goal, dan berhenti ketika kedua langkah pencarian bertemu di pertengahan pencarian. Disini, terdapat dua fringe, fringe pertama digunakan untuk langkah dari initial state ke goal *(forward)* dan fringe satu lagi untuk langkah dari goal ke initial state *(backward)*. Setiap fringe diimplementasikan dengan algoritma LIFO atau FIFO tergantung dari strategi pencarian yang digunakan (misalnya Forward=BFS, Backward=DFS).

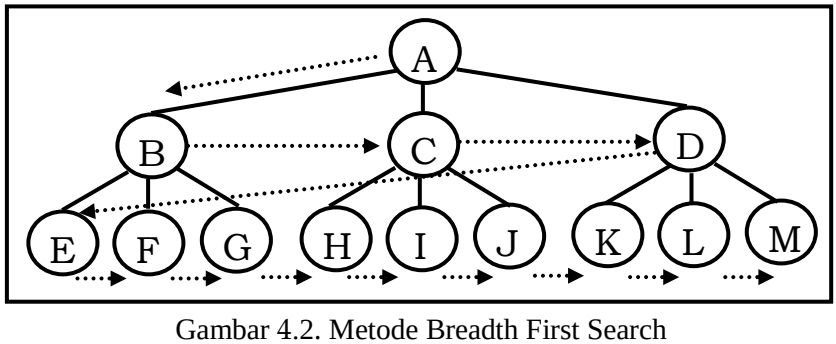
4.2.2. Metode Pencarian Dan Pelacakan

Pada dasarnya ada 2 teknik pencarian dan pelacakan yang digunakan, yaitu pencarian buta (*blind search*) dan pencarian terbimbing (*heuristic search*).

4.2.2.1. Pencarian Buta (*Blind Search*)

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.
- Pencarian dimulai dari node akar terus ke level ke-1 dari kiri ke kanan, kemudian berpindah ke level berikutnya demikian pula dari kiri ke kanan hingga ditemukannya solusi (Gambar 4.2).

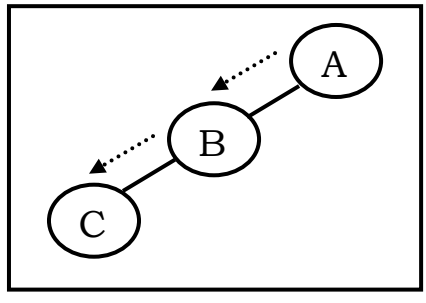


Gambar 4.2. Metode Breadth First Search

- Keuntungan
  a. Tidak akan menemui jalan buntu.
  b. Jika ada satu solusi, maka *breadth-first search* akan menemukannya. Dan jika ada lebih dari satu solusi, maka solusi minimum akan ditemukan.
- Kelemahan
  a. Membutuhkan memori yang cukup banyak, karena menyimpan semua node dalam satu pohon.
  b. Membutuhkan waktu yang cukup lama, karena akan menguji n level untuk mendapatkan solusi pada level yang ke-(n+1).

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

- Pada *Depth-First Search*, proses pencarian akan dilakukan pada semua anaknya sebelum dilakukan pencarian ke node-node yang selevel.
- Pencarian dimulai dari node akar ke level yang lebih tinggi. Proses ini diulangi terus hingga ditemukannya solusi (Gambar 4.2).

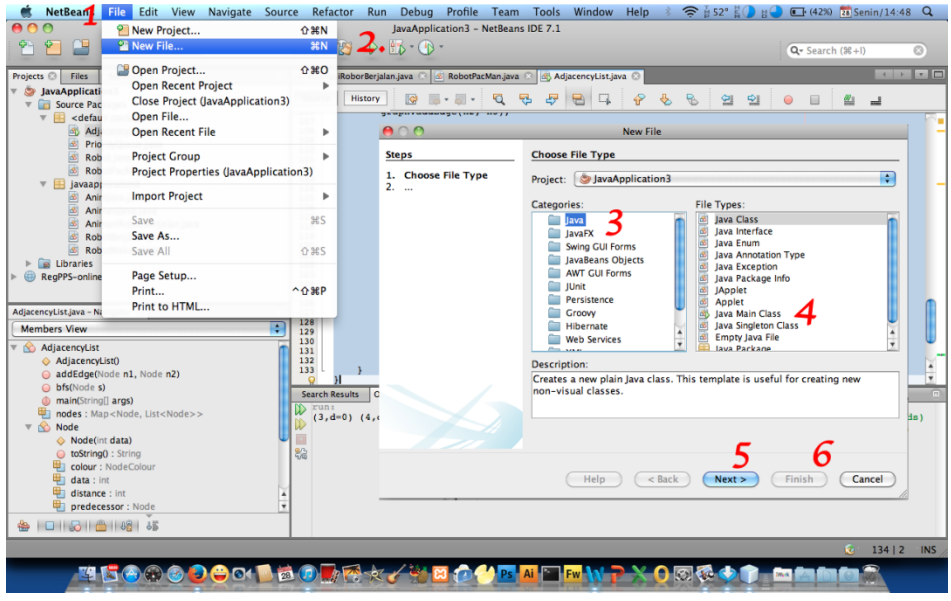


Gambar 4.3. Depth First Search

- Keuntungan
  a. Membutuhkan memori yang relatif kecil, karena hanya node-node pada lintasan yang aktif saja yang disimpan.
  b. Secara kebetulan, metode *depth-first search* akan menemukan solusi tanpa harus menguji labih banyak lagi dalam ruang keadaan.
- Kelemahan
  a. Memungkinkan tidak ditemukannya tujuan yang diharapkan.
  b. Hanya akan mendapatkan 1 solusi pada setiap pencarian.

4.3. Mengimplementasikan Algoritma Bfs.

Ketiklah *source code* Program 3.1 pada perangkat lunak Netbeans 7.0 pada bagian teks editor Java Main Class. Pilih Menu “File”, lalu pilih submenu “New File”. Kemudian pilih Categories “Java” dengan FileTypes-nya adalah “Java Main Class”. Setelah itu, tekan tombol “Next” dan masukkan nama file AdjacencyList, dan terakhir tekan tombol “Finish”. Alur langkah untuk membuat algoritma BFS dapat diikuti melalui Gambar 4.3



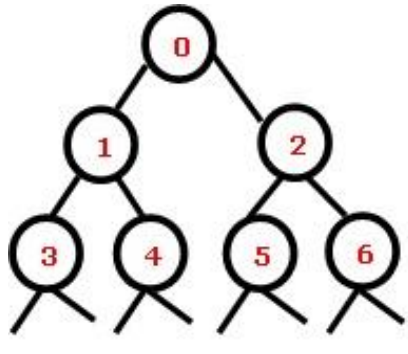
Gambar 4.4. Teks Editor Netbeans 7.0

Kode berikut merupakan urutan angka dari 0, 1, 2, .N. Initial state-nya adalah 0 dan goal state-nya adalah angka yang ditetapkan oleh user. Setiap langkah dibangkitkan secara acak atau 1. *Method* `addEdge(Node n1, Node n2)` adalah method untuk menghubungkan dua buah edge sedangkan *method* `bfs(Node s)` adalah method untuk menentukan teknik pencarian menggunakan algoritma Breadth First Search..

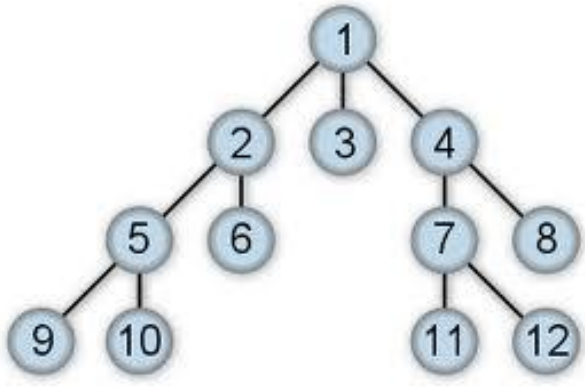
```
import java.util.ArrayDeque;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Queue;
import java.util.Set;

/**
 * Graph represented by an adjacency list.
 *
 * Reference: Introduction to Algorithms - CLRS.
 *
 * @author
 * @since: 26/02/2011
 */

public class AdjacencyList
{
    public enum NodeColour { WHITE, GRAY, BLACK }

    public static class Node
    {
        int data;
        int distance;
        Node predecessor;
        NodeColour colour;

        public Node(int data)
```

```
    {
        this.data = data;
    }

    public String toString()
    {
     return "(" + data + ",d=" + distance + ")";
    }
}

Map<Node, List<Node>> nodes;

public AdjacencyList()
{
    nodes = new HashMap<Node, List<Node>>();
}

public void addEdge(Node n1, Node n2)
{
    if (nodes.containsKey(n1)) {
        nodes.get(n1).add(n2);
    } else {
   ArrayList<Node> list = new ArrayList<Node>();
        list.add(n2);
        nodes.put(n1, list);
    }
}

public void bfs(Node s)
{
    Set<Node> keys = nodes.keySet();
    for (Node u : keys) {
        if (u != s) {
            u.colour = NodeColour.WHITE;
            u.distance = Integer.MAX_VALUE;
            u.predecessor = null;
        }
    }
    s.colour = NodeColour.GRAY;
    s.distance = 0;
    s.predecessor = null;
    Queue<Node> q = new ArrayDeque<Node>();
    q.add(s);
    while (!q.isEmpty()) {
        Node u = q.remove();
        List<Node> adj_u = nodes.get(u);
        if (adj_u != null) {
            for (Node v : adj_u) {
                if(v.colour == NodeColour.WHITE) {
                    v.colour = NodeColour.GRAY;
                    v.distance = u.distance + 1;
                    v.predecessor = u;
                    q.add(v);
                }
            }
        }
        u.colour = NodeColour.BLACK;
        System.out.print(u + " ");
    }
}
public static void main(String[] args)
```

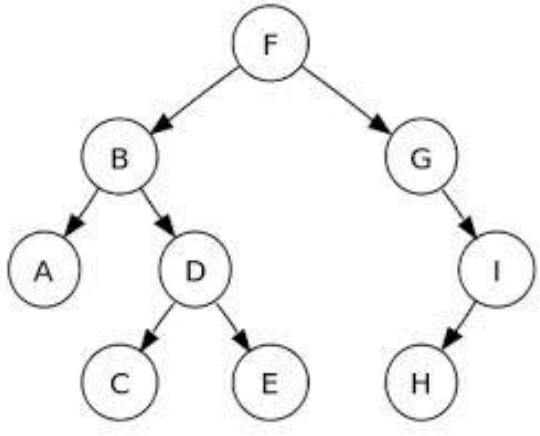
```
    {
        AdjacencyList graph = new AdjacencyList();
        Node n1 = new Node(1);
        Node n2 = new Node(2);
        Node n3 = new Node(3);
        Node n4 = new Node(4);
        Node n5 = new Node(5);
        Node n6 = new Node(6);
        Node n7 = new Node(7);
        Node n8 = new Node(8);

        graph.addEdge(n1, n2);

        graph.addEdge(n2, n1);
        graph.addEdge(n2, n3);

        graph.addEdge(n3, n4);
        graph.addEdge(n3, n2);

        graph.addEdge(n4, n3);
        graph.addEdge(n4, n5);
        graph.addEdge(n4, n6);

        graph.addEdge(n5, n4);
        graph.addEdge(n5, n6);
        graph.addEdge(n5, n7);

        graph.addEdge(n6, n4);
        graph.addEdge(n6, n5);
        graph.addEdge(n6, n7);
        graph.addEdge(n6, n8);

        graph.addEdge(n7, n5);
        graph.addEdge(n7, n6);
        graph.addEdge(n7, n8);

        graph.addEdge(n8, n6);
        graph.addEdge(n8, n7);

        graph.bfs(n3);

    }
}
```



Gambar 4.5. Tree 1



Gambar 4.6. Tree 2



Gambar 4.7. Tree 3

Tugas:

1. Tentukan bagaimana algoritma BFS di atas dapat menentukan node ke 8, 6, dan 7.
2. Ubahlah method static void main sehingga bentuk tree seperti Gambar 4.4 dapat dibentuk. Kemudian tentukan bagaimana algoritma BFS dapat menemukan node 5.
3. Ubahlah method static void main sehingga bentuk tree seperti Gambar 4.5 dapat dibentuk. Kemudian tentukan bagaimana algoritma BFS dapat menemukan node 9.

Ubahlah kode program di atas sehingga bentuk tree seperti Gambar 6 dapat dibentuk. Kemudian tentukan bagaimana algoritma BFS dapat menemukan node C.

MODUL 5
TEKNIK HEURISTIC SEARCH

5.1. Tujuan

Memperlihatkan kepada mahasiswa bagaimana menyelesaikan permasalahan pada game 8-puzzle dengan menggunakan algoritma heuristic search. Mahasiswa diharapkan mampu mengimplementasikan algoritma heuristic dengan menggunakan Java.

5.2. Dasar Teori

5.2.1. Teknik Pencarian Heuristic Search

Teknik blind search tidak selalu memecahkan masalah dengan baik. Waktu yang dibutuhkan ketika menemukan solusi atau memecahkan masalah terlalu lama dan juga memori yang dibutuhkan untuk menampung urutan-urutan solusi sangat besar akan menjadi kelemahan bagi algoritma ini. Kelemahan tersebut dapat diatasi ketika informasi-informasi tambahan yang diperoleh dari setiap langkah pencarian diidentifikasikan dan dijadikan sebagai penentu langkah berikutnya.

Salah satu teknik untuk meminimalisasikan kelemahan dari blind search adalah teknik heuristic. Heuristic merupakan suatu proses dimana pencarian solusi akan ditemukan dengan baik namun bisa juga kemungkinan tidak ada solusi. Teknik ini mmebutuhkan sebuah nilai untuk menentukan pencarian berikutnya. Nilai heuristic dapat ditentukan melalui fungsi heuristic.

Fungsi heuristic merupakan fungsi yang melakukan pemetaan dari diskripsi keadaan ke pengukur kebutuhan. Umumnya fungsi ini direpresentasikan ke dalam bentuk angka. Dalam ilmu Kecerdasan Buatan, heuristic dihadapkan dalam 2 keadaan dasar.

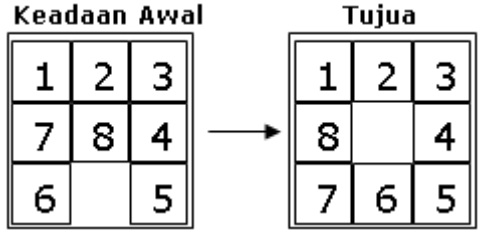
- Persoalan/problema yang mungkin memiliki solusi eksak, namun biaya perhitungan untuk menemukan solusi tersebut sangat tinggi dalam kebanyakan persoalan (seperti catur), ruang keadaan bertambah secara luar biasa seiring dengan jumlah.
- Persoalan yang mungkin tidak memiliki solusi eksak karena ambiquitas (ketidakpastian) mendasar dalam pernyataan persoalan atau data yang tersedia diagnosa medis merupakan salah satu contohnya.

Heuristi hanyalah sebuah cara menerka langkah berikutnya yang harus diambil dalam memecahkan suatu persoalan berdasarkan informasi yang ada/tersedia.

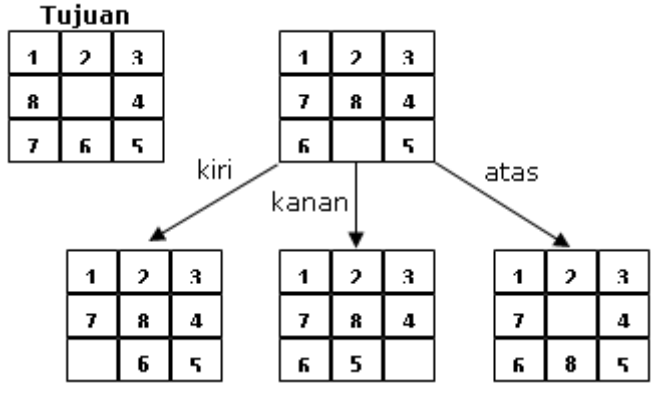
Pencarian Heuristik (*Heuristic Search*)

- Pencarian buta tidak selalu dapat diterapkan dengan baik, hal ini disebabkan waktu aksesnya yang cukup lama serta besarnya memori yang diperlukan.

- Kelemahan ini sebenarnya dapat diatasi jika ada informasi tambahan dari domain yang bersangkutan.
- Misalkan pada kasus 8-puzzle (Gambar 5.1)

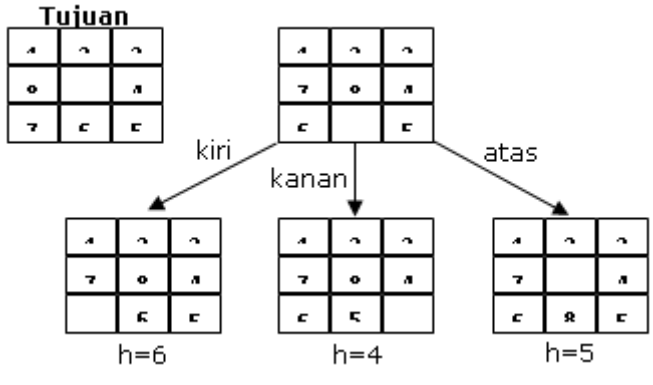


Gambar 5.1. Kasus 8-puzzle



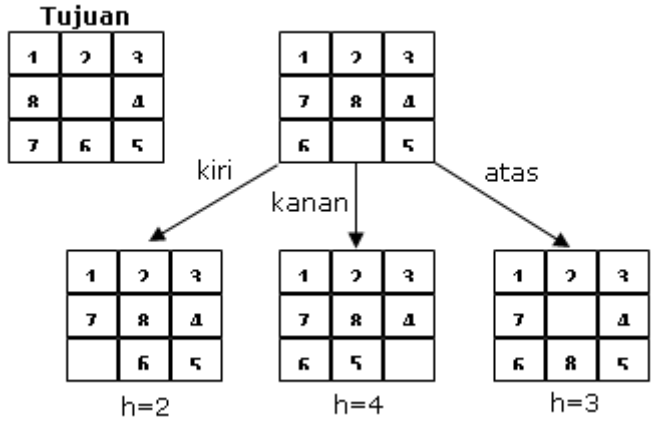
Gambar 5.2. Langkah awal kasus 8-puzzle

- Langkah pertama dari permainan tersebut seperti terlihat pada Gambar 5.2. Apabila digunakan pencarian buta, kita tidak perlu mengetahui operasi apa yang akan dikerjakan (sembarang operasi bisa digunakan).
- Pada pencarian heuristik perlu diberikan informasi khusus dalam domain tersebut.
- Informasi yang bisa diberikan, antara lain:
  a. Untuk jumlah ubin yang menempati posisi yang benar: jumlah yang lebih tinggi adalah yang lebih diharapkan (lebih baik), Gambar 5.3.



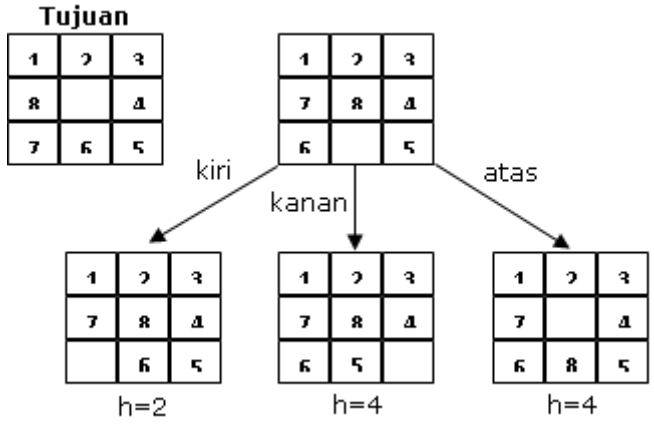
Gambar 5.3. Fungsi heuristik pertama kasus 8-puzzle

b. Untuk jumlah ubin yang menempati posisi yang salah: jumlah yang lebih kecil adalah yang diharapkan (lebih baik), Gambar 5.4.



Gambar 5.4. Fungsi heuristik kedua kasus 8-puzzle

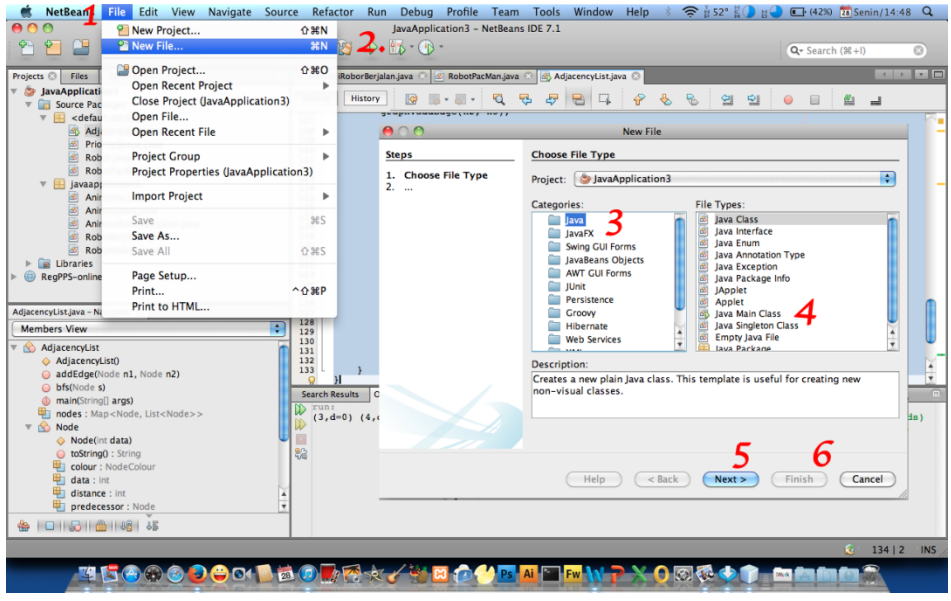
c. Menghitung total gerakan yang diperlukan untuk mencapai tujuan; jumlah yang lebih kecil adalah yang diharapkan (lebih baik), Gambar 5.5.



Gambar 5.5. Fungsi heuristik ketiga kasus 8-puzzle

5.3. Implementasi Algoritma Heurstic pada Permainan 8-Puzzle

Ketiklah *source code* Program 5.1 dan 5.2. pada perangkat lunak Netbeans 7.0 pada bagian teks editor Java Main Class. Pilih Menu "File", lalu pilih submenu "New File". Kemudian pilih Categories "Java" dengan FileTypes-nya adalah "Java Main Class". Setelah itu, tekan tombol "Next" dan masukkan nama file EightPuzzleSearch, dan terakhir tekan tombol "Finish". Alur langkah untuk membuat algoritma Heuristic dapat diikuti melalui Gambar 5.6



Gambar 5.6. Teks Editor Netbeans 7.0

File EightPuzzleSearch.java berisikan dua buah class yaitu class EightPuzzleSearch dan Node. Di dalam class Node memasukkan semua node ke dalam sebuah tree. Class ini juga dapat menghasilkan alur (path) dari root ke node tertentu yang diinginkan. Pemanggilan path ini dapat dilakukan melalui pemanggilan method getPath(). Berikut ini adalah pendeklarasian class Node.

```
class Node {
     int[] state = new int[9];
     int cost;
     Node parent = null;
     Vector<Node> successors = new Vector<Node>();

     Node(int s[], Node parent) {
        this.parent = parent;
        for (int i = 0; i < 9; i++) state[i] = s[i];
     }

     public String toString() {
        String s = "";
        for (int i = 0; i < 9; i++) {
           s = s + state[i] + " "; }
        return s;
     }

     public boolean equals(Object node) {
        Node n = (Node)node;
        boolean result = true;
        for (int i = 0; i < 9; i++) {
           if (n.state[i] != state[i]) result = false; }
        return result;
     }

     Vector<Node> getPath(Vector<Node> v) {
        v.insertElementAt(this, 0);
        if (parent != null) v = parent.getPath(v);
        return v;
```

```
     }

     Vector<Node> getPath() {
        return getPath(new Vector<Node>());
     }
}
```

Program 5.1. Pendeklarasian Class Node.

Constructor Node menerima dua buah parameter yaitu urutan node children dan root dari children tersebut. Semua children dan node-node yang ada di dalam tree didefinisikan ke dalam tipe data integer (int). Class ini juga mempunyai method toString() yang berfungsi untuk mengubah node (dalam tipe data int) ke dalam bentuk string sehingga hasilnya akan berbentuk alur (path) dari initial state ke goal state dengan memanggil method *getPath()*.

Class EightPuzzleSearch memanggil fungsi utama *(main function)*, mendeskripsikan algoritma heuristic, menghitung cost *heuristic*, mencetak alur (path) dari root ke suatu node, dan menentukan node terbaik berdasarkan nilai dari fungsi heuristic.

```
public class EightPuzzleSearch {
     EightPuzzleSpace space = new EightPuzzleSpace();

     Vector<Node> open = new Vector<Node>();
     Vector<Node> closed = new Vector<Node>();

     int h1Cost(Node node) {
        int cost = 0;
        for (int i = 0; i < node.state.length; i++) {
           if (node.state[i] != i) cost++; }
        return cost;
     }
     int h2Cost(Node node) {
        int cost = 0;
        int state[] = node.state;
        for (int i = 0; i < state.length; i++) {
            int v0 = i, v1 = state[i];
            /*tidak menghitung ubin yang kosong */
            if (v1 == 0) continue;
            int row0 = v0 / 3, col0 = v0 % 3, row1 = v1 / 3, col1 = v1 % 3;
            int c=(Math.abs(row0-row1)+Math.abs(col0-col1));
            cost += c;
        }
        return cost;
     }

/*boleh diubah dengan memakai heuristic h1 atau h2 */
     int hCost(Node node) {
        return h2Cost(node);
     }

    Node getBestNode(Vector nodes) {
        int index = 0, minCost = Integer.MAX_VALUE;
        for (int i = 0; i < nodes.size(); i++) {
            Node node = (Node)nodes.elementAt(i);
            if (node.cost < minCost) {
            minCost = node.cost;
            index = i; } }
        Node bestNode = (Node)nodes.remove(index);
        return(bestNode);
    }
```

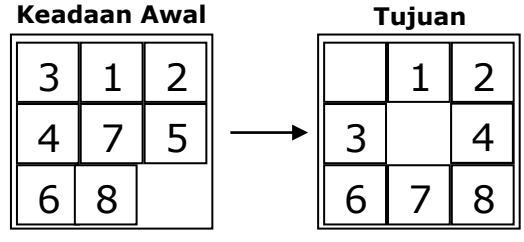
```
int getPreviousCost(Node node) {
    int i = open.indexOf(node);
    int cost = Integer.MAX_VALUE;
    if (i != -1) {
        cost = open.get(i).cost; }
    else {
        i = closed.indexOf(node);
        if (i != -1) cost = closed.get(i).cost; }
    return(cost);
}

void printPath(Vector path) {
    for (int i = 0; i < path.size(); i++) {
     System.out.print(" " + path.elementAt(i) + "\n"); }
}

void run() {
    Node root = space.getRoot();
    Node goal = space.getGoal();
    Node solution = null;
    open.add(root);
    System.out.print("\nRoot: " + root + "\n\n");
    while (open.size() > 0) {
        Node node = getBestNode(open);
        int pathLength = node.getPath().size();
        closed.add(node);
        if (node.equals(goal)) {
              solution = node;
              break;
        }
        Vector<Node> successors =
                          space.getSuccessors(node);
        for (int i = 0; i < successors.size(); i++) {
            Node successor = successors.get(i);
            int cost = hCost(successor)+pathLength+1;
            int previousCost;
            previousCost = getPreviousCost(successor);
            boolean inClosed;
            inClosed = closed.contains(successor);
            boolean inOpen = open.contains(successor);


        if(!(inClosed||inOpen)||cost<previousCost)
        {
                if(inClosed) closed.remove(successor);
                if (!inOpen) open.add(successor);
                successor.cost = cost;
                successor.parent = node;
            }
        }
    }
    // new TreePrint(getTree(root));
    if (solution != null) {
       Vector path = solution.getPath();
       System.out.print("\nSolution found\n");
       printPath(path);
    }
 }

public static void main(String args[]) {
  // melakukan pencarian
```

```
        new EightPuzzleSearch().run();
    }
}
```

Program 5.2. Pendeklarasian Class EightPuzzleSearch.

Program 5.2 membentuk sebuah object space dengan bertipe class `EightPuzzleSpace`. Objek ini berfungsi untuk mendefinisikan initial state dan goal state sehingga penggua (user) dengan mudah menentukan contoh initial dan goal state pada 8-puzzle. Sebagai contoh pada kasus ini initial dan goal state dapat dilihat seperti Gambar 5.7. Bentuk initial dan goal state direpresentasikan ke dalam bentuk array berdimensi satu dan dideklarasikan seperti dibawah ini (lihat method *getRoot()* dan *getGoal()*).

```
int ex[] = {3, 1, 2, 4, 7, 5, 6, 8, 0};  // initial state
int state[] = {0, 1, 2, 3, 4, 5, 6, 7, 8}; // goal state
```



Gambar 5.7. Initial dan Goal state pada 8-puzzle

Kode lengkap dari class ini dapat dilihat pada Program 5.3. Ketiklah *source code* Program 5.3. pada perangkat lunak Netbeans 7.0 pada bagian teks editor Java Class. Pilih Menu “File”, lalu pilih submenu “New File”. Kemudian pilih Categories “Java” dengan FileTypes-nya adalah “Java Class”. Setelah itu, tekan tombol “Next” dan masukkan nama file EightPuzzleSpace, dan terakhir tekan tombol “Finish”.

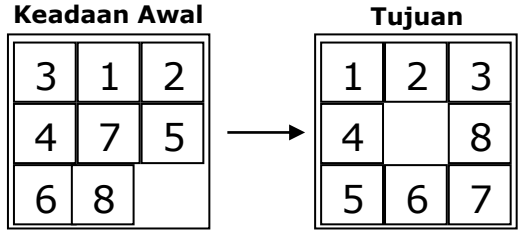
```
import java.util.Vector;
/*
 * Class EightPuzzleSpace dideklarasikan untuk menentukan
 * initial dan goal state serta mendapatkan path dari root
 * ke node tertentu
 */
/*
 * Modified by Irvanizam Zamanhuri
 */

public class EightPuzzleSpace {

    Node getRoot() {
        int ex[] = {3, 1, 2, 4, 7, 5, 6, 8, 0};
        // the Russell and Norvig eg
        int rn[] = {7, 2, 4, 5, 0, 6, 8, 3, 1};
        return new Node(ex, null);
    }

    Node getGoal() {
        int state[] = {0, 1, 2, 3, 4, 5, 6, 7, 8};
        return new Node(state, null);
    }
```

```
    Vector<Node> getSuccessors(Node parent) {
        Vector<Node> successors = new Vector<Node>();
        for (int r = 0; r < 3; r++) {
           for (int c = 0; c < 3; c++) {
              /* ubin kosong disini */
              if (parent.state[(r * 3) + c] == 0) {
                 /* memindahkan ubin ke kiri */
                 if (r > 0) {
                    successors.add(transformState(r-1, c, r, c, parent)); }
                 /* memindahkan ubin ke kanan */
                 if (r < 2) {
                    successors.add(transformState(r+1, c, r, c, parent)); }
                  /* memindahkan ubin dari bawah */
                 if (c > 0) {
                    successors.add(transformState(r, c-1, r, c, parent)); }
                  /* memindahkan ubin dari atas */
                 if (c < 2) {
                    successors.add(transformState(r, c+1, r, c, parent)); }
              }
           }
        }
       /* used in getTree */
        parent.successors = successors;
        return successors;
    }

    Node transformState(int r0, int c0, int r1, int c1, Node parent) {
        int[] s = parent.state;
        int[] newState = {s[0], s[1], s[2], s[3], s[4], s[5], s[6], s[7],
s[8]};
        newState[(r1 * 3) + c1] = s[(r0 * 3) + c0];
        newState[(r0 * 3) + c0] = 0;
        return new Node(newState, parent);
     }
}
```

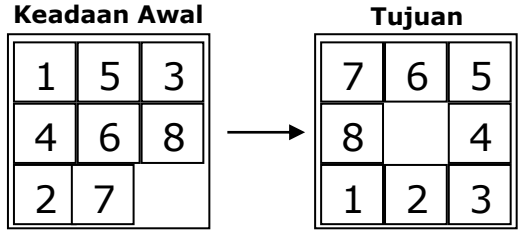
Program 5.3. Pendeklarasian Class EightPuzzleSpace.

Tugas:

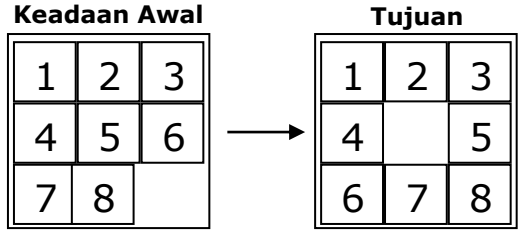
1. Pelajari class EightPuzzleSearch, EightPuzzleSpace, dan Node.
2. Ubahlah initial dan goal state dari program di atas sehingga bentuk initial dan goal statenya Gambar 8. Kemudian tentukan langkah-langkah mana saja sehingga puzzlenya mencapai goal state. Analisa dan bedakan dengan solusi pada point 1.
3. Ubahlah initial dan goal state dari program di atas sehingga bentuk initial dan goal statenya Gambar 5.9. Kemudian tentukan langkah-langkah mana saja sehingga puzzlenya mencapai goal state. Analisa dan bedakan dengan solusi pada point 1 dan 2.
4. Ubahlah initial dan goal state dari program di atas sehingga bentuk initial dan goal statenya Gambar 5.10. Kemudian tentukan langkah-langkah mana saja sehingga puzzlenya mencapai goal state. Analisa dan bedakan dengan solusi pada point 1, 2, dan 3.
5. Ubahlah initial dan goal state dari program dan class-class di atas sehingga bentuk initial dan goal statenya Gambar 5.11. Kemudian tentukan langkah-langkah mana saja sehingga puzzlenya mencapai goal state.



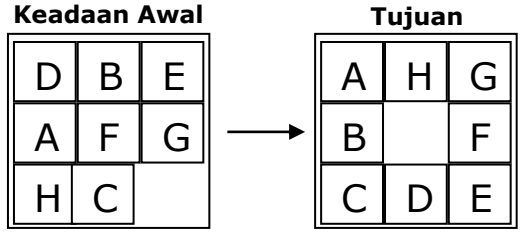
Gambar 5.8. Initial dan Goal state pada 8-puzzle 2



Gambar 5.9. Initial dan Goal state pada 8-puzzle 3



Gambar 5.10. Initial dan Goal state pada 8-puzzle 4



Gambar 5.11. Initial dan Goal state pada 8-puzzle 5

MODUL 6
PERMAINAN TIC TAC TOE

6.1. Tujuan

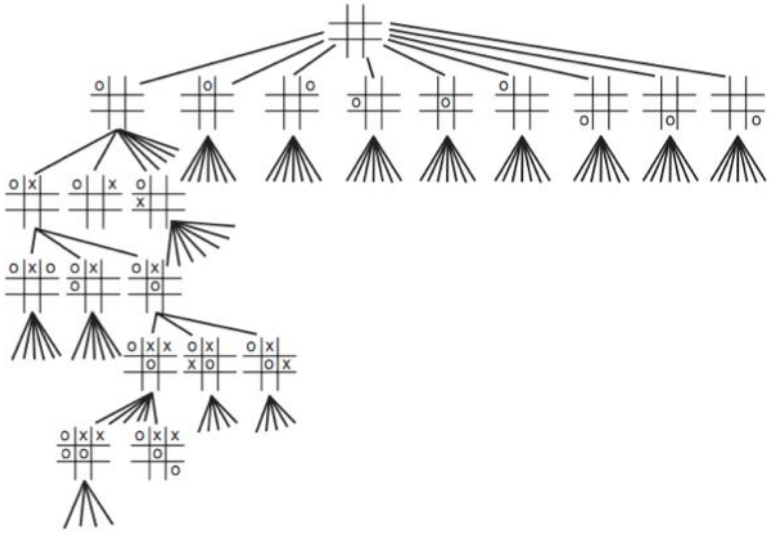
Meningkatkan pemahaman mahasiswa terhadap code permainan tic tac toe. Selain itu, modul 6 memberikan pengetahuan tentang Object Oriented Programming menggunakan bahasa pemrograman Java terutama Java Swing dan Japplet.

6.2. Dasar Teori

6.2.1. Permainan Tic Tac Toe

Penyelesaian masalah permainan tic tac toe dapat menggunakan algoritma heuristic untuk mencapai solusi yang optimal. Pada modul ini memperlihatkan bagaimana membuat sebuah permainan tic tac toe. Initial state dari permainan ini adalah puzzle ukuran 8 yang tidak berisikan apa-apa. Ketika pemain pertama menekan salah satu ubin, maka ubin tersebut akan diberikan tanda silang. Pemain kedua harus menghalangi pemain pertama untuk membuat tanda silang berjajaran baik secara vertikal, horizontal, atau diagonal. Permainan ini akan berakhir (goal state) ketika salah seorang pemain sudah menderetkan tanda meraka masing-masing baik secara vertikal, horizontal, atau diagonal.

Solusi dari permasalahan ini dapat dilakukan dengan membuat topologi Tree, kemudian setiap langkah dari pemain pertama atau kedua akan menjadikan initial state selanjutnya, kemudian langkah tersebut akan dijadikan sebagai initial state yang baru sampai menemukan goal statenya. Ilustrasi penyelesaian masalah permainan tic tac toe ini dapat dilihat melalui Gambar 6.1.



Gambar 6.1. Pohon Permainan Tic Tac Toe

6.3. Implementasi Permainan Tic Tac Toe

6.3.1. Menggunakan Graphical User Interface (GUI)

Permainan ini dibuatkan dengan menggunakan Java Swing yang terdiri dari 3 buah Java Class dan 3 buah pendeklarasian enumaration. Ke-enam file tersebut dituliskan secara terpisah namun disimpan pada folder yang sama (misalnya folder tic tac toe). Ke-enam file tersebut dituliskan nama secara berurutan seperti berikut:

1. State.Java
2. Seed.Java
3. GameState.Java
4. Cell.Java
5. Board.Java
6. GameMain.Java

Pemberian sebuah nilai integer kepada variabel untuk membedakan status penggunaan (misalnya jika tic tac toe sedang dimainkan, variable PLAYING diberikan nilai 0, variable DRAW = 1, dan sebagainya) tidak begitu efektif di dalam penulisan code. Sekarang, JDK1.5 memperkenalkan fitur baru yang dinamakan dengan *enumaration*, yang merupakan class spesial untuk menyimpan semua variabel secara berurutan. Enumaration State, Seed, dan GameState didefinisikan secara file terpisah seperti di bawah ini.

```
package TicTacToe;
/**  *  Enumeration for the various states of the game  */ public enum
GameState {  // to save as "GameState.java"
        PLAYING, DRAW, CROSS_WON, NOUGHT_WON
}
```

```
package TicTacToe;
/**  * Enumeration for the seeds and cell contents  */ public enum Seed {
// to save as "Seed.java"
        EMPTY, CROSS, NOUGHT
}
```

```
package TicTacToe;
/**  *  Enumeration for the various states of the game  */ public enum
State {  // to save as "GameState.java"
        PLAYING, DRAW, CROSS_WON, NOUGHT_WON
}
```

Kemudian diketikkan *code* untuk class Cell, Board, dan GameMain secara terpisah. Program ketiga class tersebut dapat dilihat seperti di bawah ini.

Class Cell.java

Code ini dikutip dari:

http://www3.ntu.edu.sg/home/ehchua/programming/java/JavaGame_TicTacToe.html

```
package TicTacToe;

import java.awt.Graphics;
import java.awt.*;
import java.awt.Graphics2D;

public class Cell {
      //content of this cell (Seed.EMPTY, Seed.CROSS, or Seed.NOUGHT)
      Seed content;
      int row, col; // row and column of this cell

 /**Constructor to initialize this cell with the specified row and col */
   public Cell(int row, int col) {
      this.row = row;
      this.col = col;
      clear(); // clear content
   }

   /** Clear this cell's content to EMPTY */
   public void clear() {
      content = Seed.EMPTY;
   }

   /**Paint itself on the graphics canvas, given the Graphics context */
   public void paint(Graphics g) {
      // Use Graphics2D which allows us to set the pen's stroke
      Graphics2D g2d = (Graphics2D)g;
      g2d.setStroke(new BasicStroke(GameMain.SYMBOL_STROKE_WIDTH,
            BasicStroke.CAP_ROUND,  BasicStroke.JOIN_ROUND));  //  Graphics2D
only
      // Draw the Seed if it is not empty
      int x1 = col * GameMain.CELL_SIZE + GameMain.CELL_PADDING;
      int y1 = row * GameMain.CELL_SIZE + GameMain.CELL_PADDING;
      if (content == Seed.CROSS) {
         g2d.setColor(Color.RED);
         int x2 = (col + 1) * GameMain.CELL_SIZE - GameMain.CELL_PADDING;
         int y2 = (row + 1) * GameMain.CELL_SIZE - GameMain.CELL_PADDING;
         g2d.drawLine(x1, y1, x2, y2);
         g2d.drawLine(x2, y1, x1, y2);
      } else if (content == Seed.NOUGHT) {
         g2d.setColor(Color.BLUE);
         g2d.drawOval(x1, y1, GameMain.SYMBOL_SIZE, GameMain.SYMBOL_SIZE);
      }
   }
}
```

Kemudian tuliskan code program untuk Class Board.java untuk membuat dan mengatur papan permainan tic tac toe.

Class Board.java

```
package TicTacToe;
import java.awt.*;

/**
 * The Board class models the ROWS-by-COLS game-board.
 */
public class Board {
   // package access
   // composes of 2D array of ROWS-by-COLS Cell instances
   Cell[][] cells;

   /** Constructor to initialize the game board */
   public Board() {

      // allocate the array
      cells = new Cell[GameMain.ROWS][GameMain.COLS];
      for (int row = 0; row < GameMain.ROWS; ++row) {
         for (int col = 0; col < GameMain.COLS; ++col) {
            // allocate element of array
            cells[row][col] = new Cell(row, col);
         }

     }
   }

   /** Initialize (or re-initialize) the game board */
   public void init() {
      for (int row = 0; row < GameMain.ROWS; ++row) {
         for (int col = 0; col < GameMain.COLS; ++col) {
            // clear the cell content
            cells[row][col].clear();
         }
      }
   }

   /** Return true if it is a draw (i.e., no more EMPTY cell) */
   public boolean isDraw() {
      for (int row = 0; row < GameMain.ROWS; ++row) {
         for (int col = 0; col < GameMain.COLS; ++col) {
            if (cells[row][col].content == Seed.EMPTY) {
              // an empty seed found, not a draw, exit
               return false;
            }
         }
      }
      return true; // no empty cell, it's a draw
   }

   /** Return true if the player with "seed" has won after placing at
      (seedRow, seedCol) */
   public boolean hasWon(Seed seed, int seedRow, int seedCol) {
      return (cells[seedRow][0].content == seed   // 3-in-the-row
                  && cells[seedRow][1].content == seed
                  && cells[seedRow][2].content == seed
               || cells[0][seedCol].content == seed // 3-in-the-column
                  && cells[1][seedCol].content == seed
```

```
                && cells[2][seedCol].content == seed
            || seedRow == seedCol              // 3-in-the-diagonal
                && cells[0][0].content == seed
                && cells[1][1].content == seed
                && cells[2][2].content == seed
            || seedRow + seedCol == 2          // 3-in-the-opposite-diagonal
                && cells[0][2].content == seed
                && cells[1][1].content == seed
                && cells[2][0].content == seed);
   }

   /** Paint itself on the graphics canvas, given the Graphics context */
   public void paint(Graphics g) {
      // Draw the grid-lines
      g.setColor(Color.GRAY);
      for (int row = 1; row < GameMain.ROWS; ++row) {
         g.fillRoundRect(0,      GameMain.CELL_SIZE      *      row      -
GameMain.GRID_WIDHT_HALF,
               GameMain.CANVAS_WIDTH-1, GameMain.GRID_WIDTH,
               GameMain.GRID_WIDTH, GameMain.GRID_WIDTH);
      }
      for (int col = 1; col < GameMain.COLS; ++col) {
         g.fillRoundRect(GameMain.CELL_SIZE         *         col         -
GameMain.GRID_WIDHT_HALF, 0,
               GameMain.GRID_WIDTH, GameMain.CANVAS_HEIGHT - 1,
               GameMain.GRID_WIDTH, GameMain.GRID_WIDTH);
      }

      // Draw all the cells
      for (int row = 0; row < GameMain.ROWS; ++row) {
         for (int col = 0; col < GameMain.COLS; ++col) {
            cells[row][col].paint(g);  // ask the cell to paint itself
         }
      }
   }
}
```

Kemudian tuliskan code program untuk Class GameMain.java untuk menjalankan permainan tic tac toe.

Class GameMain.java

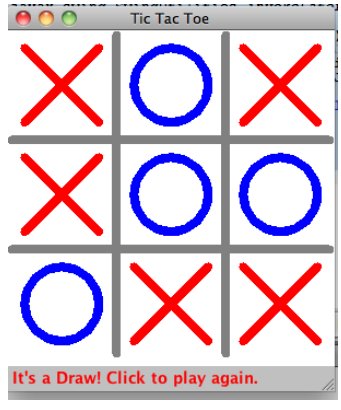
```
package TicTacToe;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/**
 * Tic-Tac-Toe: Two-player Graphic version with better OO design.
 * The Board and Cell classes are separated in their own classes.
 */
@SuppressWarnings("serial")
public class GameMain extends JPanel {
   // Named-constants for the game board
   public static final int ROWS = 3;  // ROWS by COLS cells
   public static final int COLS = 3;
```

```
   public static final String TITLE = "Tic Tac Toe";

   // Name-constants for the various dimensions used for graphics drawing
   public static final int CELL_SIZE = 100; // cell width and height (square)
   public static final int CANVAS_WIDTH = CELL_SIZE * COLS;   // the drawing
canvas
   public static final int CANVAS_HEIGHT = CELL_SIZE * ROWS;
   public static final int GRID_WIDTH = 8;  // Grid-line's width
   public static final int GRID_WIDHT_HALF = GRID_WIDTH / 2; // Grid-line's
half-width
   // Symbols (cross/nought) are displayed inside a cell, with padding from border

   public static final int CELL_PADDING = CELL_SIZE / 6;
   public static final int SYMBOL_SIZE = CELL_SIZE - CELL_PADDING * 2;
   public static final int SYMBOL_STROKE_WIDTH = 8; // pen's stroke width

   private Board board;             // the game board
   private GameState currentState;//the current state of the game
   private Seed currentPlayer;      // the current player
   private JLabel statusBar;     // for displaying status message

   /** Constructor to setup the UI and game components */
   public GameMain() {

      // This JPanel fires MouseEvent
      this.addMouseListener(new MouseAdapter() {
         @Override
         public void mouseClicked(MouseEvent e) {  // mouse-clicked handler
            int mouseX = e.getX();
            int mouseY = e.getY();
          // Get the row and column clicked
            int rowSelected = mouseY / CELL_SIZE;
            int colSelected = mouseX / CELL_SIZE;

            if (currentState == GameState.PLAYING) {
               if (rowSelected >= 0 && rowSelected < ROWS
                     && colSelected >= 0 && colSelected < COLS
                     &&   board.cells[rowSelected][colSelected].content   ==
Seed.EMPTY) {
                  board.cells[rowSelected][colSelected].content           =
currentPlayer; // move
                  updateGame(currentPlayer,  rowSelected,  colSelected);  //
update currentState
               // Switch player
                  currentPlayer   =   (currentPlayer   ==   Seed.CROSS)   ?
Seed.NOUGHT : Seed.CROSS;
                }
            } else {         // game over
               initGame();  // restart the game
            }
            // Refresh the drawing canvas
            repaint();  // Call-back paintComponent().
         }
      });

      // Setup the status bar (JLabel) to display status message
      statusBar = new JLabel("         ");
      statusBar.setFont(new Font(Font.DIALOG_INPUT, Font.BOLD, 14));
      statusBar.setBorder(BorderFactory.createEmptyBorder(2, 5, 4, 5));
      statusBar.setOpaque(true);
      statusBar.setBackground(Color.LIGHT_GRAY);

      setLayout(new BorderLayout());
```

```
      add(statusBar, BorderLayout.PAGE_END); // same as SOUTH
      setPreferredSize(new Dimension(CANVAS_WIDTH, CANVAS_HEIGHT + 30));
            // account for statusBar in height

      board = new Board();    // allocate the game-board
      initGame();  // Initialize the game variables
   }

   /** Initialize the game-board contents and the current-state */
   public void initGame() {
      for (int row = 0; row < ROWS; ++row) {
         for (int col = 0; col < COLS; ++col) {
            board.cells[row][col].content = Seed.EMPTY; // all cells empty
         }
      }
      currentState = GameState.PLAYING;  // ready to play
      currentPlayer = Seed.CROSS;         // cross plays first
   }

   /** Update the currentState after the player with "theSeed" has placed on (row, col) */
   public void updateGame(Seed theSeed, int row, int col) {
      if (board.hasWon(theSeed, row, col)) {  // check for win
         currentState = (theSeed == Seed.CROSS) ? GameState.CROSS_WON :
GameState.NOUGHT_WON;
      } else if (board.isDraw()) {  // check for draw
         currentState = GameState.DRAW;
      }
     // Otherwise, no change to current state (PLAYING).
   }

  /** Custom painting codes on this JPanel */
   @Override
   public void paintComponent(Graphics g){
      //invoke via repaint()
      super.paintComponent(g);     // fill background
      setBackground(Color.WHITE); // set its background color

      board.paint(g);  // ask the game board to paint itself

     // Print status-bar message
      if (currentState == GameState.PLAYING) {
         statusBar.setForeground(Color.BLACK);
         if (currentPlayer == Seed.CROSS) {
            statusBar.setText("X's Turn");
         } else {
            statusBar.setText("O's Turn");
         }
      } else if (currentState == GameState.DRAW) {
         statusBar.setForeground(Color.RED);
         statusBar.setText("It's a Draw! Click to play again.");
      } else if (currentState == GameState.CROSS_WON) {
         statusBar.setForeground(Color.RED);
         statusBar.setText("'X' Won! Click to play again.");
      } else if (currentState == GameState.NOUGHT_WON) {
         statusBar.setForeground(Color.RED);
         statusBar.setText("'O' Won! Click to play again.");
      }
   }

  /** The entry "main" method */
   public static void main(String[] args) {
      // Run GUI construction codes in Event-Dispatching thread for thread safety
      javax.swing.SwingUtilities.invokeLater(new Runnable() {
```

```
            public void run() {
                JFrame frame = new JFrame(TITLE);
        // Set the content-pane of the JFrame to an instance of main JPanel
                frame.setContentPane(new GameMain());
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.pack();
                // center the application window
                frame.setLocationRelativeTo(null);
                frame.setVisible(true);    // show it
            }
        });
    }
}
```

Untuk menjalankan program ini, perlu di-compile dan dijalankan file Java yang berisikan ethod void main. Pada program ini Java Class GameMain.java yang akan di-compile dan dijalankan. Output dari program ini menghasilkan GUI tic tac toe seperti Gambar 6.2.



Gambar 6.2. GUI Permainan Tic Tac Toe

6.3.2. Menggunakan Java Applet (JApplet)

Program permainan tic tac toe ini juga dapat dimainkan melalui web browser. Buatlah sebuah file Java Applet dengan nama AppletMain.java dengan memasukkan class `javax.swing.JApplet`. Program file AppletMain.java diimplementasikan seperti berikut.

```
import javax.swing.*;

/** Tic-tac-toe Applet */
@SuppressWarnings("serial")
public class AppletMain extends JApplet {
      /** init() to setup the GUI components */
      @Override
      public void init() {
            // Run GUI codes in the Event-Dispatching thread for thread safety
            try {
                // Use invokeAndWait() to ensure that init() exits after GUI construction
            SwingUtilities.invokeAndWait(new Runnable() {
                @Override
                public void run() {
                    setContentPane(new GameMain());
```

```
                }
            });
            } catch (Exception e) {
               e.printStackTrace();
            }
        }
}
```

Terakhir, tuliskan sebuah file HTML dengan nama (misalnya TicTacToe.html) yang menempelkan Class AppletMain. Berikut code program TicTacToe.html.

```
<html>
   <head>
        <title>Tic Tac Toe</title>
   </head>
<body>
   <h1>Tic Tac Toe</h1>
   <applet code="AppletMain.class" width="300" height="330" alt="Error
Loading Applet?!">   Your browser does not seem to support &lt;APPLET&gt;
tag!
   </applet>
</body>
</html>
```

Tugas:

Tuliskan semua Java Code diatas, kemudian pelajari code nya, dan ubahkan beberapa bagian untuk melihat perubahannya.

MODUL 7
FIRST ORDER LOGIC

7.1. Tujuan

Memperkenalkan kepada mahasiswa dasar-dasar bahasa pemrograman logic (SWI PROLOG). Mahasiswa diharapkan mampu menerjemahkan dan mereresentasikan kasus-kasus order logic ke dalam program komputer; mampu memahami konsep logika proposional *(Proposional Logic)* dalam menyelesaikan suatu permasalahan logika.

7.2. Dasar Teori

7.2.1. Logika Proposisi (*Propositional Logic*)

Logika Proposisi (*Propositional Logic*) menawarkan logika dalam bentuk sederhana sehingga mudah dipahami. Meskipun begitu, Logika Proposisi sudah mampu membantu menarik kesimpulan. Namun, banyak kasus yang muncul akan menjadi terlihat panjang dan rumit saat diwujudkan dalam bentuk Logika Proposisi. Dan itu bisa lebih panjang dan rumit dibandingkan problem itu sendiri.
Saya ambil contoh berikut ini. Di sebuah kelas II SD, terdapat 35 siswa. Setiap hari Senin sampai dengan Kamis, mereka mengenakan seragam merah-putih. Sedangkan hari lain, mereka mengenakan seragam pramuka. Anak tetanggaku yang bernama Amin, ada salah satu siswa kelas II SD tersebut. Hari Rabu pagi kami bertemu saat dia berangkat sekolah. Seragam apa yang dia kenakan?
Bagaimana menyelesaikan contoh tersebut dengan menggunakan Logika Proposisi?

Solusi:
Misalkan:
p: amin adalah siswa kelas II SD
q: amin mengenakan seragam merah putih
r : hari rabu

Kalimat yang bisa kita nyatakan dari cerita tersebut adalah
1 : p Λ r → q
2 : p
3 : r

Dengan ekpresi seperti itu, kita sudah bisa menarik kesimpulan tentang Amin. Tetapi banyak informasi yang tidak dinyatakan dan terlewatkan. Akibatnya, ekspresi tersebut tidak bisa digunakan untuk membuat kesimpulan tentang seragam yang dipakai Ali pada hari Rabu jika diketahui bahwa Ali juga seorang siswa kelas SD tersebut. Agar bisa membuat kesimpulan tentang Ali, kita bisa mengubahnya menjadi seperti di bawah ini:

1 : $p_1 \Lambda r \rightarrow q$

2 : p_1

3 : r

4 : $p_2 \Lambda r \rightarrow q$

5 : p_2

dengan $\mathbf{p_1}$ berarti "amin adalah anak kelas II SD" dan $\mathbf{p_2}$ berarti "ali adalah anak kelas II SD". Bagaimana jika untuk semua siswa? Kita harus menambahkan lagi kalimat nomor 1 dan 2 dengan sebelumnya mengubah $\mathbf{p_1}$ menjadi $\mathbf{p_3}$. Demikian seterusnya sampai $\mathbf{p_{35}}$. Maka akan diperoleh 71 kalimat. Padahal, solusi ini hanya untuk hari Rabu saja, belum hari-hari yang lain.

Predicate: Simbol dengan Parameter

First order Logic menawarkan penggunaan simbol dengan parameter. Simbol ini dikenal sebagai predikat. Sebuah predikat didefinisikan sebagai atribut(sifat) sebuah obyek atau relasi antar obyek. Obyek-obyek tersebutlah yang dijadikan sebagai parameter predikat tersebut.

Sebagai contoh, kita kembali ke contoh sebelumnya. Untuk menyelesaikan contoh tersebut, kita menggunakan simbol **p** untuk menyatakan atribut **seorang siswa kelas II SD**, **r** untuk menyatakan atribut **nama hari**, dan **q** untuk menyatakan relasi **mengenakan seragam**. Definisi lengkap setiap simbol, termasuk parameternya, adalah sebagai berikut:

p(x): x adalah seorang siswa kelas II SD

r(x): x adalah nama hari

q(x,y) : x mengenakan seragam y.

Dengan definisi tersebut, jika kita ingin mengungkapkan kalimat **amin adalah seorang siswa kelas II SD**, **hari rabu**, dan **amin mengenakan seragam pramuka** maka dapat dinyatakan sebagai berikut:

p(amin)

r(rabu)

q(amin,pramuka)

Quantifier

Selain penggunaan predikat, First Order Logic juga menawarkan *quantifier* untuk membuat kalimat logika yang lebih sederhana. Ada 2 jenis *quantifier*, yaitu *universal* dan *existential*. *Quatifier* ini berlaku terhadap parameter yang muncul di sebuah kalimat masih dalam bentuk variabel. *Universal quantifier* terhadap sebuah variabel x (disimbolkan dengan **∀x**) berarti bahwa kalimat

tersebut berlaku untuk setiap obyek x, sedangkan *existential quantifier* (disimbolkan dengan **∃x**) berarti berlaku untuk sebagian obyek saja.

Contoh: Menggunakan definisi untuk **p(x)**, **r(x)**, dan **q(x,y)**, berikut adalah kalimat-kalimat logika dengan menggunakan *quantifier* dan artinya:

∀x(p(x) Λ r(rabu) → q(x,merah-putih)) : untuk setiap x, jika x adalah seorang siswa kelas II SD dan pada hari Rabu maka x akan mengenakan seragam merah-putih.

∃x(p(x) → ¬q(x,merah-putih)) : ada x, jika x adalah seorang siswa kelas II SD maka x tidak mengenakan seragam merah putih.

7.2.2. Contoh First Order Logic dan Penarikan Kesimpulan

Lihat kembali contoh seragam Amin di atas. Solusi untuk problem di atas adalah sebagai berikut.

Solusi:

Misalkan:

p(x) : x adalah seorang siswa kelas II SD

r(x) : x adalah nama hari

q(x,y) : x mengenakan seragam y.

Kalimat yang bisa kita nyatakan dari cerita tersebut adalah

1 : ∀x(p(x) Λ r(senin) → q(x,merah-putih))

2 : ∀x(p(x) Λ r(selasa) → q(x,merah-putih))

3 : ∀x(p(x) Λ r(rabu) → q(x,merah-putih))

4 : ∀x(p(x) Λ r(kamis) → q(x,merah-putih))

5 : ∀x(p(x) Λ r(jumat) → q(x,pramuka))

6 : ∀x(p(x) Λ r(jumat) → q(x,pramuka))

Jika diketahui bahwa Amin adalah seorang siswa kelas II SD dan hari rabu, maka ditambahkan kalimat berikut:

7 : p(amin) Λ r(rabu)

Proses penarikan kesimpulan untuk menjawab pertanyaan **apa seragam yang dipakai oleh Amin pada hari Rabu** adalah sebagai berikut:

8 : p(amin) Λ r(rabu) → q(amin,merah-putih) *{Instansiasi x dengan Amin pada kalimat 3}*

9 : q(amin,merah-putih) *{Modus Ponens antara 7 dan 8}*

Arti kalimat 9 adalah Amin mengenakan seragam merah-putih.

————————- []

Instansiasi: membuang *quantifier* dan mengganti kemunculan setiap variabel yang terkait dengan *quantifier* tersebut dengan sebuah obyek.

Contoh yang lain: Menggunakan contoh seragam siswa kelas II SD di atas, tetapi yang ditanyakan adalah apakah Taufiq seorang siswa kelas II SD jika diketahui dia tidak mengenakan seragam pramuka pada hari Jumat.

Solusi:

Menggunakan definisi sebelumnya, kita tetap memperoleh kalimat logika sebagai berikut:

1 : ∀x(p(x) Λ r(senin) → q(x,merah-putih))

2 : ∀x(p(x) Λ r(selasa) → q(x,merah-putih))

3 : ∀x(p(x) Λ r(rabu) → q(x,merah-putih))

4 : ∀x(p(x) Λ r(kamis) → q(x,merah-putih))

5 : ∀x(p(x) Λ r(jumat) → q(x,pramuka))

6 : ∀x(p(x) Λ r(jumat) → q(x,pramuka))

Diketahui bahwa taufiq tidak mengenakan seragam pramuka pada hari Jumat. Ditambahkan kalimat-kalimat berikut:

7 : ¬q(taufiq,pramuka)

8 : r(jumat)

Proses penarikan kesimpulan untuk menjawab pertanyaan **apa Taufiq seorang siswa kelas II SD** adalah sebagai berikut:

9 : p(taufiq) Λ r(jumat) → q(taufiq,pramuka) *{Instansiasi x dengan taufiq pada kalimat 5}*

10 : ¬(p(taufiq) Λ r(jumat)) *{Modus Tollens antara 7 dan 9}*

11 : ¬p(taufiq) V ¬r(jumat) *{Hukum de Morgan untuk 10}*

12 : p(taufiq) → ¬r(jumat) *{Ekuivalensi implikasi dengan 11}*

13 : ¬p(taufiq) *{Modus Tollens antara 8 dan 12}*

Arti kalimat 14 adalah Taufiq bukan seorang siswa kelas II SD.

7.3. Penggunaan SWI-PROLOG (under MacOS atau LinuX) untuk Kasus First Order Logic

Kasus 1 :

- Buka salah satu teks editor (misalnya vi, vim, pico, gedit, notepad, etc), kemudia ketikkan program berikut dan berikan nama file **OrderLogic1.pl** pada folder local, (misalnya /Document/Data/FOL).

```
p(amin).
r(rabu).

q(X,merah-putih) :- p(X), r(senin).
q(X,merah-putih) :- p(X), r(selasa).
q(X,merah-putih) :- p(X), r(rabu).
q(X,merah-putih) :- p(X), r(kamis).
q(X,pramuka) :- p(X), r(jumat).
q(X,pramuka) :- p(X), r(sabtu).
```

- Jalankan program SWI-PROLOG melalui "terminal" dengan menuliskan perintah :

```
IZamanhuri$ /opt/local/bin/swipl   <enter>

% library(swi_hooks) compiled into pce_swi_hooks 0.00 sec, 2,284 bytes
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version 5.10.4)
Copyright (c) 1990-2011 University of Amsterdam, VU Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

?- <kursor>
```

- Setelah masuk ke dalam SWI-PROLOG, jalankan file OrderLogic1.pl dengan menuliskan perintah dibawah ini pada bagian <kursor>.

```
?- ['OrderLogic1.pl'].
% OrderLogic1.pl compiled 0.00 sec, 1,944 bytes
true.

?- q(amin,X).
X = merah-putih .
```

KESIMPULAN: *Si Amin memakai baju merah-putih.*

- Untuk KELUAR dari program SWI-PROLOH ketikkan.

```
?- halt.
```

Kasus 2 :

- Buka salah satu teks editor (misalnya vi, vim, pico, gedit, notepad, etc), kemudia ketikkan program berikut dan berikan nama file **OrderLogic2.pl** pada folder local, (misalnya /Document/Data/FOL).

```
p(amin).
r(rabu).

q(X,merah-putih) :- p(X), r(senin).
q(X,merah-putih) :- p(X), r(selasa).
q(X,merah-putih) :- p(X), r(rabu).
q(X,merah-putih) :- p(X), r(kamis).
q(X,pramuka) :- p(X), r(jumat).
q(X,pramuka) :- p(X), r(sabtu).

not q(taufiq,pramuka).
r(jumat).
```

- Jalankan program SWI-PROLOG melalui "terminal" dengan menuliskan perintah :

```
IZamanhuri$ /opt/local/bin/swipl   <enter>

% library(swi_hooks) compiled into pce_swi_hooks 0.00 sec, 2,284 bytes
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version 5.10.4)
Copyright (c) 1990-2011 University of Amsterdam, VU Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

?- <kursor>
```

- Setelah masuk ke dalam SWI-PROLOG, jalankan file OrderLogic2.pl dengan menuliskan perintah dibawah ini pada bagian <kursor>.

```
?- ['OrderLogic2.pl'].
% OrderLogic2.pl compiled 0.00 sec, 1,944 bytes
true.

?- p(taufiq).
false .
```

Artinya: *Si Taufiq BUKAN siswa kelas II SD.*

```
?- not(p(taufiq)).
true .
```

Artinya: *Si Taufiq BUKAN siswa kelas II SD.*

- Untuk KELUAR dari program SWI-PROLOH ketikkan.

```
?- halt.
```

Tugas:

- Buat dan carilah contoh First Order Logic yang lainnya, kemudian implementasikan kasus tersebut dengan menggunakan SWI-PROLOG.

Implementasikan Kasus "Hukum Pernikahan" (Lihat slide-07) menggunakan program SWI-PROLOG.

MODUL 8
KNOWLEDGE REPRESENTATION

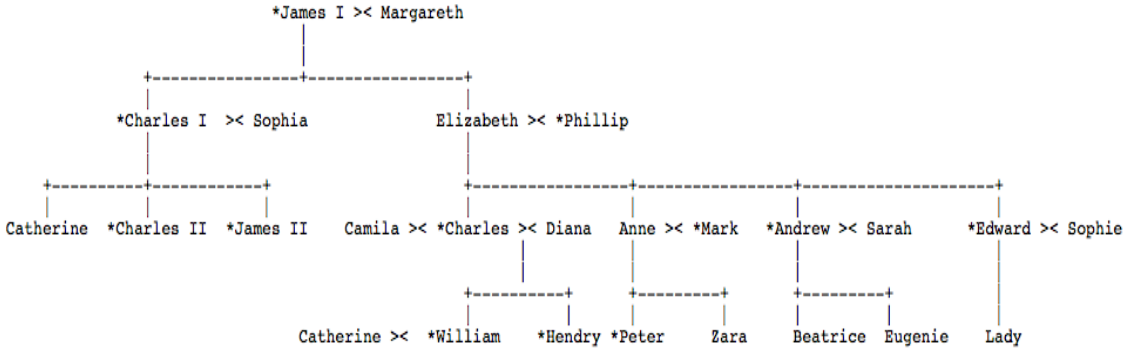
8.1. Tujuan

Memberikan pemahaman kepada mahasiswa untuk merepresentasikan kasus logik berdasarkan pengetahuan dengan menggunakan First Order Logic. Modul 8 ini memperlihatkan kasus Silsilah Keturunan Kerajaan Inggris untuk diimplementasikan dengan menggunakan SWI-PROLOG.

8.2. Dasar Teori

8.2.1. Silsilah Keturunan Kerajaan Inggris

Modul 8 memperlihatkan bentuk silsilah keturunan Kerajaan Inggris. Silsilah keturunan ini dijabarkan hanya empat level saja, dari keluarga Raja James I sampai dengan cicinya seperti Pangeran William, Hendry, Peter, dan Lady. Gambar 8.1 memperlihatkan silsilah keturunan Raja Inggris.



Gambar 8.1 Silsilah Keturunan Kerajaan Inggris.

Keterangan simbol:

- Simbol * merepresentasikan jenis kelamin Laki-laki
- Simbol >< menunjukkan status perkawinan
- Simbol x >< y wanita melambangkan keturunan (anak) dari pasangan x dan y.

Dengan menggunakan PROLOG, implementasikanlah beberapa ketentuan berikut:

M adalah ibu dari X jika dia merupakan orangtua dari X dan dia adalah wanita F adalah ayah dari X jika dia adalah orangtua dari X dan dia adalah laki-laki X adalah saudarakandung Y jika mereka mempunyai orantua yang sama.

Selanjutnya tambahkanlah dalam program PROLOG Anda beberapa definisi ketentuan-ketentuan untuk :

"kakak", "abang", "ibutiri", "tantekandung" "tante", "paman", "tantesepupu", "kakekbuyut" "kakek", "nenek", "sepupu", "nenekbuyut", "cicit", "cicitperempuan"

Melalui Gambar 8.1 dan program PROLOG, bangkit beberapa pertanyaan (query) berikut ini:

1. Apakah George I adalah ayah dari Charles I?
2. Siapakah nama ayah Charles I?
3. Siapakah nama Ibu Hendry?
4. Siapakah anak dari Charles?
5. Apakah Andrew adalah paman William?
6. Siapakah tante Peter?
7. Siapakah sepupu Zara?
8. Siapakah nenek dari William?
9. Apakah Elizabeth adalah nenek dari William dan Hendry?
10. Siapakah kakek dari Peter?
11. Siapakah ibutiri William?
12. Siapakah abang dari Diana?
13. Siapakah kakak dari Andrew?
14. Siapakah abang dari James II?
15. Apakah Sarah adalah tante dari Peter?
16. Siapakah nenekbuyut dari Hendry?
17. Sipakah paman dari Charles?
18. Apakah Diana adalah ibu kandung Hendry?
19. Siapakah sepupu Hendry?
20. Siapakah cicitperempuan dari Elizabeth?

8.3. Implementasi Silsilah Keluarga

Buat dan implementasikan kasus seperti Gambar 8.2 ke dalam program SWI-PROLOG untuk mengetahui beberapa pertanyaan (query) yang terkait dengan silsilah keluarga dan keturunan dari pasangan Yuda dan Nina.

Gambar 8.2 Silsilah Keturunan Yuda dan Nina.

Perintah-perintah

Silahkan dicoba perintah-perintah di bawah ini satu persatu dan lihat hasilnya. kemudian cocokan jawabanya dengan pohon silsilah keluarga pada gambar diatas.

- married(yuda,X).
- child(rico,X).
- parents(nana,X,Y).
- grandparents(rudi,X,Y).
- sibling(danang,X).
- sister(ana,X).
- sister(dita,X).
- brother(tedi,X).
- brother(rudi,X).

Kasus Silsilah Keluarga Yuda & Nina :

- Buka salah satu teks editor (misalnya vi, vim, pico, gedit, notepad, etc), kemudia ketikkan program berikut dan berikan nama file **SilsilahKeluarga.pl** pada folder local, (misalnya /Document/Data/FOL).

```
married(yuda,nina).
married(rico,dina).
married(hari,ambar).
married(tatang,yani).
married(joko,endah).

child(rico,yuda).
child(ambar,yuda).
child(tatang,yuda).
child(joko,yuda).
child(budi,rico).
child(ani,rico).
child(ajeng,rico).
child(rani,rico).
child(danang,rico).
child(ika,hari).
child(tuti,hari).
child(rudi,hari).
child(ana,hari).
child(eko,tatang).
child(dita,tatang).
child(tedi,tatang).
child(adi,joko).
child(nana,joko).
child(rifki,joko).
child(antok,joko).

male(yuda).
male(rico).
male(hari).
```

```
male(tatang).
male(joko).
male(budi).
male(danang).
male(rudi).
male(eko).
male(tedi).
male(adi).
male(rifki).
male(antok).

parents(A,B,C) :-child(A,B),married(B,C).
grandparents(A,D,E) :-child(A,B),child(B,D),married(D,E).
grandparents(A,D,E)                                                    :-
child(A,B),married(B,C),child(C,D),married(D,E).
sibling(A,F) :-child(A,B), child(F,B), (F) \== (A).
sister(A,G):-child(A,B), child(G,B), (G) \== (A), not(male(G)).
brother(A,H):-child(A,B), child(H,B), (H) \== (A), male(H).
grandchilds(A,B,C) :- married(A,B), child(C,A).
```

- Jalankan program SWI-PROLOG melalui "terminal" dengan menuliskan perintah :

```
IZamanhuri$ /opt/local/bin/swipl   <enter>

% library(swi_hooks) compiled into pce_swi_hooks 0.00 sec,
2,284 bytes
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version
5.10.4)
Copyright (c) 1990-2011 University of Amsterdam, VU Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free
software,
and you are welcome to redistribute it under certain
conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

?- <kursor>
```

- Setelah masuk ke dalam SWI-PROLOG, jalankan file OrderLogic1.pl dengan menuliskan perintah dibawah ini pada bagian <kursor>.

```
?- ['SilsilahKeluarga.pl'].
% SilsilahKeluarga.pl compiled 0.00 sec, 1,944 bytes
true.

?- married(yuda,X).
X = nina.

?- child(rico,X).
X = yuda.
```

```
?- parents(nana,X,Y).
X = joko,
Y = endah.

?- grandparents(rudi,X,Y).
X = yuda,
Y = nina.

?- sibling(danang,X).
X = budi .

?- sister(ana,X).
X = ika .

?- sister(dita,X).
false.

?- brother(tedi,X).
X = eko .

?- brother(rudi,X).
false.
```

- Untuk KELUAR dari program SWI-PROLOH ketikkan.

```
?- halt.
```

Tugas:

- Buat dan implementasikan Silsilah Keturunan Kerajaan Inggris, seperti Gambar 8.1 dan penjelasan pada subbab 8.2.1 dengan menggunakan SWI-PROLOG.

Bangkitkan semua jawaban dari 20 pertanyaan yang dideskripsikan pada subbab 8.2.1.