8 Puzzle

Documentation Part 2

Zander Labuschagne
23585137

Table of contents

1. Algorithm 1
2. References 1
Appendix A: Source Code 2

1. Algorithm

The algorithm start every move by generating all possible moves that's not already made to arrive at the current state, this avoids loops. The algorithm then filter the child nodes or all possible moves generated to only the ones that is determined by the heuristic to be a "good move". The heuristic determines this by calculating the amount of tiles out of place, except for the blank space and adding that value to the distance from the current state to the start state, the node(s) or move(s) with the lowest value are considered good moves. This is repeated until a solution is found, if there are no more possible moves to be made that was not made before the solution is reached, the puzzle is then considered to be unsolvable.

This algorithm generates a tree of nodes or possible moves where the root of the tree denotes the start state and the children nodes, all the possible moves considered as good moves. If it happens that more than one move is considered as a good move, the nodes representing these moves are kept on the tree while their children is being evaluated, this is repeated until any children nodes are found which is not considered to be a good move, if this happens, those nodes and their predecessors are removed from the tree and only the children and their predecessors are kept on the tree if those children are considered to be good moves. A breadth first algorithm is used to traverse the tree, making it possible to evaluate all child nodes on the same level.

If it happens that one of the nodes considered as a good move was evaluated by the algorithm at an earlier state, then the algorithm choose that node that was found at an earlier state to be the best move among the other good moves being evaluated. All the moves happened between the current state and the previous state where this state was found is then removed to shorten the path or amount of moves to be made.

2. References

Luger, G. F. 2009. Artificial Intelligence: Structures and Strategies for Complex Problem Solving. 6th ed. Mexico City, NM: Pearson Addison Wesley.

Appendix A: Source Code

```java
                    nioCtrlTask.close();
                }
            }catch(IOException ioExceptionCaught) {
                {
                    final Imagination task = new Imagination(this,getClass().getResource("/videos/error.mkv").toExternalForm());
                    task.execute("Failed to open playback file. The error occurred was: "+ioExceptionCaught.getMessage());
                    throw new Error(ioExceptionCaught.getMessage(), ioExceptionCaught);
                }
            }catch (Exception exc)
            {
                final Imagination task = new Imagination(this,getClass().getResource("/videos/error.mkv").toExternalForm());
                task.execute("Unknown error occurred. Please contact the developer immediately.", "Error: "+exc.getClass(), stackTrace.FORMAT_PYTHON);
            }
        }
    }.start();
}

/**
 * Play Movie Event
 * @param evt ActionEvent when to make
 * this is not thread
 */
public static synchronized void playMovie()
{
    new Thread(new Runnable()
    {
        /* This thread can be interrupted, unless it blocks on the
         * Object.wait() method inherited
         */
        public void run()
        {
            try
            {
                /*
                 * Create a new instance of Imagination, which is blocks on the
                 * Object.wait() method inherited.
                 */
                Imagination task = new Imagination(this,getClass().getResource("/videos/error.mkv").toExternalForm());
                task.execute("Failed to open playback file. The error occurred was: "+ioExceptionCaught.getMessage());
                throw new Error(ioExceptionCaught.getMessage(), ioExceptionCaught);
            }catch(IOException ioExceptionCaught) {
                {
                    final Imagination task = new Imagination(this,getClass().getResource("/videos/error.mkv").toExternalForm());
                    task.execute("Unable to open playback file. The error occurred was: "+ioExceptionCaught.getMessage());
                    throw new Error(ioExceptionCaught.getMessage(), ioExceptionCaught);
                }
            }catch (Exception exc)
            {
                final Imagination task = new Imagination(this,getClass().getResource("/videos/error.mkv").toExternalForm());
                task.execute("Unknown error occurred. Please contact the developer immediately.", "Error: "+exc.getClass(), stackTrace.FORMAT_PYTHON);
            }
        }
    }.start();
}

/**
 * Gets the new location for moved hire
 * @param hire The hire that's moved
 * @return The new location
 */
@SuppressWarnings("unused")
public int[] calculateNewLocation(int hire, int x, int y)
{
    int[] hell, heigh;

    if(x == 0)
    {
        hell[0] = 0;
    }else if(x < 0)
    {
        hell[0] = -1;
    }else if(x > 0)
    {
        hell[0] = 1;
    }else
    {
        hell[0] = 0;
    }

    if(y == 0)
    {
        heigh[0] = 0;
    }else if(y < 0)
    {
        heigh[0] = -1;
    }else
    {
        heigh[0] = 0;
    }

    hire=hire+hell[0]+heigh[0];
    return hell;
}

/**
 * Get the devices on their correct positions
 */
@Override public void doInBackground()
{
    int x = new.getMouse().getX();
    int y = new.getMouse().getY();
    this.setLocation(x,y);
    this.setTitle("");

    x = new.getMouse().getX();
    y = new.getMouse().getY();
    this.setLocation(x,y);
    this.setTitle("");

    x = new.getMouse().getX();
    y = new.getMouse().getY();
    this.setLocation(x,y);
    this.setTitle("");

    x = new.getMouse().getX();
    y = new.getMouse().getY();
    this.setLocation(x,y);
    this.setTitle("");

    x = new.getMouse().getX();
    y = new.getMouse().getY();
    this.setLocation(x,y);
    this.setTitle("");

    this.repaint(false);
}

/**
 * Move a hire
 * @param hire Which hire to move
 */
public void moveHire(int hire)
{
    if(hire!=0)
    {
        if(hire>0)
        {
            for(int i=0;i<hire;i++)
            {
                this.setLocation(this.getX()+1,this.getY());
            }
        }else
        {
            for(int i=0;i>hire;i--)
            {
                this.setLocation(this.getX()-1,this.getY());
            }
        }
    }
}

/**
 * Move a hire upwards
 * @param hire Which hire to move
 */
public void moveHireUp(int hire)
{
    if(hire!=0)
    {
        if(hire>0)
        {
            for(int i=0;i<hire;i++)
            {
                this.setLocation(this.getX(),this.getY()-1);
            }
        }else
        {
            for(int i=0;i>hire;i--)
            {
                this.setLocation(this.getX(),this.getY()+1);
            }
        }
    }
}

/**
 * Move a hire downwards
 * @param hire Which hire to move
 */
public void moveHireDown(int hire)
{
    if(hire!=0)
    {
        if(hire>0)
        {
            for(int i=0;i<hire;i++)
            {
                this.setLocation(this.getX(),this.getY()+1);
            }
        }else
        {
            for(int i=0;i>hire;i--)
            {
                this.setLocation(this.getX(),this.getY()-1);
            }
        }
    }
}

/**
 * Move a hire to the left
 * @param hire Which hire to move
 */
public void moveHireLeft(int hire)
{
    if(hire!=0)
    {
        if(hire>0)
        {
            for(int i=0;i<hire;i++)
            {
                this.setLocation(this.getX()-1,this.getY());
            }
        }else
        {
            for(int i=0;i>hire;i--)
            {
                this.setLocation(this.getX()+1,this.getY());
            }
        }
    }
}

/**
 * Move a hire to the right
 * @param hire Which hire to move
 */
public void moveHireRight(int hire)
{
    if(hire!=0)
    {
        if(hire>0)
        {
            for(int i=0;i<hire;i++)
            {
                this.setLocation(this.getX()+1,this.getY());
            }
        }else
        {
            for(int i=0;i>hire;i--)
            {
                this.setLocation(this.getX()-1,this.getY());
            }
        }
    }
}

/**
 * Move a hire towards
 * @param hire Which hire to move
 */
public void moveHireTowards(int hire)
{
    if(hire!=0)
    {
        if(hire>0)
        {
            for(int i=0;i<hire;i++)
            {
                this.setLocation(this.getX()+1,this.getY()+1);
            }
        }else
        {
            for(int i=0;i>hire;i--)
            {
                this.setLocation(this.getX()-1,this.getY()-1);
            }
        }
    }
}

/**
 * Move a hire towards
 * @param hire Which hire to move
 */
public void moveHireAway(int hire)
{
    if(hire!=0)
    {
        if(hire>0)
        {
            for(int i=0;i<hire;i++)
            {
                this.setLocation(this.getX()-1,this.getY()-1);
            }
        }else
        {
            for(int i=0;i>hire;i--)
            {
                this.setLocation(this.getX()+1,this.getY()+1);
            }
        }
    }
}

/**
 * New Hobbie name Stars Mars Hire
 * @param hire
 * @throws Throwable
 */
protected void finalize(MoveHireHireHireHireHireHireHireHire a)
{
    if(hire!=0)
    {
        hire=0;
    }
    hire = new Hire(Hire,Mars,Hire);
    super.finalize();
}

/**
 * Set Hire when window is focused
 * @param a
 */
private void HireWindowGainedFocus(WindowEvent a)
{
    a.setHire(hire);
}

/**
 * Moves the current game state and load a new state from hire
 * @param a
 * @throws Throwable
 */
protected void shuffle_loadNewStateFromHireTheNetActionPerf a)
{
    new.dispose();
    int count = 0;
    while(count < (HireCount.nMaxNumOfHire))
    {
        new.loadHire(Hire.getHire(count).getHireInfo());
        count++;
    }
}

/**
 * Saves the current game state to hire
 * @param a
 */
protected void shuffle_SaveNewStateFromHireTheNetActionPerf a)
{
    int count = 0;
    while(count < (HireCount.nMaxNumOfHire))
    {
        Hire.getHire(count).setHireInfo(new.getHireInfo());
        count++;
    }
}

/**
 * Moves the current game state and Hire
 * @param a
 * @throws Throwable
 */
protected void shuffle_FlyTheNetHireTheNetActionPerf a)
{
    new.fire();
    fire.hire();
}

/**
 * Attempt Move This A
 * @param a
 * @throws InterruptedException
 */
private void HireNetHireTheNetActionPerf a)
{
    new.Move();
    HireNetHire();
    new.getHire();
}

/**
 * Attempt Move This A
 * @param a
 */
private void HireNetHireTheNetActionPerf a)
{
    new.Move();
    HireNetHire();
    new.getHire();
}

/**
 * Attempt Move This A
 * @param a
 */
private void HireNetHireTheNetActionPerf a)
{
    new.Move();
    HireNetHire();
    new.getHire();
}

/**
 * Attempt Move This A
 * @param a
 */
private void HireNetHireTheNetActionPerf a)
{
    new.Move();
    HireNetHire();
    new.getHire();
}

/**
 * Attempt Move This A
 * @param a
 */
private void HireNetHireTheNetActionPerf a)
{
    new.Move();
    HireNetHire();
    new.getHire();
}

/**
 * Attempt Move This A
 * @param a
 */
private void HireNetHireTheNetActionPerf a)
{
    new.Move();
    HireNetHire();
    new.getHire();
}

/**
 * Attempt Move This A
 * @param a
 */
private void HireNetHireTheNetActionPerf a)
{
    new.Move();
    HireNetHire();
    new.getHire();
}
```

```
/**
 * Initialize Music Title A
 * @param e
 */
private void InitAbilityMusicTitleA(EventObject e)
{
    SpawnMusic();
    SetupMusicCallback();
    PlayLightMusic();
}

/**
 * Process current game state and file
 * @param e
 */
public void ProcessFileEvent(EventObject e)
{
    SaveMusic();
    SystemLoaded();
}

/**
 * Update about Playing
 * @param e
 * @return ShowLightSet
 */
private void EnableOffMusicTitleAMusicPlayer(e)
{
    return event = new EmptyFile();
    SpawnLoaded();
    SpawnLightEnableMusicPartFile();
    SpawnLightEnableMusic();
    return event;
}

private void EnableOffMusicTitleAMusicPlayer(e)
{
    new Thread(new Runnable()
    {
        public void run()
        {
            try
            {
                Thread.sleep(5000);
            }
            catch (Exception ex)
            {
                final ImageIcon icon = new ImageIcon(this.getClass().getResource("/com/alert/icon.png"));
                JOptionPane.showMessageDialog(null, ex.getMessage() + "\nPlease contact the developer immediately.", "Error Help", JOptionPane.ERROR_MESSAGE, icon);
            }
        }
    }).start();
    LightMusic.SpawnLoaded();
    return event;
}

private void SpawnLightEnableMusicPartFile(e) {
    // TODO add your code here
}

private void SpawnLightEnableMusicPartFile(e) {
    // TODO add your code here
}

private void SpawnLightEnableMusicPartFile(e) {
    // TODO add your code here
}

private void EnableOffMusicTitleAMusicPlayer(e)
{
    Sample = true;

    for (int i = 0; i < this.getSize(); i++) {
        Object o = this.getElementAt(i);
        if (o instanceof Sample) {
            // Sample.setSelected(true);
            Sample.setSelected(true);
            // Sample.setSelected(true);
            // Sample.setSelected(true);
            //
```

```
    }
}
/**
 * @author Xavier Valero-Pasqau Diedhiou
 * Email to Xavier.Valero.Pasqau@gmail.com
 * This software was designed to be independent of GUI and other software
 */

import java.util.Random;
/*
import java.swing.JOptionPane;
import java.awt.Color;
import
javax.swing.event.ListDataListener;
import javax.swing.ListModel;
*/
import java.lang.Exception;
import java.lang.Integer;
import java.lang.ArrayIndexOutOfBoundsException;

public class BattleMap
{
    //Instance Variables
    /**
     * The starting state of the game presented as a matrix
     */
    private int[][] startmtx;
    /**
     * The current state of the game
     * Initialized with startmtx
     */
    private int[][] currentstate;
    /**
     * Game grid is presented as the origin map
     * Just behavior for more details.
     */
    private BattleMap origin;
    /**
     * Game grid is presented as the empty space
     */
    private BattleMap battleMap;
    /**
     * Integer that contains the amount of hours made
     * in the game
     */
    private int countHours;
    /**
     * Matrix to be used to determine end game
     */
    private int[][] endmatrix;
    /**
     * Object of me class to handle all this me
     */
    private MC gameHandler;
    /**
     * String containing text when correctly/incorrectly played
     */
    private String textMessenger;
    /**
     * int containing text number stored
     */
    private int textNum;
    /**
     * Number of rows/blocks contained in the x-axis
     */
    private int x_axisBlocks;
    /**
     * Number of rows/blocks contained in the y-axis
     */
    private int y_axisBlocks;

    /**
     * Default constructor
     */
    public BattleMap()
    {
        this(null, null);
    }
    /**
     * Overloaded constructor
     * @param startmtx the matrix on which the game should start, presented as matrix format
     * @param endmatrix the state on which the game is won, presented as matrix format
     */
    public BattleMap(int[][] startmtx, int[][] endmatrix)
    {
        int[][] tempMatrix;
        int[][] tempMatrix2;
        int[][] tempMatrix3;
        if(startmtx != null)
        {
            setNewStartmatrix();
        }
        else
        {
            startmtx = new int[][]{{1,1,1,1,1,1,1},{1,0,0,0,0,0,1},{1,0,0,0,0,0,1},{1,0,0,0,0,0,1},{1,1,1,1,1,1,1}};
            setNewStartmatrix();
        }
        if(endmatrix != null)
        {
            tempMatrix2 = endmatrix;
            setNewEndmatrix();
        }
        else
        {
            endmatrix = new int[][]{{1,1,1,1,1,1,1},{1,0,0,0,0,0,1},{1,0,0,0,0,0,1},{1,0,0,0,0,0,1},{1,1,1,1,1,1,1}};
            setNewEndmatrix();
        }
        countHours = 0;
        gameHandler = new MC();
        textNum = 0;
        textMessenger = "";
    }
    /**
     * Get the empty map
     * @return BattleMap Default sized test data structure Presented as the form of a two-based Rows
     * @param origin the game object which should be set/considered as the origin game
     */
    public void setOrigin(BattleMap origin)
    {
        this.origin = origin;
    }
    /**
     * Get the test map
     * @return BattleMap the game occupying the empty block
     * @param battleMap the game object which should be set/considered as the empty block/code
     */
    public void setNewBattleMap(BattleMap battleMap)
    {
        this.battleMap = battleMap;
    }
    private void setX_axisInt(int x)
    {
        x_axisBlocks = x;
    }

    private void setY_axisInt(int y)
    {
        y_axisBlocks = y;
    }
    /**
     * Default constructor
     * Overloaded constructor Deeply cloned test data structure
     * @param startmtx the matrix to be used to determine end game
     * @return BattleMap the starting state of the game
     */
    public int[][] setNewStartmatrix()
    {
        try
        {
            this.startmtx = startmtx;
            int[][] tempMatrix = new int[startmtx.length][];
            int[][] tempMatrix2 = new BattleMap[startmtx.length][startmtx[0].length];
            for (int x = 0; x < startmtx.length; x++)
            {
                tempMatrix[x] = new int[startmtx[x].length];
                for (int y = 0; y < startmtx[x].length; y++)
                {
                    tempMatrix[x][y] = startmtx[x][y];
                    tempMatrix2[x][y] = new BattleMap(tempMatrix, x, y);
                }
            }
            this.battleMap = tempMatrix2[0][0];
            this.x_axisBlocks = startmtx.length;
            this.y_axisBlocks = startmtx[0].length;
            return tempMatrix;
        }
        catch(ArrayIndexOutOfBoundsException aio)
        {
            JOptionPane.showMessageDialog(null, aio.toString() + "\nPlease contact the developer: ddeadsolar@gmail.com", "Error: Could not setNewStartmatrix()", JOptionPane.ERROR_MESSAGE, null);
            return null;
        }
    }
    /**
     * Sets a predefined state to be built as a data structure
     * Used as initial game state
     */
    public int[][] setNewEndmatrix()
    {
        try
        {
            BattleMap[][] tempMatrix = new BattleMap[startmtx.length][startmtx[0].length];
            int[][] tempMatrix = new int[startmtx.length][startmtx[0].length];
            int x = 0;
            for (int x = 0; x < startmtx.length; x++)
            {
                for (int y = 0; y < startmtx[x].length; y++)
                {
                    if(x == endmatrix.length || endmatrix[x].length <= y)
                    {
                        tempMatrix[x][y] = new BattleMap(tempMatrix, x, y);
                    }
                    else
                    {
                        tempMatrix[x][y] = endmatrix[x][y];
                    }
                }
            }
            for (int x = 0; x < startmtx.length; x++)
                for (int y = 0; y < startmtx[x].length; y++)
                    if(x==y) {
                        x++;
                    }
            endmatrix = tempMatrix;
            return tempMatrix;
        }
        catch(ArrayIndexOutOfBoundsException aio)
        {
            JOptionPane.showMessageDialog(null, aio.toString() + "\nPlease contact the developer: ddeadsolar@gmail.com", "Error: Could not setNewEndmatrix()", JOptionPane.ERROR_MESSAGE, null);
            return null;
        }
    }
    /**
     * Updates the matrix to be used as a reference throughout the game
     * Should be called after every move
     */
    public void updateStartmtx()
    {
        int[][] x = new int[getX_axis()][getY_axis()];
        for (int x = 0; x < getX_axis(); x++)
            for (int y = 0; y < getY_axis(); y++)
                x[x][y] = getBattleAt(x, y).getPrintVal();
        setStartmtx(x);
    }
    /**
     * sets the current state of the game after step or initialization
     * @param startmtxNew the matrix to configure or update the data structure
     */
    public void setStartmtx(int[][] startmtxNew){ set up a state according to the array
        setNewStartmatrix(startmtx);
    }
    /**
     * sets the winning state of the game to determine end game
     * @param endmtx the matrix to be used to compare with the active game state to determine end game.
     */
    public void setEndmatrix(int[][] endmtx)
    {
        this.endmtx = endmtx;
    }
    /**
     * sets the textNum
     * @param textNum the text done
     */
    public void setTextNum(int textNum)
    {
        this.textNum = textNum;
    }
    /**
     * sets the text that made the text done
     * = text_0, etc
     * @param text
     */
    public void setTextMsg(int text)
    {
        textMsg = text;
    }
    /**Accessors
     * Get the amount of hours made
     * @return int amount of hours made
     */
    public int getCountHours()
    {
        return countHours;
    }
    /**
     * get the origin map reference to find or reference other nodes
     * @return the origin node reference
     */
    public BattleMap getOrigin()
    {
        return origin;
    }
    /**
     * get the empty space node
     * @return the empty space node reference
     */
    public BattleMap getBattleMap()
    {
        return battleMap;
    }
    /**
     * Request a node by value
     * @param value to search for the node value
     * @return BattleMap node that has requested node indexed here
     * @throws node reference containing the requested value
     */
    public BattleMap getNodeInt value)
    {
        BattleMap ptr = getOrigin();

        try
        {
            for(int i = 0; i < getX_axis() && ptr.getPrintVal() != value; i++)
                for(int j = 0; j < getY_axis() && ptr.getPrintVal() != value; j++)
                    ptr = ptr.getNodeAt(i, j);
            return ptr;
        }
        catch(ArrayIndexOutOfBoundsException aio)
        {
            JOptionPane.showMessageDialog(null,"The node with the queried value does not exist.\ngetNode(int value) Returned null");
            return null;
        }
        catch(NullPointerException npe)
        {
            try
            {
                JOptionPane.showMessageDialog(null, ptr.getMessage() + "\nPlease contact the developer: ddeadsolar@gmail.com", "Get current node value", JOptionPane.ERROR_MESSAGE, null);
            }
            catch(NullPointerException npe2)
            {
                JOptionPane.showMessageDialog(null, "\nPlease contact the developer: ddeadsolar@gmail.com", "Error: BattleMap.getNodeInt value)", JOptionPane.ERROR_MESSAGE, null);
            }
            return ptr;
        }
    }
    /**
     * Request a node by coordinate
     * @param x requested x-axis coordinate
     * @param y requested y-axis coordinate to find requested node
     * @return BattleMap node reference to the requested node
     * @throws node reference to the requested node
     */
    public BattleMap getNodeAt(int x, int y)
    {
        BattleMap ptr = getOrigin();

        try
        {
            if(x < 0 && y >= 0)
                return ptr;

            else if(x < getX_axis() - 1 || y < getY_axis() - 1) { x = 0; y = 0;
                //Call the BattleMap.toString() for the node at the queried coordinates does not exist in the given range.\nBattleMap.getNodeAt(int x, int y) returned null");
                return null;
            }
            else
                for(int i = 0; i < x; i++)
                {
                    for(int j = 0; j < y; j++)
                    {
                        ptr = ptr.getNodeAt(1);
                        System.
                        if(ptr == null)
                        {
                            ptr = ptr.getNodeAt(1);
                            System.
                        }
                    }
                }
            return ptr;
        }
        catch(ArrayIndexOutOfBoundsException aio)
        {
            try
            {
                JOptionPane.showMessageDialog(null, ptr.getMessage() + "\nPlease contact the developer: ddeadsolar@gmail.com", "Node coordinates out of range", JOptionPane.ERROR_MESSAGE, null);
            }
            catch(NullPointerException npe)
            {
                JOptionPane.showMessageDialog(null, "\nPlease contact the developer: ddeadsolar@gmail.com", "Error: BattleMap.getNodeAt(int x, int y)", JOptionPane.ERROR_MESSAGE, null);
            }
            return ptr;
        }
    }
    /**
     * get the underlying state
     * @return matrix representing the set game
     */
    public int[][] getStartmatrix()
    {
        return startmtx;
    }
    /**
     * get the initial/start state of the game
     * @return matrix which represent the current state
     */
    public int[][] getCurrentstate()
    {
        return currentstate;
    }
    /**
     * get the current game state
     * @return matrix which represent the current game state
     */
    public int[][] getCurrentstate()
    {
        return currentstate;
    }
    /**
     * get the last done dweet's value
     * @return string containing last move value
     */
    public String getLastMoveDirection()
    {
        return lastMoveDirection;
    }
    /**
     * get the last input dweet's number
     * @return integer representing the last dweet's number
     */
    public int getLastMove()
    {
        return textNum;
    }
    /**
     * get the x-axis index
     * @return int
     */
    private int getX_axis()
    {
        return x_axisBlocks;
    }
    /**
     * get the y-axis index
     * @return
     */
    private int getY_axis()
    {
        return y_axisBlocks;
    }
    /**
     * Matrix representation of the current game state in string format
     * @return string formatted matrix representation of the game
     */
    public String toString()
```

```
        string state = "";
        for (int y = 0; y < gOH_point1; y++)
        {
            for (int x = 0; x < gOW_point1; x++)
                state += (gOwnStrokeMatch[gOH_point1 - y - 1, x] != 0 ? "■" : Strings.Format("■", gOwnStrokeMatch[gOH_point1 - y - 1, x]));
            state += "\r\n";
        }

        return state;
    }

    /// <summary>Returns a string representation of the current game state to be used for write/output to .csv file</summary>
    /// <returns>String formatted index separated value representation of current game state</returns>
    public String toCSV()
    {
        string state = "";
        for(int y = 0; y < gOH_point1; y++)
            for (int x = 0; x < gOW_point1; x++)
                state += (gOwnStrokeMatch[gOH_point1 - y - 1, x] != 0 ? "■" : Strings.Format("■", gOwnStrokeMatch[gOH_point1 - y - 1, x]) + (x != (gOW_point1 - 1) && x != (gOW_point1 - 1) ? "," : "");

        return state;
    }

    /// <summary>Returns a string representation of the current game state to be used for write/output to .csv file</summary>
    /// <returns>String formatted index separated value representation of current game state</returns>
    public String toCSVNoIndex()
    {
        string state = "";
        for(int y = 0; y < gOH_point1; y++)
            for (int x = 0; x < gOW_point1; x++)
                state += (gOwnStrokeMatch[gOH_point1 - y - 1, x] != 0 ? "■" : Strings.Format("■", gOwnStrokeMatch[gOH_point1 - y - 1, x]) + (x != (gOW_point1 - 1) && x != (gOW_point1 - 1) ? "," : "");

        return state;
    }

    /// <summary>Prints the current game state to console in matrix format</summary>
    public void printGameStateMatrix()
    {
        System.out.println(toString() + "\n");
    }

    /// <summary>
    /// Determines if game over
    /// </summary>
    /// <returns>true if game over, else false</returns>
    public boolean gameOver()
    {
        boolean b = true;

        for(int x = 0; x < gOW_point1; x++)
            for (int y = 0; y < gOH_point1; y++)
                if (gOwnStrokeMatch[y, x] != 0)
                    b = false;

        return b
```

```
                    arrOneNumber[i] = "0";
                }
                var[][] array = new int[get_size()][get_size()];
                int x = 0;
                for(int i = 0; i < get_size(); i++) {
                    for(int j = 0; j < get_size(); j++) {
                        array[i][j] = Integer.parseInt(arrOneNumber[x]);x++;}}
                var[][] arrNode = new int[get_size()][get_size()];
                int k1 = 0;
                for(int i = get_size() - 1; i >= 0; i--) {
                    for(int j = 0; j < get_size(); j++) {
                        arrNode[i][j] = Integer.parseInt(arrOneNumber[k1]);k1++;}}
                node.clear();
                node.add(new Node(0));
                createSource(source);
                int[][] arrayOfSource = new ImageController().getPBMData("size/info5.pbm");
                createImage(arrayOfSource, "Game loaded from " + fileName + "\nEnjoy and don't ever give up on anything.", AlphaTheme.BUTTERSCOTCH_TRIANGLE, icon);
            }catch(FileNotFoundException fnfe){
                final ImageIcon icon = new ImageController().getPBMData("size/error.pbm")[0];
                JOptionPane.showMessageDialog(null, "Error!\nFile not found", JOptionPane.ERROR_MESSAGE, icon);
            }catch(ArrayIndexOutOfBoundsException fuel){
                final ImageIcon icon = new ImageController().getPBMData("size/error.pbm")[0];
                JOptionPane.showMessageDialog(null, "Non-gifMessage!\n", "Incorrect .ics format", JOptionPane.ERROR_MESSAGE, icon);
            }
        }catch(Exception ex){
            final ImageIcon icon = new ImageController().getPBMData("size/error.pbm")[0];
            JOptionPane.showMessageDialog(null, "Error!\nFile not found\nPlease make sure that you've put the correct format.ics\nIf this should continue to occur, please check the documentation for more information", "Error!\nFile not found", JOptionPane.ERROR_MESSAGE, icon);
        }
    }
    }.start();
}

/**
 * Sets up a destructor for cleaning up possible memoryleaks and cleaning up memory after game has terminated
 * @throws Throwable
 */
@Override
protected void finalize() throws Throwable
{
    super.finalize();
    gameTracker.close();
}
}
/**
 * @author Xander Lehenbauer 2048854
 * Modified Node class to be used as nodes in the puerto gamemode Pingvin's Promise!
 */

public class NodeMayNode
{
    //Instance Variables
    /**
     * Integer containing the value of this node
     */
    int val;
    /**
     * Integer specifying the x coordinate of this node(latitude)
     */
    int x;
    /**
     * Integer specifying the y coordinate of this node(longitude)
     */
    int y;
    /**
     * Reference to the node left adjacent of this node
     */
    NodeMayNode left;
    /**
     * Reference to the node right adjacent to this node
     * can be null
     */
    NodeMayNode right;
    /**
     * Reference to the node upwards adjacent to this node
     * can be null
     */
    NodeMayNode up;
    /**
     * Reference to the node downwards adjacent to this node
     * can be null
     */
    NodeMayNode down;

    //Default Constructor
    public NodeMayNode()
    {
        this(0, 0, 0);
    }

    //Overloaded Constructor #1
    /**
     * @param v1 initializes the value
     * @param x1 coordinate of this node(latitude)
     * @param y1 coordinate of this node(longitude)
     */
    public NodeMayNode(int v1fk, int x1, int y)
    {
        this(v1fk, x, y, null, null, null, null);
    }

    //Overloaded Constructor #2
    /**
     * Initializes
     * @param v1 value of this node
     * @param x1 coordinate of this node
     * @param y1 coordinate of this node
     * @param left reference to the left adjacent node
     * @param right reference to the right adjacent node
     * @param up reference to the upwards adjacent node
     * @param down reference to the downwards adjacent node
     */
    public NodeMayNode(int v1fk, int x1, int y, NodeMayNode left, NodeMayNode right, NodeMayNode up, NodeMayNode down)
    {
        setVal(v1fk);
        setX(x);
        setY(y);
        setLeft(left);
        setRight(right);
        setUp(up);
        setDown(down);
    }

    //Mutators
    /**
     * Sets the value of this node
     * @param val value of this node
     */
    public void setVal(int val)
    {
        this.val = val;
    }
    /**
     * Sets the x coordinate of this node
     * @param x coordinate of this node
     */
    public void setX(int x)
    {
        this.x = x;
    }
    /**
     * Sets the y coordinate of this node
     * @param y coordinate of this node
     */
    public void setY(int y)
    {
        this.y = y;
    }
    /**
     * Sets the reference to the node left adjacent to this node
     * @param left reference to the left adjacent node
     */
    public void setLeft(NodeMayNode left)
    {
        this.left = left;
    }
    /**
     * Sets the reference to the node right adjacent to this node
     * @param right reference to the right adjacent node
     */
    public void setRight(NodeMayNode right)
    {
        this.right = right;
    }
    /**
     * Sets the reference to the node upwards adjacent to this node
     * @param up reference to the upwards adjacent node
     */
    public void setUp(NodeMayNode up)
    {
        this.up = up;
    }
    /**
     * Sets the reference to the node downwards adjacent to this node
     * @param down reference to the downwards adjacent node
     */
    public void setDown(NodeMayNode down)
    {
        this.down = down;
    }

    //Accessors
    /**
     * Gets the value of this node
     * @return value of this node
     */
    public int getVal()
    {
        return val;
    }
    /**
     * Get the x coordinate of this node
     * (latitude)
     * @return x coordinate of this node
     */
    public int getX()
    {
        return x;
    }
    /**
     * Get the y coordinate of this node
     * (longitude)
     * @return y coordinate of this node
     */
    public int getY()
    {
        return y;
    }
    /**
     * Get the reference to the node left adjacent to this node
     * @return reference to the node left adjacent to this node
     */
    public NodeMayNode getLeft()
    {
        return left;
    }
    /**
     * Get the reference to the node right adjacent to this node
     * @return reference to the node right adjacent to this node
     */
    public NodeMayNode getRight()
    {
        return right;
    }
    /**
     * Get the reference to the node upwards adjacent to this node
     * @return reference to the node upwards adjacent to this node
     */
    public NodeMayNode getUp()
    {
        return up;
    }
    /**
     * Get the reference to the node downwards adjacent to this node
     * @return reference to the node downwards adjacent to this node
     */
    public NodeMayNode getDown()
    {
        return down;
    }
    /**
     * Determines if this node is the origin node
     * @return true if this node is the origin node
     */
    public boolean isOrigin()
    {
        return (x == 0 && y == 0);
    }
    /**
     * Determines if this node is the exit node
     * @return true if this node is the exit node
     */
    public boolean isExitNode()
    {
        return (getVal() != 0);
    }
}
/**
 * @author Xander Lehenbauer 2048854
 * Class that implements the Runnable interface
 * Each instance was designed to be independent of GUI
 * This class is the MAIN version for NodeMayThreadedMultiThreading
 */

public class NodeMayThreadedMultiThreadedRunnableMay
{
    //Default Constructor
    /**
     * Default Constructor
     */
    public NodeMayThreadedMultiThread()
    {
        super();
    }

    //Overloaded Constructor
    /**
     * @param source file name in which the game should start, proportional to matrix format
     * @param exitNode The name in which the game is over, proportional to matrix format
     */
    public NodeMayThreadedMultiThread(int[][] starter, int[][] exitNode)
    {
        super(starter, exitNode);
    }

    //Runnable Method
    /**
     * Reads the current game state of requested
     * Writes the current game state in index-separated value format into a .ics file to be loaded in future
     * @param source file name in which the game should start the .ics will be saved
     * @param fileName file name to save game as
     * @throws
     */
    public void saveStateToString(fileName)
    {
        new Thread(new Runnable()
        {
            public void run()
            {
                saveStateToStringFileName();
            }
        }).start();
    }
    //Runnable Method
    /**
     * Loads a requested game state from a .ics in index-separated value format
     * @param source file name in which the game should start, proportional to matrix format
     * @throws exit to which the exitNode of the .ics file to load
     * @throws FileNotFoundException if the file is not found
     * @param fileName name of file to save game as
     */
    @Override
    public void loadStateToString(fileName)
    {
        new Thread(new Runnable()
        {
            public void run()
            {
                loadStateToStringFileName();
            }
        }).start();
    }
}
/**
 * @author Xander Lehenbauer 2048854
 * Modified Node Class to support 64 capabilities
 */

import java.util.LinkedList;

public class NodeMayNode64 extends NodeMay
{
    private LinkedList<NodeMayNode64> list;
    private int leftDistance;
    private int rightDistance;
    private int[][] arrOfNodes;
    private int x;
    public int primaryNode;
    public NodeMayNode64 parent;

    //Default Constructor
    public NodeMayNode64()
    {
        this(0, 0, 0, 
```

```
        numberOfNodes = 0;
        head = nullptr;
        // numberOfNodes = head;
    }
    /**
     * Get the array of bool valid states of children
     */
    public bool isMeshValidNodeMapNode childAt()
    {
        bool isAllChildrenAt();
    }
    /**
     * Returns all the children of a node
     */
    public bool hasChildren()
    {
        bool // new LinkedListNodeMeshMapNode[];
    }
    /**
     * Increment 1 the Heuristic estimate
     */
    public int getF()
    {
        return f;
    }
    /**
     * @return array of valid bools
     */
    public int[] getHasAMesh()
    {
        return isMeshNode;
    }
    /**
     * @return array of children nodes
     */
    public LinkedListNodeMeshMapNode getNext()
    {
        return next;
    }
    /**
     * Returns the head of the node
     */
    public int getHead()
    {
        return head;
    }
}
/**
 * Sorts the sorters function that moves the node forming the number
 * @brief a node element in the list node and be moved into the next node's position, by swapping this node with the next node
 * @param node node which is requested to move
 */
@Override
public void heuristic(node)
{
    MeshNode node = getNextNode();
    if(node.getF() != null && node.getF().isNotNode())
    {
        node.getF().node.getF().isNotNode();
    }
    if(node.getF() != null && node.getF().isNotNode() && node.getF().isNotNode())
    {
        node.getF().node.getF().isNotNode();
    }
    if(node.getF() != null && node.getF().isNotNode())
    {
        node.getF().node.getF().isNotNode();
    }
}
/**
 * Sorts the sorters function that moves the node forming the number
 * @brief a node element in the list node and be moved into the next node's position, by swapping this node with the next node
 * @param node node which is requested to move
 */
@Override
public void heuristic(node)
{
    MeshNode node = getNextNode();
    if(node.getF() != null && node.getF().isNotNode())
    {
        node.getF().node.getF().isNotNode();
    }
    if(node.getF() != null && node.getF().isNotNode() && node.getF().isNotNode())
    {
        node.getF().node.getF().isNotNode();
    }
    if(node.getF() != null && node.getF().isNotNode())
    {
        node.getF().node.getF().isNotNode();
    }
}
/**
 * Move the node a block left, by swapping this node with the next node
 * @brief move a block to the left node by node adjacent to this node
 * @param node node which is requested to move
 */
@Override
public void moveLeft(MeshNode node)
{
    if(node.getF() != null)
    {
        node.getF().node.getF().isNotNode();
        node.getF().node.getF().isNotNode();
        node.getF().node.getF().isNotNode();
    }
    if(node.getF() != null)
    {
        node.getF().node.getF().isNotNode();
        node.getF().node.getF().isNotNode();
        node.getF().node.getF().isNotNode();
    }
    if(node.getF() != null && node.getF().getF() != null)
    {
        node.getF().node.getF().isNotNode();
        node.getF().node.getF().isNotNode();
        node.getF().node.getF().isNotNode();
    }
    if(node.
```

```
    }
}
/**
 * Main Key node a third down, to randomly fire node with the call node
 * Key method is only in call the a node with activation to the node.
 * Only method is only in call the a node with
 * @param node node which is required to send
 */
protected void doinNodeFire(Node node) {
    {
        if (node.getLeft() != null)
        {
            node.setLeftFlag(getLeft().getFlag());
            node.getLeft().addRight(node);
        }
        node.getLeft().getRight().setRightFlag(node.getFlag());
        node.getRight().setLeft(node.getLeft().getRight());
        if (node.getRight() != null)
        {
            node.setRight(node.getRight().getRight());
            node.getRight().setLeftFlag(node);
        }
        node.getRight().getLeft().setLeft(node.getLeft());
        node.getLeft().getRight().setRight(node.getRight());
        if (node.getFlag() != 1 && (!m_tabs.get(node.getFlag()).equals("") && !m_tabs.get(node.getFlag()).equals(" ")))
        {
            node.setNextNode(getNode(node.getFlag()));
        }
        if (node.getLeft() != null)
        {
            node.setLeft(node.getLeft());
        }
        if (node.getRight() != null)
        {
            node.setRight(node.getRight());
        }
    }
    if (node.getFlag() != 1 && !m_tabs.get(node.getFlag()).equals(" "))
    {
        node.setFlag(1);
    }
    if (node.getFlag() != 1)
    {
        node.setFlag(1);
    }
    node.setNextNode(0);
    node.setRight(null);
    node.setLeft(null);
    if (getArgument().size() > 0)
    {
        node.setFlag(1);
    }
    node.setNextNode(0);
    if (getArgument().size() > 0)
    {
        node.setFlag(1);
    }
    node.setNextNode(0);
    node.setRight(null);
    node.setLeft(null);
    if (node.getFlag() != 1)
    {
        node.setFlag(1);
    }
    node.setNextNode(0);
    node.setRight(null);
    node.setLeft(null);
    if (node.getFlag() != 1)
    {
        node.setFlag(1);
    }
    node.setNextNode(0);
    node.setRight(null);
    node.setLeft(null);
    if (getArgument().size() > 0
```