TUGAS AKHIR

RANCANG BANGUN APLIKASI PENCARIAN RUTE TERPENDEK ANTAR KAMPUS DI WILAYAH KAB. JEMBER MENGGUNAKAN ALGORITMA A* (A Star)

Diajukan Untuk Melengkapi Tugas Dan Memenuhi Syarat Kelulusan
Program Strata 1 Jurusan Teknik Informatika
Fakultas Teknik Universitas Muhammadiyah Jember

Disusun Oleh :
Adenta Shara Biyan Safi'i
(0910651235)

Dosen Pembimbing :
Daryanto, S. Kom, M. Kom
Ulya Anisatur, R, S. Kom

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS MUHAMMADIYAH JEMBER
2015

HALAMAN PENGESAHAN

RANCANG BANGUN APLIKASI
PENCARIAN RUTE TERPENDEK ANTAR KAMPUS DI WILAYAH KAB. JEMBER MENGGUNAKAN ALGORITMA A* (A-Star)

Oleh :

Adenta Shara Biyan Safi'i
0910651235

Diajukan untuk melengkapi persyaratan memperoleh
gelar Sarjana Komputer pada Program Studi Teknik Informatika
di
UNIVERSITAS MUHAMMADIYAH JEMBER

Disetujui oleh :

Tim Penguji I	*DosenPembimbing I*
Ari Eko Wardoyo, ST, M. Kom	**Daryanto, S. Kom, M. Kom**
NIP. 197502142005011001	**NPK. 11 03 589**
Tim Penguji II	*Dosen Pembimbing II*
Hardian Oktavianto, S.Si	**Ulya Anisatur R.S. Kom**
NPK. 12030715	**NPK. 0710037903**

Jember, 30 Januari 2015
Mengesahkan

Dekan Fakultas Teknik	*Ketua Program Studi Teknik Informatika*
Ir. Rusgianto, MM	**Agung Nilogiri, ST, M. Kom**
NIP. 131 863 867	**NIP. 19770330 200501 1 002**

KATA PENGANTAR

Puji syukur Alhamdulillah atas kehadirat Allah. SWT, karena atas Rahmat dan Karunia-Nya, penulis dapat menyelesaikan laporan Tugas Akhir yang berjudul "Rancang Bangun Aplikasi Pencarian Rute Terpendek Antar Kampus di Wilayah Kab. Jember Menggunakan Algoritma A* (A Star)". Laporan Tugas Akhir ini disusun sebagai salah satu syarat untuk menyelesaikan pendidikan program Strata 1 pada Fakultas Teknik Jurusan Teknik Informatika Universitas Muhammadiyah Jember. Dengan selesainya laporan Tugas Akhir ini, penulis mengucapkan terima kasih kepada :

1. Bapak Ir. Rusgianto, MM selaku Dekan Fakultas Teknik Universitas Muhammadiyah Jember.
2. Bapak Agung Nilogiri, ST, M. Kom selaku Kepala Program Studi Teknik Informatika Fakultas TeknikUniversitas Muhammadiyah Jember.
3. Bapak Daryanto, S. Kom, M. Kom dan Ibu Ulya Anisatur, R. S. Kom selaku dosen pembimbing yang meluangkan waktunya untuk membimbing penulisan laporan ini.
4. Bapak Ari Eko Wardoyo, ST, M. Kom dan Bapak Hardian Oktavianto. S, Si selaku dosen penguji yang memberikan saran dan kritik yang membangun dalam penelitian ini.
5. Bapak dan Ibu Dosen Fakultas Teknik Universitas Muhammadiyah Jember.
6. Kepada Ibu dan Bapak serta keluarga besar saya yang selalu mendukung, dan mendoakan, sehingga penulis dapat menyelesaikan Tugas Akhir ini.

Penulis menyadari bahwa laporan Tugas Akhir ini jauh dari kata sempurna, baik menyangkut aspek penulisan maupun materi. Untuk itu tanggapan berupa kritik dan saran yang bersifat membangun sangat penulis harapkan demi kesempurnaan laporan ini.

Jember, 23 Januari 2015

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR LAMPIRAN

Pseucode Algoritma A* (A Star) :

```
/**
 * Copyright 2005 Sean J. Barbeau
   Licensed under the Apache License, Version 2.0 (the "License");
   you may not use this file except in compliance with the License.
   You may obtain a copy of the License at
    http://www.apache.org/licenses/LICENSE-2.0
   Unless required by applicable law or agreed to in writing, software
   distributed under the License is distributed on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
   See the License for the specific language governing permissions and
   limitations under the License.
 */

package astar;
import com.sun.xml.internal.bind.v2.schemagen.xmlschema.Import;
import javax.swing.JOptionPane;

/**
 * This class allows an implementation of the A*Star algorithm with different heuristics depending on which is
 * picked by the user.
 * Each of the functions in this class need a new option for each new heuristic
 * Current options for Heuristics:
      1:  Fewest Links
      2:  Shortest Distance
 * When adding a new heuristic, another option must be added to the "switch" statement in both the getCost() and
 * getEstimatedCostToGoal() statements
 *
```

```
 * @author Adhen
 */
public class HeuristicsNode extends NodeXY {

   /**
    * Creates a new instance of a Heuristics Node
    * @param label label for the node
    * @param x x position of the node
    * @param y y position of the node
    */
   protected HeuristicsNode(String label, int x, int y){
      super(label, x, y);
   }



   /**
    * This function gets the real cost betweeen this node and node_B
    * @param nodeB
    * @param heuristic
    * @return actual cost from this node to node_B
    */
   @Override

   public double getCost(NodeXY nodeB, Heuristic heuristic){
      //Get the cost from one node to another according to a heuristic

      //Initialize variables
      double cost = 0;

      //Select heuristic
      switch (heuristic) {
         case FEWEST_LINKS:
            //Fewest Links
```

```
            cost = 1;
            break;
        case SHORTEST_DISTANCE:
            //Shortest Distance - measure the distance between nodes using D = sqrt( (x2-x1)squared + (y2-y1)squared )
            double totalDistance = 0;

            int tempx = this.location.x - nodeB.location.x;
            int tempy = this.location.y - nodeB.location.y;

            tempx = tempx * tempx;
            tempy = tempy * tempy;

            try {
               totalDistance = Math.sqrt(tempx + tempy);

            }
            catch(Exception E)
            {
               System.out.println("Error in get_Distance()  "+E);
               totalDistance = 0;//In case of error
            }
            cost = totalDistance;
            break;
        //case 3:
            //XXXXXXXXXXXXXX - method to calculate a new heuristic
            //ADD NEW HEURISTIC HERE:  CALCULATE THE COST AND SET THE VARIABLE cost = TO THE CALCULATED COST
            //break;
        default:
            //Use fewest Links as default
            cost = 1; //It is known that 1 will definitely be an underestimate, so 1 is used
            break;
```

```
        }
        //Return the cost calculated by the heuristic
        return cost;
    }
    /**
     * This function gets the estimated cost between this node and the goal node
     * @param goalNode
     * @param heuristic type of heuristic to be used
     * @return estimated cost to goal
     */
    @Override
    public double getEstimatedCostToGoal(NodeXY goalNode, Heuristic heuristic) {
        //Initialize variable for cost
        double estCost = 0;

        //Select which heuristic to use
        switch(heuristic) {
            case FEWEST_LINKS:
                //Fewest Links Heuristic
                estCost = 1; //It is known that 1 will definitely be an underestimate, so 1 is used
                break;
            case SHORTEST_DISTANCE:
                //Shortest Distance
                estCost = this.getCost(goalNode, heuristic); //Uses the other function to calculate distance between nodes,
                //since the direct distance from this node to the goal is an underestimate of the total distance to the goal
                break;
            //case 3:
                //XXXXXXXXXXXXXX - method to calculate a new heuristic
                //ADD NEW HEURISTIC HERE:  CALCULATE THE ESTIMATED COST TO THE GOAL AND SET THE VARIABLE est_cost = TO THE CALCULATED
```

```
COST
            //break;
        default:
            //Use Fewest Links as default
            estCost = 1; //It is known that 1 will definitely be an underestimate, so 1 is
used
            break;
      }
      //Return value
      return estCost;
   }
}
```

Pseucode Proses Pencarian Algoritma A* (A Star) :

```
/**
 * Copyright 2005 Sean J. Barbeau
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

     http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License.
 */

package astar;
import com.barbeau.networks.visualization.MapDisplay;
import com.barbeau.networks.SearchSpace;
import java.util.*;
```

```
import java.awt.*;
import java.util.Collections.*;

/**
*******************************
* A* algorithm implementation *
*******************************

Created by Sean J. Barbeau
This application was created in the NetBeans and is editable in that environment
by opening this root directory in NetBeans.
The executable is "\dist\Astar.jar", and is launched in Windows by double-clicking on it from a normal window.  To be safe you may want to copy the entire root and subdirectories to your hard drive and then execute it.
The source code is in "\src\astar".
Both input files should be located in the same directory as the .jar file to load from their default location, although the file browser buttons
can be used to specify files in other locations.
Program execution:
1)  Start application
2)  Click on "Load files"
3)  Select Start and Goal nodes and select heuristic
4)  User can disable nodes so they won't be used in the search path by selecting the 'Enable/Disable Node' radio button option and clicking on the map.
5)  User can move nodes by selecting the 'Move Node' radio button option and clicking and dragging a node around the map.
6)  User can select to either progress step-by-step through the algorithm or immediately finish the algorithm by toggling the "Step-by-Step" button.
7)  User can select the length of the step time from .1 to 10 seconds by moving the "Step time" slider.
6)  Click "Start" to begin algorithm execution.  6) and 7) can also be performed during execution of the algorithm.
7)  After algorithm finishes (or during execution if desired) click "Reset" to reset the
```

algorithm to its initial state. You can either run the algorithm again or load a new set of files.

The output from the algorithm (path from start to goal node) should be shown in the "A Log" text box within the application window as well as the graphical display.*
Start and goal nodes are shown in blue, current node being processed is in green, and any disabled nodes are red.
Links are shown as light blue if they are traveled, and as dark blue if they are part of the path after the algorithm finds a path.

To add more heuristics, modify the "getCost(Node nodeB, int heuristic)" and "getEstimatedCostToGoal(Node goalNode, int heuristic)" functions in the class "Heuristics_Node" to add a new heuristic calculation as part of the 'switch' statement.
Then pass the integer used for that 'switch' statement into the "heuristic" argument of the AstarSearch object when it is created.

References used for A algorithm:*

Russel, Norvig. "Artificial Intelligence: A Modern Approach". Prentice Hall 2003.
Lugar. "Artificial Intelligence: Structures and Strategies for Complex Problem Solving". Addison Wesley, 2005.
Winston. "Artificial Intelligence". Addison-Wesley Publishing Company, 1992.
DevX.com Bulletin, "Breadth first search with java", Available online at http://news.devx.com/printthread.php?t=145297.
Brackeen. "Game Character Path Finding in Java", informit.com, available online at http://www.informit.com/articles/article.asp?p=101142&seqNum=2&rl=1.
Heyes-Jones, "A algorithm Tutorial", available online at http://www.geocities.com/jheyesjones/astar.html.*
Adair, Ball, and Pawlan. "The Java Turtorial, Trail: 2D Graphics", available online at http://java.sun.com/docs/books/tutorial/2d/index.html.
Adair, Ball, and Pawlan. "Stroking and Filling Graphics Primitives", available online at http://java.sun.com/docs/books/tutorial/2d/display/strokeandfill.html.
Sun Microsystems. "How to Use File Choosers", available online at http://java.sun.com/docs/books/tutorial/uiswing/components/filechooser.html.

Sun Microsystems. "Maintaining a Priority Queue and Displaying Text in Multiple Styles", available online at http://java.sun.com/developer/JDCTechTips/2002/tt0821.html.
Sun Microsystems. "Java Forums - Drawing arrows", available online at http://forum.java.sun.com/thread.jspa?threadID=378460&tstart=135.

```
 * @author Adhen
 */

public class AstarSearch extends Thread
{
   /** Declares variables necessary for the search **/
   private SearchSpace searchSpace = new SearchSpace();  //Search space for algorithm
   private HeuristicsNode startNode;  //Start node for search algorithm
   private HeuristicsNode goalNode;  //Goal node for search algorithm
   private HeuristicsNode currentNode = null; //Current node being examined
   private Heuristic heuristic = Heuristic.FEWEST_LINKS;  //Selected Heuristic to use to measure cost.  Default = Fewest Links
   private LinkedList path = new LinkedList(); //Variable that holds the path if the goal node is found
   private int numIterations = 0; //Counter to count the iterations of algorithm
   //Variables to show text to the user in the main interface
   private javax.swing.JTextArea textLog;
   private static String NEW_LINE = "\n";

   //Variable to show the map of the search space graphically to the user
   private MapDisplay map;
   //Variable defines whether application runs quickly through (default) or step-by-step
   private boolean stepByStep = false;
   //Variable defines the length of step time (in milliseconds)
   private int stepTimeDelay = 1000;

   /**
    * Constructor for AstarSearch
```

```
     *
     * @param searchSpace search space to run Astar search on
     * @param startNode node to start Astar search from
     * @param goalNode node to find using Astar search
     * @param heuristic heuristic to use in Astar algorithm
     * @param textLog log to print output to
     * @param map map to display search output on
     */
    public AstarSearch(SearchSpace searchSpace, HeuristicsNode startNode, HeuristicsNode goalNode, Heuristic heuristic, javax.swing.JTextArea text_log, MapDisplay map)
    {
      super();
      //Initialize search variables
      this.searchSpace = searchSpace;
      this.startNode = startNode;
      this.goalNode = goalNode;
      this.heuristic = heuristic;
      this.numIterations = 0;
      this.path = null;
      //Set text log to print messages to the user
      this.textLog = text_log;
      //Set map display to show search space and nodes
      this.map = map;
    }
   /**
     * Thread to run Astar search
     */
    @Override
   public void run()
    {
       //Print header
this.printToLog("**********************************************************************");
```

```
        this.printToLog("Starting A* search...");

        try
        {
            //Print start and goal nodes
            this.printToLog("Starting node = " + startNode.label);
            this.printToLog("Goal node = " + goalNode.label);
            //Start search algorithm, which returns the path from the start to the goal node if one exists
            this.path = aStarSearch();
            //If a path to the goal node was found then print the path, else say a path to goal doesn't exist.
            //Print out the path & set the MapDisplays path
            if(path != null) {
                this.printPath(this.path);
                this.map.setPath(this.path);
            }
            Else
{
this.printToLog("*****************************************************************");
                this.printToLog("Algorithm couldn't find path to goal, and therefore one does NOT exist.");
            }
            System.out.println("Finished run method in AstarSearch");
            //Print footer
            this.printToLog("Ended A* search.");
this.printToLog("*****************************************************************");
            this.printToLog("Please press the 'Reset' button to re-initialize the application.");
        }
        catch(Exception e)
        {
```

```
            System.out.println(e);
        }
    }
    /**
     * This function starts the actual A* search algorithm
     * @return the path from the start to the goal node if one exists
     */
    public LinkedList aStarSearch() {
        //Variable to hold nodes to still be searched
        PriorityQueue available = new PriorityQueue();

        //Variable to hold nodes that have already been explored
        LinkedList visited = new LinkedList();
        try {
            //Print Heuristic that is being used
            switch(this.heuristic) {
                case FEWEST_LINKS:
                    //Fewest Links
                    this.printToLog("The 'Fewest Links' Heuristic is being used.");
                    break;
                case SHORTEST_DISTANCE:
                    //Shortest Distance
                    this.printToLog("The 'Shortest Distance' Heuristic is being used.");
                    break;
                //case 3:
                    //XXXXXXXXXXXXX
                    //this.printToLog("The 'XXXXXXXXXXXX' Heuristic is being used.");
                    //break;
                //  To print a new heuristic to the text box, uncomment the above 3 lines and replace XXXXXXXXXX with the name of the heuristic
                default:
                    //Use Fewest links as the default
                    this.printToLog("The 'Fewest Links' Heuristic is being used.");
```

```
                break;
            }
            //Set start node variables
            this.startNode.costFromStart = 0;
            //this.startNode.estCostToGoal = this.startNode.getCost(this.goalNode, this.heuristic);
//Changed to below line to reflect better generalized version
            this.startNode.estCostToGoal =
this.startNode.getEstimatedCostToGoal(this.goalNode, this.heuristic);
            this.startNode.pathParent = null;
            //Add Start Node to available list
            available.add(this.startNode);
            //Loop through all the available searchable nodes while there are still nodes available
            while (!available.isEmpty()) {
               //Get the "least expensive" node on list and deletes it from the available queue
               HeuristicsNode nodeA = (HeuristicsNode) available.removeFirst();
               //Set current node for access from outside
               this.currentNode = nodeA;
               //Draw nodeA on map as node that is currently being expanded
               nodeA.drawNode(map, Color.GREEN, map.getExpandedNodeSize());
               //Increment the number of iterations for the algorithm
               numIterations++;
               //If this is the goal then the algorithm is done
               if (nodeA == this.goalNode)
               {
                  //Print node's label
                  this.printToLog("Found goal!! => " + nodeA.label + " (after " + numIterations +
" iterations, total cost = " + nodeA.costFromStart + ")");
                  //Set found_goal variable
                  //found_goal = true;
                  //Deemphasize the goal node in the graphic display
                  nodeA.deemphasizeCurrentNode(map);
                  //returns path
                  return getPath();
```

```
            }
            //Print node's label
            this.printToLog("Expanding " + nodeA.label + ".  Checking neighbors & costs------
--->");
            //Get a list of this node's neighbors
            java.util.List tempList = nodeA.getChildren();
            //Get iterator to loop through nodes neighbors
            Iterator i = tempList.iterator();
            /* Loop through each node "nodeB" that nodeA is connected to, and examine it
            * to see if it has been visited or if a shorter path has been found to it
            */
            while (i.hasNext()) {
               //Get next node that nodeA is connected to
               HeuristicsNode nodeB = (HeuristicsNode)i.next();
               //Get link that defines that connection
               LinkXY tempLink = (LinkXY) this.searchSpace.findLink(nodeA, nodeB);
               //Define this link as traveled
               tempLink.traveled = true;
               //Have thread sleep to pause execution
               if(this.stepByStep == true) {
                  AstarSearch.sleep(this.stepTimeDelay);
               }
               //Draw line to next node
               //node_A.draw_Arrow_to(nodeB, Color.CYAN, map);  //Removed in favor of
using the link objects to draw
               tempLink.drawLink(map, Color.CYAN);
               //Find out whether this node is already available
               boolean isAvailable = available.contains(nodeB);
               //Find out whether this node has been visited
               boolean wasVisited = visited.contains(nodeB);
               //Calculate cost from start for nodeB from this path = cost from start to nodeA +
cost from nodeA to nodeB
               double tempCostFromStart = nodeA.costFromStart + nodeB.getCost(nodeA,
```

```
this.heuristic);
            //If this node is enabled
            if(nodeB.enabled == true) {
            /*If nodeB hasn't already been visited or if the total cost from the startNode to
nodeB
             is less than the one that is already been found to nodeB
             * THEN recalculate the costs and assign new parent node
             */
               if (  (isAvailable == false && wasVisited == false) || (tempCostFromStart <
nodeB.costFromStart)  )
               {
                  //Set nodeB's parent for this path as nodeA
                  nodeB.pathParent = nodeA;
                  //Set new cost from start by copying the newly calculated value
                  nodeB.costFromStart = tempCostFromStart;
                  //Calculate cost from nodeB to goal
                  nodeB.estCostToGoal  =  nodeB.getEstimatedCostToGoal(this.goalNode,
this.heuristic);
                  /* Make sure 'available' and 'visited' lists correctly reflect nodeB's
conditions */
                  if (isAvailable == false)
                  {
                     //Add it to the available list so it can be considered as part of a path to
the goal
                      available.add(nodeB);
                      this.printToLog("     " + nodeB.label + " was added to the list of available
nodes.");
                  }
                  if (wasVisited == true)
                  {
                     //Remove nodeB from the visited list so it can be considered again as part
of a new path to the goal
                      visited.remove(nodeB);
```

```
                    this.printToLog("      " + nodeB.label + " was previously visited but has been added back to the search because it appears a path with lesser cost exists.");
                }
                //Print info on the currently considered nodeB to user
                this.printToLog("      " + nodeB.label + " Costs--> from_start = " + nodeB.costFromStart + ", est_to_goal = " + nodeB.estCostToGoal + ", total = " + nodeB.get_total_heuristic_cost());
            }//End of IF statement to see if nodeB was previously visited or if a shorter path to nodeB had been found
            else {
                //Print info to the user that nodeB is no longer considered
                this.printToLog("    " + nodeB.label + " has already been visited and doesn't appear to be lesser cost to the goal, so it has been removed from the list of considered nodes.");
            }
        } //End of IF to see if node is enabled
        else
        {
            //Node has been DISABLED!!!!  Print message to user
            this.printToLog("      " + nodeB.label + " has been DISABLED.  It cannot be considered in the path to the goal.");
            //Redraw line to next node to show that its not considered
            //node_A.draw_Arrow_to(nodeB, Color.RED, map);  //Removed in favor of using the link object to draw
            tempLink.drawLink(map, Color.RED);
            //Set link as disabled
            tempLink.enabled = false;
        }
    }// End of examination of nodeB, a node that nodeA is connected to
    //Put this node on the visited list since it has now been visited
    visited.add(nodeA);
    //Have thread sleep to pause execution if requested
    if(this.stepByStep == true) {
```

```
                AstarSearch.sleep(stepTimeDelay);
            }
            //Deemphasize the current node in the graphic display
            nodeA.deemphasizeCurrentNode(map);
            //Sort Priority Queue for available nodes so if a shorter path from start was found to a node, the list order reflects it
            Collections.sort(available);
          }
          //If execution reaches this point then the goal was not found, so return null
        return null;
    }
    catch(Exception e) {
        this.printToLog("Error in A* algorithm:" + e);
        return null; //In case of error return null
    }
}
   /**
    * Calculates and returns the path traversed as a list
    * @return the path traversed as a list
    */
   public LinkedList getPath() {
      NodeXY tempNode;
      LinkedList path = new LinkedList();//Variable to hold path
      //Get goalNode to start
      tempNode = this.goalNode;
      //Go backwards through the path but add each node to the front of the list (a stack implementation)
      while(tempNode.pathParent != null)
      {
         //Add node to front of the list
         path.addFirst(tempNode);
         //Go to parent node of temp_node
         tempNode = tempNode.pathParent;
```

```
        }
        //Add the first node
        path.addFirst(this.startNode);
        //Return the list
        return path;
    }
    /**
     * Prints the found path from the start to the goal
     * @param path path from the start to the goal
     */
    public void printPath(LinkedList path) {
        NodeXY tempNode;
        NodeXY previousNode = null;

        Iterator i = path.iterator();  //Get iterator to loop through searchSpace
        //Print intro statement
        this.printToLogNoNewLine("PATH ");
        //Print path with no NEW_LINE between node labels
        while (i.hasNext()) {
            //Get next node
            tempNode = (NodeXY)i.next();
            //Print this node (don't print arrow in front of startNode
            if(tempNode == this.startNode) {
                this.printToLogNoNewLine(tempNode.label);
            }
            else {
                this.printToLogNoNewLine("->" + tempNode.label);
                //***Paint path on map ***
                //Get link that defines that connection
                LinkXY tempLink = (LinkXY) this.searchSpace.findLink(previousNode,
tempNode);
                //previous_node.draw_Arrow_to(temp_node, Color.BLUE, this.map); //Removed in
favor of using link objects to draw lines
```

```
            tempLink.drawLink(map, Color.BLUE);
        }
        //Set previous node
        previousNode = tempNode;
        //Redraw the node to show as part of path
        previousNode.drawNode(this.map,previousNode.color, map.getNormalNodeSize());
    }
    //print blank space to advance to the next line for next print message
    this.printToLog(" ");
  }
  /**
   * Prints output shown to the user in the textbox on the screen & prints to System.out as well
   * @param text text to be printed to screen and system.out
   */
  public void printToLog(String text) {
    try
    {
      this.textLog.append(text + NEW_LINE); //Adds NEW_LINE character so each entry is a new line
      //Moves cursor to the end of the text area to keep new text in view
      this.textLog.setCaretPosition(textLog.getDocument().getLength());
      //Print out to output screen also
      System.out.println(text);
    }
    catch (Exception e)
    {
      System.out.println("Error in print_to_log:" + e);
    }
  }
  public void printToLogNoNewLine(String text) {
    try
    {
```

```
        //This method prints output shown to the user in the textbox on the screen & prints to System.out as well
            this.textLog.append(text); //Adds NEW_LINE character so each entry is a new line
            //Moves cursor to the end of the text area to keep new text in view
            this.textLog.setCaretPosition(textLog.getDocument().getLength());
            //Print out to output screen also
            System.out.println(text);
        }
        catch (Exception e)
        {
            System.out.println("Error in printToLogNoNewLine:" + e);
        }
    }
    /**
     * Returns true if the algorithm is in step-by-step mode, false if it is not
     * @return true if the algorithm is in step-by-step mode, false if it is not
     */
    public boolean getStepByStep() {
        return stepByStep;
    }
    /**
     * Sets whether or not the algorithm is in step-by-step mode
     * @param value true if the algorithm should be in step-by-step mode, false if it should not
     */
    public void setStepByStep(boolean value) {
        this.stepByStep = value;
    }
    /**
     * Returns the length of step time in between each step of the algorithm (in milliseconds)
     * @return the length of step time in between each step of the algorithm (in milliseconds)
     */
    public int getStepTimeDelay() {
        return this.stepTimeDelay;
```

```
    }
    /**
     * Sets the length of step time in between each step of the algorithm (in milliseconds)
     * @param time the length of step time in between each step of the algorithm(in milliseconds)
     */
    public void setTimeDelay(int time) {
        this.stepTimeDelay = time;  //Already in milliseconds from slider
    }
    /**
     * Returns the current node that the algorithm is examining
     * @return the current node that the algorithm is examining
     */
    public HeuristicsNode getCurrentNode() {
        return this.currentNode;
    }
}
```

Pseucode Pemberian Node-node :

```
package astar;
import com.barbeau.networks.Location;
import com.barbeau.networks.visualization.MapDisplay;
import com.barbeau.networks.Node;
import java.awt.*;

/**
 * This class holds the basic information for a node in a network that has X and Y locations
 *
 * @author Adhen
 */
public abstract class NodeXY extends Node implements Comparable {

    public NodeXY pathParent;  //This reference to a node will serve as the link to its parent
```

```
when tracing a path to the goal node.  This way
                    //the path from start to goal can easily be reconstructed when reaching the goal
   public double costFromStart;  //Distance traveled from start node to get to this node
   public double estCostToGoal;  //Estimated distance from this node to the goal, estimated by a heuristic (either # of links or distance traveled
   public Location location;  //Location of node in (x,y)

   //*** Properties of the node used for the graphic MapDisplay ***
   public Color color;  //Variable that holds the current color of the node for the graphic map (used when traversing the path)
   public Color previousColor; //Variable holds the previous color of the node for the graphic map (used when traversing the path)
   public int size; //Defines the current size of the node for the graphic map
   public int previous_size; //Defines the previous size of the node for the graphic map

   /**
    * Creates a new instance of Node with a location to be used for Astar search
    * @param label name of the node
    * @param x X coordinate of the location of the node
    * @param y Y coordinate of the location of the node
    */
   public NodeXY(String label, int x, int y) {
      super(label);
      //Create location property with coords
      this.location = new Location(x, y);

      //Initialize costs
      this.costFromStart = 0;
      this.estCostToGoal = 0;

      //Set graphic properties of node
      this.color = Color.DARK_GRAY;
```

```
        this.previousColor = Color.DARK_GRAY;
        this.size = 7;
        this.previous_size = 7;

        //Print current properties of node
        //System.out.println("Initialized Node '" + this.label + "' to location (" + this.location.x + ", " + this.location.y + ")");
        }

    /**
     * This method calculates the heuristic value f(n) = h(n) + i(n) where h(n) is the cost of the trip from the start node  *
     * to this node "n" and i(n) is the estimated cost to travel from this node "n" to the goal node.  This class will       *
     * be extended to use different heuristics to implement f(n).
     * @return total heuristics cost to the goal
     */
    public double get_total_heuristic_cost() {

        return (this.costFromStart + this.estCostToGoal);
    }



    /**
     * This function compares two node's costs and returns 1 if this node is greater than, 0 if it is equal to and -1 if it is less than node_B. *
     */
    @Override
    public int compareTo (Object node_B){
        double cost = this.get_total_heuristic_cost();
        double node_B_Cost = ((NodeXY)node_B).get_total_heuristic_cost();

        //Calculate difference in costs
```

```
        double temp = cost - node_B_Cost;

        if (temp > 0){
            //this node is greater than node_B
            return 1;
        }
        else {
            if (temp < 0) {
                //this node is less then node_B
                return -1;
            }
            else {
                //this node is equal to node_B
                return 0;
            }
        }

    }

    //*********************************************************
    //* Graphic methods used to draw nodes on a visual map *
    //*********************************************************
    public void drawNode(MapDisplay map, Color color, int size) {

//*************************************************************************************
*************************
        //* This function Draws the node on the map with the specified color and is called from
AstarSearch *

//*************************************************************************************
*************************

        //Graphic object
```

```
        Graphics g;
        //Set graphics object
        g = map.getGraphics();
        //Set previous graphic node properties
        this.previousColor = this.color;
        this.previous_size = this.size;

        //Set current graphic node properties
        this.color = color;
        this.size = size;

        //Set color of graphic object
        g.setColor(color);
        //Draw circle
        g.fillOval((int)(this.location.x * map.SCALE), (int)(this.location.y * map.SCALE),
this.size, this.size);
        //Draw text Label
        g.drawString(this.label, (int)(this.location.x * map.SCALE), (int)(this.location.y *
map.SCALE));
    }

    public void drawNode(Graphics g, MapDisplay map) {

//*************************************************************************************
***************
        //* This function draws the node on the map and is called by the map when it is
refreshed *

//*************************************************************************************
***************

        //Set color of graphic object
        g.setColor(this.color);
```

```
        //Draw circle
        g.fillOval((int)(this.location.x * map.SCALE), (int)(this.location.y * map.SCALE), this.size, this.size);
        //Draw text Label
        g.drawString(this.label, (int)(this.location.x * map.SCALE), (int)(this.location.y * map.SCALE));
    }

    public void emphasizeCurrentNode(MapDisplay map) {
        //This function emphasizes the current node by showing it as expanded and green

        //Graphic object
        Graphics g;
        //Set graphics object
        g = map.getGraphics();

        //Set previous graphic node properties
        this.previousColor = this.color;
        this.previous_size = this.size;

        //Set current graphic node properties
        this.color = Color.GREEN;
        this.size = map.getExpandedNodeSize();

        //Set color of graphic object
        g.setColor(color);
        //Draw circle
        g.fillOval((int)(this.location.x * map.SCALE), (int)(this.location.y * map.SCALE), this.size, this.size);
        //Draw text Label
        g.drawString(this.label, (int)(this.location.x * map.SCALE), (int)(this.location.y * map.SCALE));
```

```
    }

    public void deemphasizeCurrentNode(MapDisplay map) {
        //This function deemphasizes the current node by showing it as regular and its previous color

        //Graphic object
        Graphics g;
        //Set graphics object
        g = map.getGraphics();

        //Set current graphic node properties
        //this.color = this.previousColor;
        //If this node wasn't a start or goal node, then paint it the right color (enabled/disabled)
        if (this.previousColor != Color.BLUE) {
            if(this.enabled == true) {
                this.color = Color.BLACK;
            }
            else {
                this.color = Color.RED;
            }
        }
        else {
            this.color = this.previousColor;
        }

        this.size = map.getExpandedNodeSize();

        //Set color of graphic object
        g.setColor(color);
        //Draw circle
        g.fillOval((int)(this.location.x * map.SCALE), (int)(this.location.y * map.SCALE), this.size, this.size);
```

```
        //Draw text Label
        g.drawString(this.label, (int)(this.location.x * map.SCALE), (int)(this.location.y * map.SCALE));
    }

    public void setLocation (Location location) {
        this.location.x = location.x;
        this.location.y = location.y;
    }



//*************************************************************************************
    //* Abstract methods to be implemented in the extended class for specific Heuristics   *

//*************************************************************************************

    //This function gets the real cost betweeen this node and node_B
    public abstract double getCost(NodeXY node_B, Heuristic heuristic);

    //This function gets the estimated cost between this ndoe and the goal node
    public abstract double getEstimatedCostToGoal(NodeXY goal_node, Heuristic heuristic);

}
```

Pseucode Jalur Pencarian Rute :

```
/**
 * Copyright 2005 Sean J. Barbeau

   Licensed under the Apache License, Version 2.0 (the "License");
   you may not use this file except in compliance with the License.
```

```
 *  You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 *  Unless required by applicable law or agreed to in writing, software
 *  distributed under the License is distributed on an "AS IS" BASIS,
 *  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 *  See the License for the specific language governing permissions and
 *  limitations under the License.
 */

package astar;

import com.barbeau.networks.visualization.MapDisplay;
import com.barbeau.networks.Link;
import java.awt.*;

/**
 * This class defines the one-way links that attach nodes to each other and have X and Y locations,
 * used for GUI operations
 * @author Adhen
 */
public class LinkXY extends Link{


    //Graphics characteristics of this link
    private Color color;

    /**
     * Creates a new instance of Link between two nodes
     * @param nodeA
     * @param nodeB
```

```
     */
    public LinkXY(NodeXY nodeA, NodeXY nodeB) {
        super(nodeA, nodeB);

        //Set default color
        this.color = Color.LIGHT_GRAY;
    }

    /**
     * This function draws an arrow from this.nodeA to this.nodeB with an arrow head pointing at nodeB
     * @param g graphics to use when drawing.  It is called by the map when it is refreshed
     * @param map map to use when drawing
     */
    public void draw_Link(Graphics g, MapDisplay map) {

        //Set color for graphics
        g.setColor(this.color);

        //Get coordinates of nodes (takes into account the size of node and scale of map
        int nodeAx = (int)((((NodeXY) nodeA).location.x + ((NodeXY) nodeA).size/2) * map.SCALE);
        int nodeAy = (int)((((NodeXY) nodeA).location.y + ((NodeXY) nodeA).size/2) * map.SCALE);
        int nodeBx = (int)((((NodeXY) nodeB).location.x + ((NodeXY) nodeB).size/2) * map.SCALE);
        int nodeBy = (int)((((NodeXY) nodeB).location.y + ((NodeXY) nodeB).size/2) * map.SCALE);

        double stroke = .1;

        //Get direction of line
        double tempDirection = Math.atan2(nodeAx-nodeBx,nodeAy-nodeBy);
```

```
        //Build Arrow head
        Polygon temp_Head = new Polygon();
        int i1=12+(int)(stroke*2);
        int i2=6+(int)stroke;

        //Add points to polygon
        temp_Head.addPoint(nodeBx,nodeBy);

temp_Head.addPoint(nodeBx+getX(i1,tempDirection+.5),nodeBy+getY(i1,tempDirection+.5
));

temp_Head.addPoint(nodeBx+getX(i2,tempDirection),nodeBy+getY(i2,tempDirection));
        temp_Head.addPoint(nodeBx+getX(i1,tempDirection-
.5),nodeBy+getY(i1,tempDirection-.5));
        temp_Head.addPoint(nodeBx,nodeBy);

        //Draw line to nodeB from nodeA
        g.drawLine(nodeBx,nodeBy,nodeAx,nodeAy);

        //Draw and fill arrow head
        g.drawPolygon(temp_Head);
        g.fillPolygon(temp_Head);
    }
    /**
     * This function draws an arrow from this.nodeA to this.nodeB with an arrow head
pointing at nodeB
     * It is called by the algorithm to define the color of the link
     * @param map map to draw link on
     * @param color color of link
     */
    public void drawLink(MapDisplay map, Color color) {
```

```
        //Graphic object
        Graphics g;

        //Set graphics object
        g = map.getGraphics();

        //Set color of graphic object
        g.setColor(color);

        //Set color for link
        this.color = color;

        //Get coordinates of nodes (takes into account the size of node and scale of map
        int nodeAx = (int)((((NodeXY) nodeA).location.x + ((NodeXY) nodeA).size/2) *
map.SCALE);
        int nodeAy = (int)((((NodeXY) nodeA).location.y + ((NodeXY) nodeA).size/2) *
map.SCALE);
        int nodeBx = (int)((((NodeXY) nodeB).location.x + ((NodeXY) nodeB).size/2) *
map.SCALE);
        int nodeBy = (int)((((NodeXY) nodeB).location.y + ((NodeXY) nodeB).size/2) *
map.SCALE);

        double stroke = .1;

        //Get direction of line
        double tempDirection = Math.atan2(nodeAx-nodeBx,nodeAy-nodeBy);

        //Build Arrow head
        Polygon tempHead = new Polygon();
        int i1=12+(int)(stroke*2);
        int i2=6+(int)stroke;

        //Add points to polygon
```

```
        tempHead.addPoint(nodeBx,nodeBy);

tempHead.addPoint(nodeBx+getX(i1,tempDirection+.5),nodeBy+getY(i1,tempDirection+.5)
);
        tempHead.addPoint(nodeBx+getX(i2,tempDirection),nodeBy+getY(i2,tempDirection));
        tempHead.addPoint(nodeBx+getX(i1,tempDirection-
.5),nodeBy+getY(i1,tempDirection-.5));
        tempHead.addPoint(nodeBx,nodeBy);

        //Draw line to nodeB from nodeA
        g.drawLine(nodeBx,nodeBy,nodeAx,nodeAy);

        //Draw and fill arrow head
        g.drawPolygon(tempHead);
        g.fillPolygon(tempHead);

    }

  private static int getY(int length, double direction) {return (int)(length *
Math.cos(direction));}

  private static int getX(int length, double direction) {return (int)(length *
Math.sin(direction));}

  /**
   * Resets the link status to its defaults
   */
   @Override
  public void resetToDefault() {
     super.resetToDefault();
     this.color = Color.LIGHT_GRAY;
  }
}
```

DAFTAR PUSTAKA

Anonim, Creating a GUI with JFC Swing, Sun Microsystems Inc ., http://java.sun.com/docs/books/tutorial/uiswing/index.html, 2004.

A Algorithm Tutorial, http://www.geocities.com/jheyesjones/astar.html*.

Dobson, Simon. Weighted graphsand shortest paths, UCD School of Computer Science and Informatics, Dublin, 2006.

Munir, R., Matematika Diskrit, PT. Informatika Bandung, Bandung, Indonesia, 2003.

Patrick Lester, A Path finding for Beginners, http://www.gamedev.net/reference/articles/article2003.asp, 2003*.

Policyalmanac. "A Star Pathfinding for Beginners." 2010. <http://www.policyalmanac.org/games>.

Rinaldi Munir, M.T., D*iktat Kuliah IF2251 Strategi Algoritmik*, Lab. Ilmu dan Rekayasa Komputasi Departemen Teknik Informatika Institut Teknologi Bandung, 2005.

Siang, J. J., Matematika Diskret dan Aplikasinya pada Ilmu Komputer, PT. Andi Offset, Yogyakarta, Indonesia, 2006.