Artificial Intelligence
CS4242 ONLINE
Menelio Alvarez

Programming Assignment #2.2

8 Puzzle Problem

To create A program to solve the * Puzzle Problem I used four classes:

- **PuzzleBox Class:** Used to store information about the puzzle box. Every time a move was generated A new Puzzle Box object was created.
- **AStarSearch Class:** Used to preform the actual A* search through the different moves.
- **GUI Class:** Used to display the A* search as it worked.
- **PuzzleBoxPane Class:** Used to store information relevant to displaying the PuzzleBox object in the GUI.
- **PuzzleBoxUtils Class:** This class was used to perform several helpful task such as generate a random state for the Puzzle Box

PuzzleBox Class

```
package eightTile;

import java.util.Arrays;

public class PuzzleBox {
      //array of char for tiles, kept in order from position 1 to 9 ('*'= blank
tile)
      private char[] tiles;
      //array of puzzle boxes for the moves
      private PuzzleBox[] moves;
      //distance from goal, value of h(n)
      private int hn;
      //distance from start, value of g(n)
      private int gn;

      /**<h1>No Args Constructor</h1>
       * <p>
       * @postioncondition             : an instance of the PuzzleBox class has
       * been instantiated with random tiles.
       * */
      public PuzzleBox() {
            this.tiles = util.PuzzleBoxUtils.genTiles();
            this.hn = util.PuzzleBoxUtils.distanceToGoal(this.tiles);
            //no args constructor would only be used to create root PuzzleBox so
gn=0
            this.gn=0;
      }

      /**<h1> Args Constructor</h1>
       * <p>
       * @param parentsGn         : gn of parent puzzel box
       * @param tiles             : char[] of tiles for this puzzle box
       * @postioncondition             : an instance of the PuzzleBox class has
       * been instantiated with given tiles.
       * */
      public PuzzleBox(char[] tiles, int parentsGn) {
            this.tiles = tiles;
            this.hn = util.PuzzleBoxUtils.distanceToGoal(this.tiles);
            this.gn = parentsGn+1;

      }

      /**<h1>Copy Constructor</h1>
       * Copies PuzzleBox
       * <p>
       * @param PuzzleBox              : Given PuzzleBox to be copied
       * @postcondition                : A copy of given PuzzleBox has been made
       * */
      public PuzzleBox(PuzzleBox toCopy) {
            this.tiles = Arrays.copyOf(toCopy.getTiles(), toCopy.getTiles().length);
            this.hn = util.PuzzleBoxUtils.distanceToGoal(this.tiles);
            this.gn = toCopy.getGn();
      }
```

```java
/**<h1> Generate Moves <h1>
 * Generates child puzzle box nodes based on possible moves, and populates
 * PuzzleBox[] moves
 * <p>
 * @postcondition                : PuzzleBox[] moves has been populated
 * */
public void genMoves() {
        //index of *
        int sIndex=0;
        for(int i=0; i< tiles.length;i++)
                if(tiles[i]=='*')
                        sIndex = i;
        //unmodified tile array for child puzzle mox
        char[] childTiles =Arrays.copyOf(tiles, tiles.length);
        //populate moves with all possible puzzle boxes based on sIndex
        if(sIndex==0) {//space is in top left corner
                //initials move[]
                moves= new PuzzleBox[2];
                //move space to top middle
                childTiles[1] = tiles[0];
                childTiles[0] = tiles[1];
                moves[0]= new PuzzleBox(childTiles, gn);
                //reset tiles
                childTiles = Arrays.copyOf(tiles, tiles.length);;
                //move space to left middle
                childTiles[3] = tiles[0];
                childTiles[0] = tiles[3];
                moves[1]= new PuzzleBox(childTiles, gn);
                //reset tiles
                childTiles = Arrays.copyOf(tiles, tiles.length);;

        }else if(sIndex==1) {//space is in top middle
                //initials move[]
                moves= new PuzzleBox[3];
                //move space to top left corner
                childTiles[0] = tiles[1];
                childTiles[1] = tiles[0];
                moves[0]= new PuzzleBox(childTiles, gn);
                //reset tiles
                childTiles = Arrays.copyOf(tiles, tiles.length);
                //move space to middle
                childTiles[4] = tiles[1];
                childTiles[1] = tiles[4];
                moves[1]= new PuzzleBox(childTiles, gn);
                //reset tiles
                childTiles = Arrays.copyOf(tiles, tiles.length);
                //move space to top right corner
                childTiles[2] = tiles[1];
                childTiles[1] = tiles[2];
                moves[2]= new PuzzleBox(childTiles, gn);
                //reset tiles
                childTiles = Arrays.copyOf(tiles, tiles.length);

        }else if(sIndex==2) {//space is in top right corner
```

```
                //initials move[]
                moves= new PuzzleBox[2];
                //move space to top middle
                childTiles[1] = tiles[2];
                childTiles[2] = tiles[1];
                moves[0]= new PuzzleBox(childTiles, gn);
                //reset tiles
                childTiles = Arrays.copyOf(tiles, tiles.length);
                //move space to right middle
                childTiles[5] = tiles[2];
                childTiles[2] = tiles[5];
                moves[1]= new PuzzleBox(childTiles, gn);
                //reset tiles
                childTiles = Arrays.copyOf(tiles, tiles.length);

        }else if(sIndex==3) {//space is in left middle
                //initials move[]
                moves= new PuzzleBox[3];
                //move space to top bottom left corner
                childTiles[6] = tiles[3];
                childTiles[3] = tiles[6];
                moves[0]= new PuzzleBox(childTiles, gn);
                //reset tiles
                childTiles = Arrays.copyOf(tiles, tiles.length);
                //move space to middle
                childTiles[4] = tiles[3];
                childTiles[3] = tiles[4];
                moves[1]= new PuzzleBox(childTiles, gn);
                //reset tiles
                childTiles = Arrays.copyOf(tiles, tiles.length);
                //move space to top left corner
                childTiles[0] = tiles[3];
                childTiles[3] = tiles[0];
                moves[2]= new PuzzleBox(childTiles, gn);
                //reset tiles
                childTiles = Arrays.copyOf(tiles, tiles.length);

        }else if(sIndex==4) {//space is in middle
                //initials move[]
                moves= new PuzzleBox[4];
                //move space to top middle
                childTiles[1] = tiles[4];
                childTiles[4] = tiles[1];
                moves[0]= new PuzzleBox(childTiles, gn);
                //reset tiles
                childTiles = Arrays.copyOf(tiles, tiles.length);
                //move space to top left middle
                childTiles[3] = tiles[4];
                childTiles[4] = tiles[3];
                moves[1]= new PuzzleBox(childTiles, gn);
                //reset tiles
                childTiles = Arrays.copyOf(tiles, tiles.length);
                //move space to bottom middle
                childTiles[7] = tiles[4];
                childTiles[4] = tiles[7];
                moves[2]= new PuzzleBox(childTiles, gn);
                //reset tiles
                childTiles = Arrays.copyOf(tiles, tiles.length);
                //move space to right middle
                childTiles[5] = tiles[4];
                childTiles[4] = tiles[5];
                moves[3]= new PuzzleBox(childTiles, gn);
                //reset tiles
                childTiles = Arrays.copyOf(tiles, tiles.length);

        }else if(sIndex==5) {//space is in right middle
                //initials move[]
                moves= new PuzzleBox[3];
                //move space to top right corner
                childTiles[2] = tiles[5];
                childTiles[5] = tiles[2];
                moves[0]= new PuzzleBox(childTiles, gn);
                //reset tiles
                childTiles = Arrays.copyOf(tiles, tiles.length);
                //move space to middle
                childTiles[4] = tiles[5];
                childTiles[5] = tiles[4];
                moves[1]= new PuzzleBox(childTiles, gn);
                //reset tiles
                childTiles = Arrays.copyOf(tiles, tiles.length);
                //move space to top bottom right corner
                childTiles[8] = tiles[5];
                childTiles[5] = tiles[8];
                moves[2]= new PuzzleBox(childTiles, gn);
                //reset tiles
                childTiles = Arrays.copyOf(tiles, tiles.length);

        }else if(sIndex==6) {//space is in bottom left corner
                //initials move[]
                moves= new PuzzleBox[2];
                //move space to bottom middle
                childTiles[3] = tiles[6];
                childTiles[6] = tiles[3];
                moves[0]= new PuzzleBox(childTiles, gn);
                //reset tiles
                childTiles = Arrays.copyOf(tiles, tiles.length);
                //move space to left middle
                childTiles[7] = tiles[6];
                childTiles[6] = tiles[7];
                moves[1]= new PuzzleBox(childTiles, gn);
                //reset tiles
                childTiles = Arrays.copyOf(tiles, tiles.length);

        }else if(sIndex==7) {//space is in top middle
                //initials move[]
                moves= new PuzzleBox[3];
                //move space to bottom left corner
                childTiles[8] = tiles[7];
                childTiles[7] = tiles[8];
                moves[0]= new PuzzleBox(childTiles, gn);
```

```
              //reset tiles
              childTiles = Arrays.copyOf(tiles, tiles.length);
              //move space to middle
              childTiles[4] = tiles[7];
              childTiles[7] = tiles[4];
              moves[1]= new PuzzleBox(childTiles, gn);
              //reset tiles
              childTiles = Arrays.copyOf(tiles, tiles.length);
              //move space to bottom right corner
              childTiles[6] = tiles[7];
              childTiles[7] = tiles[6];
              moves[2]= new PuzzleBox(childTiles, gn);
              //reset tiles
              childTiles = Arrays.copyOf(tiles, tiles.length);

       }else if(sIndex==8) {//space is in bottom left corner
              //initials move[]
              moves= new PuzzleBox[2];
              //move space to bottom middle
              childTiles[5] = tiles[8];
              childTiles[8] = tiles[5];
              moves[0]= new PuzzleBox(childTiles, gn);
              //reset tiles
              childTiles = Arrays.copyOf(tiles, tiles.length);
              //move space to right middle
              childTiles[7] = tiles[8];
              childTiles[8] = tiles[7];
              moves[1]= new PuzzleBox(childTiles, gn);
              //reset tiles
              childTiles = Arrays.copyOf(tiles, tiles.length);
       }

}
/**<h1>Get tiles </h1>
 * Returns the tiles of this puzzle box
 * <p>
 * @return tiles                : char[] tiles
 * @postcondition               : tiles have been returned
 * */
public char[] getTiles() {
       return this.tiles;
}

/**<h1>Get Moves </h1>
 * Returns the moves of this puzzle box
 * <p>
 * @return moves                : PuzzleBox[] of moves
 * @postcondition               : moves have been returned
 * */
public PuzzleBox[] getMoves() {
       return this.moves;
}

/**<h1> Get h(n)</h1>
 * Returns iny value of hn
```

```
         * <p>
         * @return hn                   : Return int value of h(n)
         * @postcondition               : value of h(n)
         * */
        public int getHn() {
                return hn;
        }
        /**<h1> Get g(n)</h1>
         * Returns iny value of gn
         * <p>
         * @return gn                   : Return int value of g(n)
         * @postcondition               : value of g(n)
         * */
        public int getGn() {
                return gn;
        }

        /**<h1> Get f(n)</h1>
         * Returns iny value of g(n)+h(n)
         * <p>
         * @return gn                   : Return int value of g(n)
         * @postcondition               : value of g(n)
         * */
        public int getFn() {
                return gn+hn;
        }
}
```

AStarSearch Class

```java
package eightTile;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;

/*Class to preform A* search*/
public class AStarSearch {
    //Current PuzzleBox
    PuzzleBox current;
    //ArrayList of PuzzleBoxs to consider for moves
    ArrayList<PuzzleBox> possibleMoves;

    //bool to indicated if goal has been reached goal reached
    boolean goalReached;

    /**<h1> Constructor </h1>
     * Initializes A* search obj with given puzzle book(should be root
     * <p>
     * @param root                   : First puzzle box considerd in A* search
     * @postcondition A A* search obj has been instatiated with a root puzzle box
     * */
    public AStarSearch(PuzzleBox root) {
        current = root;
        possibleMoves = new ArrayList<PuzzleBox>();
        possibleMoves.add(root);
        current.genMoves();

        possibleMoves.clear();
        //add them to ArrayList of possibleMoves
        for(int i=0; i < current.getMoves().length;i++) {
            possibleMoves.add(current.getMoves()[i]);
        }
        //sort possibleMoves
        for(int i=0; i < possibleMoves.size(); i++) {
            for(int j=0; j < possibleMoves.size(); j++) {
                if(possibleMoves.get(i).getFn() < 
possibleMoves.get(j).getFn() ) {
                    Collections.swap(possibleMoves, i, j);
                }
            }
        }
    }

    /**<h1>step</h1>
     * Finds the next move by finding lowest f(n) and expanding associated puzzle
     * boxes moves, then sorting them into possibleMoves by lowest f(n)
     * <p>
     * @postcondition current and possibleMoves has been updated
     * */
    public void step() {
        //generate moves
        current.genMoves();
        possibleMoves.clear();
        //add them to ArrayList of possibleMoves
        for(int i=0; i < current.getMoves().length;i++) {
            possibleMoves.add(current.getMoves()[i]);
        }
        //sort possibleMoves
        for(int i=0; i < possibleMoves.size(); i++) {
            for(int j=0; j < possibleMoves.size(); j++) {
                if(possibleMoves.get(i).getFn() <
possibleMoves.get(j).getFn() ) {
                    Collections.swap(possibleMoves, i, j);
                }
            }
        }
        //set current to puzzle box with lowest f(n)
        current = possibleMoves.get(0);
        //test if goal is reached
        char[] goal = {'1','2','3','8','*','4','7','6','5'};

        goalReached = Arrays.equals(current.getTiles(), goal);

        //if goal reach check =true, return
        if(goalReached) {
            return;
        }
    }

    /**<h1> Get status </h1>
     * Returns boolean value indicating weather or no goal has  been reached
     * <p>
     * @param goalReached                  : bool indicating status of A* search
     * @postcondition                      : bool indicating status of search has
been returned
     * */
    public boolean getStatus() {
        return goalReached;
    }

    /**<h1>Get Current</h1>
     * Retruns current Puzzle Box A* search if on
     * <p>
     * @return current                     :The current Puzzle Box
     * @postcondition                      :A Puzzle Box has been returned
     * */
    public PuzzleBox getCurrent() {
        return current;
    }
}
```

GUI

```
package eightTile;
//Course: CS4242
//Student name: Menelio Alvarez
//Student ID: 000874829
//Assignment #: 2.2
//Due Date: September 13, 2019
//Signature: _________________
//Score: _________________
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.ScrollPane;
import javafx.scene.layout.AnchorPane;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;

public class GUI extends Application {
       //position in anchor pane of root puzzle box node
       double Y = 10.0;
       double X = 585.0;

       //moves count
       int mc =0;

       //Label
       Label goal = new Label("GOAL REACHED");
       @Override
       public void start(Stage stage) throws Exception {
              //Root PuzzleBox and A* search object
              PuzzleBox rootPb =new PuzzleBox();
              AStarSearch search = new AStarSearch(rootPb);

              //Array of PuzzleBoxPanes for moves
              PuzzleBoxPane[] pbPane= new PuzzleBoxPane[( (rootPb.getFn()*4)
+rootPb.getGn() + 1)];

              //Pain to display tree
              AnchorPane aPane = new AnchorPane();
              aPane.setMinSize(1500, 1000);

              //scrollbar
           ScrollPane scroller = new ScrollPane(aPane);
              AnchorPane.setRightAnchor(scroller, 5.0);

              //GridPane for controls and button
              GridPane control = new GridPane();
              Button gen = new Button("Genrate Puzzle");
              Button step = new Button("Step");
              control.add(gen, 0, 0);
              control.add(step, 0, 1);
```

```
        //gen button event
        gen.setOnAction(e->{
            PuzzleBoxPane rootPbPane= new PuzzleBoxPane(rootPb,"Root");
            AnchorPane.setTopAnchor(rootPbPane.getPuzzleBoxPane(), Y);
            AnchorPane.setLeftAnchor(rootPbPane.getPuzzleBoxPane(), X);
            aPane.getChildren().add(rootPbPane.getPuzzleBoxPane());
            Y= Y+200;
            X=10;
            for(int i=0; i < rootPb.getMoves().length;i++) {
                pbPane[mc] = new
PuzzleBoxPane(rootPb.getMoves()[i],"Move"+(i+1));
                AnchorPane.setTopAnchor(pbPane[mc].getPuzzleBoxPane(), Y);
                AnchorPane.setLeftAnchor(pbPane[mc].getPuzzleBoxPane(),
X);
                aPane.getChildren().add(pbPane[mc].getPuzzleBoxPane());
                mc++;
                X=X+200;
            }
        });

        //step button event
        step.setOnAction(e->{

            if(!search.getStatus()) {

                search.step();

                Y= Y+200;
                X = 585.0;
                pbPane[mc] =  new
PuzzleBoxPane(search.getCurrent(),"Choosen Move");
                AnchorPane.setTopAnchor(pbPane[mc].getPuzzleBoxPane(), Y);
                AnchorPane.setLeftAnchor(pbPane[mc].getPuzzleBoxPane(),
X);
                aPane.getChildren().add(pbPane[mc].getPuzzleBoxPane());
                mc++;

                if(!search.getStatus()) {
                    Y= Y+200;
                    X=10;
                    search.getCurrent().genMoves();
                    for(int i=0; i <
search.getCurrent().getMoves().length;i++) {
                        pbPane[mc] = new
PuzzleBoxPane(search.getCurrent().getMoves()[i],"Move"+(i+1));

    AnchorPane.setTopAnchor(pbPane[mc].getPuzzleBoxPane(), Y);

    AnchorPane.setLeftAnchor(pbPane[mc].getPuzzleBoxPane(), X);

    aPane.getChildren().add(pbPane[mc].getPuzzleBoxPane());
                        mc++;
                        X=X+200;
                    }
                }else {
```

```
					Y= Y+80;
					X= X+200;
					AnchorPane.setTopAnchor(goal, Y);
					AnchorPane.setLeftAnchor(goal, X);
					aPane.getChildren().add(goal);
				}
			}

		});


		//set up stage
		Scene scene = new Scene(new BorderPane(scroller, aPane,null,  null, 
control),1500, 1000);
		stage.setScene(scene);
		stage.show();

	}

	public static void main(String[] args) {
		Launch(args);

	}

}
```

PuzzleBoxPane

```
package eightTile;

import javafx.scene.control.Label;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.GridPane;

/*Class for creating Puzzle Box in gPanes
 * */
public class PuzzleBoxPane {

	//Global variable
	//image of tiles
	private Image i1 = new 
Image("file:/C:/Users/Manny/Desktop/KSU/2019_Artificial%20Intelligence/Assignment%202
/JavaWorkSpace/Assignment2/src/Assest/1.png");
	private ImageView v1 = new ImageView(i1);
	private Image i2 = new 
Image("file:/C:/Users/Manny/Desktop/KSU/2019_Artificial%20Intelligence/Assignment%202
/JavaWorkSpace/Assignment2/src/Assest/2.png");
	private ImageView v2 = new ImageView(i2);
	private Image i3 = new 
Image("file:/C:/Users/Manny/Desktop/KSU/2019_Artificial%20Intelligence/Assignment%202
/JavaWorkSpace/Assignment2/src/Assest/3.png");
	private ImageView v3 = new ImageView(i3);
	private Image i4 = new 
Image("file:/C:/Users/Manny/Desktop/KSU/2019_Artificial%20Intelligence/Assignment%202
/JavaWorkSpace/Assignment2/src/Assest/4.png");
	private ImageView v4 = new ImageView(i4);
	private Image i5 = new 
Image("file:/C:/Users/Manny/Desktop/KSU/2019_Artificial%20Intelligence/Assignment%202
/JavaWorkSpace/Assignment2/src/Assest/5.png");
	private ImageView v5 = new ImageView(i5);
	private Image i6 = new 
Image("file:/C:/Users/Manny/Desktop/KSU/2019_Artificial%20Intelligence/Assignment%202
/JavaWorkSpace/Assignment2/src/Assest/6.png");
	private ImageView v6 = new ImageView(i6);
	private Image i7 = new 
Image("file:/C:/Users/Manny/Desktop/KSU/2019_Artificial%20Intelligence/Assignment%202
/JavaWorkSpace/Assignment2/src/Assest/7.png");
	private ImageView v7 = new ImageView(i7);
	private Image i8 = new 
Image("file:/C:/Users/Manny/Desktop/KSU/2019_Artificial%20Intelligence/Assignment%202
/JavaWorkSpace/Assignment2/src/Assest/8.png");
	private ImageView v8 = new ImageView(i8);
	private Image iB = new 
Image("file:/C:/Users/Manny/Desktop/KSU/2019_Artificial%20Intelligence/Assignment%202
/JavaWorkSpace/Assignment2/src/Assest/B.png");
	private ImageView vB = new ImageView(iB);
	
	//gridPane inner and outer
	private GridPane innerPane = new GridPane();
	private GridPane outerPane = new GridPane();
```

```
//label
private Label label;

/**<h1> Constructor </h1>
 * Create Puzzle Box Pane bases on given puzzle box
 * <p>
 * @param puzzleBox                 : PuzzleBox to be represent in Pane
 * @param name                             : String name of puzzle box(number)
 * @postcondition                   : A pane representing the given
 * */
public PuzzleBoxPane(PuzzleBox pb, String name) {
    //setup pane
    innerPane.setGridLinesVisible(true);
    //tiles index
    int k = 0;
    //nested for loops to fill inner Pane
    for(int i=0; i < 3; i++) {

        for(int j=0; j < 3; j++) {
            //set 1 tile
            if(pb.getTiles()[k]=='1') {
                innerPane.add(v1, j, i);
            }
            //set 2 tile
            if(pb.getTiles()[k]=='2') {
                innerPane.add(v2, j, i);
            }
            //set 3 tile
            if(pb.getTiles()[k]=='3') {
                innerPane.add(v3, j, i);
            }
            //set 4 tile
            if(pb.getTiles()[k]=='4') {
                innerPane.add(v4, j, i);
            }
            //set 5 tile
            if(pb.getTiles()[k]=='5') {
                innerPane.add(v5, j, i);
            }
            //set 6 tile
            if(pb.getTiles()[k]=='6') {
                innerPane.add(v6, j, i);
            }
            //set 7 tile
            if(pb.getTiles()[k]=='7') {
                innerPane.add(v7, j, i);
            }
            //set 8 tile
            if(pb.getTiles()[k]=='8') {
                innerPane.add(v8, j, i);
            }
            //set Blank tile
            if(pb.getTiles()[k]=='*') {
                innerPane.add(vB, j, i);
                }
                k++;
            }
        }
        //create Label
        label = new Label("PuzzleBox "+ name +" f(n)="+pb.getFn());
        //place inner in outer
        outerPane.add(innerPane, 0, 0);
        outerPane.add(label, 0, 1);
    }

    /**Get Puzzle Box Pane
     * @return PuzzleBoxPane         : outerPane
     * @postcondition                      : returns outerPane
     * */
    public GridPane getPuzzleBoxPane() {
        return outerPane;
    }
}
```

PuzzleBoxUtils

```java
package util;

import java.util.Random;

public class PuzzleBoxUtils {
    /**<h1> Generate Tiles</h1>
     * Generates a char array of tiles containing the chars
     * 1,2,3,4,5,6,7,8,* in a random order
     * <p>
     * @return tiles           : char[] containing tiles
     * @postcondition          : An array of 9 char has been returned
     * */
    public static char[] genTiles() {

        //starting tiles
        char[] tiles= {'1','2','3','8','*','4','7','6','5'};
        //mix up
        int starIndx=0;
        Random r = new Random();
        //shuffle
        for(int i = 0; i <= 3 ; i++) {
            for(int j =0; j < tiles.length; j++) {
                if(tiles[j]=='*')
                    starIndx=j;
            }
            //top left corner
            if(starIndx==0) {
                if( (r.nextInt((2 - 1) + 1) + 1)== 1 ) {
                    tiles[0] = tiles[1];
                    tiles[1] = '*';
                }else {
                    tiles[0] = tiles[3];
                    tiles[3] = '*';
                }
            }
            //top middle
            if(starIndx==1) {
                if( (r.nextInt((3 - 1) + 1) + 1)== 1 ) {
                    tiles[1] = tiles[0];
                    tiles[0] = '*';
                }else if( (r.nextInt((3 - 1) + 1) + 1)== 2 ) {
                    tiles[1] = tiles[2];
                    tiles[2] = '*';
                }else {
                    tiles[1] = tiles[4];
                    tiles[4] = '*';
                }
            }
            //top right corner
            if(starIndx==2) {
                if( (r.nextInt((2 - 1) + 1) + 1)== 1 ) {
                    tiles[2] = tiles[1];
                    tiles[1] = '*';
        }else {
                tiles[2] = tiles[5];
                tiles[5] = '*';
        }
}
//middle left
if(starIndx==3) {
        if( (r.nextInt((3 - 1) + 1) + 1)== 1 ) {
                tiles[3] = tiles[0];
                tiles[0] = '*';
        }else if( (r.nextInt((3 - 1) + 1) + 1)== 2 ) {
                tiles[3] = tiles[4];
                tiles[4] = '*';
        }else {
                tiles[3] = tiles[6];
                tiles[6] = '*';
        }
}
//middle
if(starIndx==4) {
        if( (r.nextInt((4 - 1) + 1) + 1)== 1 ) {
                tiles[4] = tiles[5];
                tiles[5] = '*';
        }else if( (r.nextInt((4 - 1) + 1) + 1)== 2 ) {
                tiles[4] = tiles[7];
                tiles[7] = '*';
        }else if( (r.nextInt((4 - 1) + 1) + 1)== 3 ) {
                tiles[4] = tiles[3];
                tiles[3] = '*';
        }else {
                tiles[4] = tiles[1];
                tiles[1] = '*';
        }
}
//middle right
if(starIndx==5) {
        if( (r.nextInt((3 - 1) + 1) + 1)== 1 ) {
                tiles[5] = tiles[2];
                tiles[2] = '*';
        }else if( (r.nextInt((3 - 1) + 1) + 1)== 2 ) {
                tiles[5] = tiles[4];
                tiles[4] = '*';
        }else {
                tiles[5] = tiles[8];
                tiles[8] = '*';
        }
}
//bottom left corner
if(starIndx==6) {
        if( (r.nextInt((2 - 1) + 1) + 1)== 1 ) {
                tiles[6] = tiles[3];
                tiles[3] = '*';
        }else {
                tiles[6] = tiles[7];
                tiles[7] = '*';
                }
            }
            //bottom middle
            if(starIndx==7) {
                if( (r.nextInt((3 - 1) + 1) + 1)== 1 ) {
                    tiles[7] = tiles[6];
                    tiles[6] = '*';
                }else if( (r.nextInt((3 - 1) + 1) + 1)== 2 ) {
                    tiles[7] = tiles[4];
                    tiles[4] = '*';
                }else {
                    tiles[7] = tiles[8];
                    tiles[8] = '*';
                }
            }
            //bottom right corner
            if(starIndx==8) {
                if( (r.nextInt((2 - 1) + 1) + 1)== 1 ) {
                    tiles[8] = tiles[5];
                    tiles[5] = '*';
                }else {
                    tiles[8] = tiles[7];
                    tiles[7] = '*';
                }
            }
        }
        return tiles;

}

/**<h1> h(n) Distance from goal state</h1>
 * Will determine distance from goal state by counting how manny tiles
 * are different from goal state. This will server as h(n) in the A*
 * formula f(n)=(g)+h(n)
 * <p>
 * @param current tiles     : tiles to be compared to goal state
 * @return distance              : int distance from goal state
 * @postcondition                : an int reprasenting distance from goal
 * state is returned
 * */
public static int distanceToGoal(char[] tiles) {
        int d=0;
        char[] goal = {'1','2','3','8','*','4','7','6','5'};
        for(int i =0; i <tiles.length;i++) {
            if(tiles[i] != goal[i] && tiles[i] != '*') {
                d++;
            }
        }
        return d;
}

/**<h1> h(n) Distance from start </h1>
 * Will determine distance from starting state by counting how manny tiles
 * are different from starting  state. This will server as g(n) in the A*
 * formula f(n)=(g)+h(n)
```

```
     * <p>
     * @param starting tiles    : tiles of starting state
     * @param current tiles     : tiles to be compared to starting state
     * @return distance              : int distance from goal state
     * @postcondition                : an int reprasenting distance from goal
     * state is returned
     * */
    public static int distanceFromStart(char[] start, char[] tiles) {
        int d=0;

        for(int i =0; i <tiles.length;i++) {
            if(tiles[i]!= start[i]) {
                d++;
            }
        }
        return d;
    }
}
```