August 15nd, 2018
A01630086 Luis Eduardo Vargas Victoria
Intelligent systems
Assignment 1: Uninformed search - BFS

This program gives the solution for a eight-puzzle, it shows the result after the movement in a matrix and the movement of what it made that could be RIGHT, LEFT, UP and DOWN
All of this to get this result:

```
[0, 1, 2]
|3, 4, 5|
[6, 7, 8]
```

Where the program search for the 0 in the array to make the movements

The program is run with an example
This line should be modified if you want a different test:

```
String str = "1 2 5 3 4 0 6 7 8”;
```

How to run the code:

1. Save the code as **`EightPuzzle.java`**
2. Go to terminal and type **`ls`**
3. Navigate to where the file is
4. Type `javac` **`EightPuzzle.java`**
5. Type `java` **`EightPuzzle`**
6. Shows the test

```java
package EightPuzzle;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class Puzzle {
    public List<Puzzle> children = new ArrayList<Puzzle>();
    public static List<String> movements = new ArrayList<String>();
    public Puzzle parent;
    public String move;
    public char[] pieces = new char[9];
    public int zeroPosition = 0;

    public Puzzle(char[] pieces, String move) {
        setPuzzle(pieces);
        movements.add(move);
        this.move = move;

    }

    public boolean goal() {
        boolean isGoal = true;
        if (pieces[0] == '0') {
            if (pieces[1] == '1') {
                if (pieces[2] == '2') {
                    if (pieces[3] == '3') {
                        if (pieces[4] == '4') {
                            if (pieces[5] == '5') {
                                if (pieces[6] == '6') {
                                    if (pieces[7] == '7') {
                                        if (pieces[8] == '8') {
                                            return isGoal;
                                        } else {
                                            isGoal = false;
                                            return isGoal;
                                        }
                                    } else {
                                        isGoal = false;
                                        return isGoal;
                                    }
                                } else {
                                    isGoal = false;
                                    return isGoal;
                                }
                            } else {
                                isGoal = false;
                                return isGoal;
                            }
                        } else {
                            isGoal = false;
                            return isGoal;
                        }
                    } else {
                        isGoal = false;
                        return isGoal;
                    }
                } else {
                    isGoal = false;
                    return isGoal;
                }
            } else {
                isGoal = false;
                return isGoal;
            }
        } else {
            isGoal = false;
            return isGoal;
        }

    }

    public void setPuzzle(char[] pieces) {
        for (int i = 0; i < this.pieces.length; i++) {
            this.pieces[i] = pieces[i];
        }
    }

    public void getPuzzle(char[]a, char[] b) {
        for (int i = 0; i < b.length; i++) {
            a[i] = b[i];
        }
    }

    public void makeMovements() {
        for (int index = 0; index < pieces.length; index++) {
            if (pieces[index] == '0')
                zeroPosition = index;
        }

        if (zeroPosition == 0 || zeroPosition == 1 || zeroPosition == 3 ||
zeroPosition == 4 || zeroPosition == 6 || zeroPosition == 7) {
            char[] newPieces = new char[9];
            getPuzzle(newPieces, pieces);
            //Get piece to be moved
            char movePiece = newPieces[zeroPosition + 1];
            //Move piece zero to the right
            newPieces[zeroPosition + 1] = newPieces[zeroPosition];
            //Place the piece to the left
            newPieces[zeroPosition] = movePiece;

            //Create child with newPieces
            String movement = "Right";
            Puzzle child = new Puzzle(newPieces, movement);
            children.add(child);
            child.parent = this;
        }

        if (zeroPosition == 1 || zeroPosition == 2 || zeroPosition == 4 ||
zeroPosition == 5 || zeroPosition == 7 || zeroPosition == 8) {
            char[] newPieces = new char[9];
            getPuzzle(newPieces, pieces);
            //Get piece to be moved
            char movePiece = newPieces[zeroPosition - 1];
            //Move piece zero to the right
            newPieces[zeroPosition - 1] = newPieces[zeroPosition];
            //Place the piece to the left
            newPieces[zeroPosition] = movePiece;

            //Create child with newPieces
            String movement = "Left";
            Puzzle child = new Puzzle(newPieces, movement);
            children.add(child);
            child.parent = this;
        }

        if (zeroPosition == 3 || zeroPosition == 4 || zeroPosition == 5 ||
zeroPosition == 6 || zeroPosition == 7 || zeroPosition == 8) {
            char[] newPieces = new char[9];
            getPuzzle(newPieces, pieces);
            //Get piece to be moved
            char movePiece = newPieces[zeroPosition - 3];
            //Move piece zero to the right
            newPieces[zeroPosition - 3] = newPieces[zeroPosition];
            //Place the piece to the left
            newPieces[zeroPosition] = movePiece;

            //Create child with newPieces and movement
            String movement = "Up";
            Puzzle child = new Puzzle(newPieces, movement);
            children.add(child);
            child.parent = this;
        }
```

```
        if (zeroPosition == 0 || zeroPosition == 1 || zeroPosition == 2 ||
zeroPosition == 3 || zeroPosition == 4 || zeroPosition == 5) {
            char[] newPieces = new char[9];
            getPuzzle(newPieces, pieces);
            //Get piece to be moved
            char movePiece = newPieces[zeroPosition + 3];
            //Move piece zero to the right
            newPieces[zeroPosition + 3] = newPieces[zeroPosition];
            //Place the piece to the left
            newPieces[zeroPosition] = movePiece;

            //Create child with newPieces
            String movement = "Down";
            Puzzle child = new Puzzle(newPieces, movement);
            children.add(child);
            child.parent = this;
        }
    }

    public void printPuzzle() {
        System.out.println("");
        int m = 0;
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                System.out.print(pieces[m] + " ");
                m++;
            }
            System.out.println("");
        }
    }

    public void printMove() {
        System.out.print(move + " ");
    }

    public boolean samePuzzle(char[] pieces) {
        boolean samePuzzle = true;
        for (int i = 0; i < pieces.length; i++) {
            if(this.pieces[i] != pieces[i]) {
                samePuzzle = false;
            }
        }
        return samePuzzle;
    }

    public static List<Puzzle> breathFirstSearch(Puzzle root){
        List<Puzzle> path = new ArrayList<Puzzle>();
        List<Puzzle> frontier = new ArrayList<Puzzle>();
        List<Puzzle> explored = new ArrayList<Puzzle>();

        //Queue
        frontier.add(root);
        boolean goal = false;
        while (frontier.size() > 0 && !goal) {

            //Dequeue
            Puzzle currentPuzzle = frontier.get(0);
            explored.add(currentPuzzle);
            frontier.remove(0);
            currentPuzzle.makeMovements();
            for (int i = 0; i < currentPuzzle.children.size(); i++) {
                Puzzle currentChild = currentPuzzle.children.get(i);

                if (currentChild.goal()) {
                    System.out.println("It has solution");
                    goal = true;
                    trace(path, currentChild);
                }

                // Checks if the currentChild exists in both if it doesn't add to
frontier
                if(!contains(frontier, currentChild) && !contains(explored,
currentChild)) {
                    frontier.add(currentChild);
                }
            }
        }
        System.out.println("Nodes visited: "+ (frontier.size() +
explored.size()));
        return path;
    }

    public static boolean contains(List<Puzzle> list, Puzzle puzzle) {
        boolean contains = false;
        for (int i = 0; i < list.size(); i++) {
            if (list.get(i).samePuzzle(puzzle.pieces)) {
                contains = true;
            }
        }
        return contains;
    }

    public static void trace(List<Puzzle> path, Puzzle puzzle) {
        Puzzle currentPuzzle = puzzle;
        path.add(currentPuzzle);

        while(currentPuzzle.parent != null) {
            currentPuzzle = currentPuzzle.parent;
            path.add(currentPuzzle);
        }
    }


    public static void main(String[] args) {
        String str = "1 2 5 3 4 0 6 7 8";
        //String str = "1 4 2 6 5 8 7 3 0";
        //String str = "1 8 2 0 4 3 7 6 5";
        String[] splitStr = str.split("\\s+");
        String s = "";
         for (String n:splitStr) {
             s+= n;
         }
         char[] piecesInitial = s.toCharArray();
        Puzzle initPuzzle = new Puzzle(piecesInitial, "root");
        long startTime = System.nanoTime();
        List<Puzzle> solution = breathFirstSearch(initPuzzle);
        Collections.reverse(solution);
        long endTime = System.nanoTime();
        double seconds = (endTime - startTime) / 1000000000.0;
        System.out.println("Cost of the path: "+ (solution.size()));
        System.out.println("Used memory: " + (72 * (solution.size())) + " bytes");
        System.out.println("Running time: "+ seconds + " s");
        if (solution.size() > 0) {

            System.out.print("Path to goal: [");
            for (int i = 0; i < solution.size(); i++) {
                solution.get(i).printMove();
            }
            System.out.print("]");

            for (int i = 0; i < solution.size(); i++) {
                solution.get(i).printPuzzle();
            }
        } else {
            System.out.println("No solution");
        }
    }
}
```

Tests

```
It has solution
Nodes visited: 17
Cost of the path: 4
Used memory: 288 bytes
Running time: 6.19943E-4 s
Path to goal: [root Up Left Left ]
1 2 5
3 4 0
6 7 8

1 2 0
3 4 5
6 7 8

1 0 2
3 4 5
6 7 8

0 1 2
3 4 5
6 7 8
```

```
It has solution
Nodes visited: 247
Cost of the path: 9
Used memory: 648 bytes
Running time: 0.00675553 s
Path to goal: [root Up Left Down Left Up Right Up
Left ]
1 4 2
6 5 8
7 3 0

1 4 2
6 5 0
7 3 8

1 4 2
6 0 5
7 3 8

1 4 2
6 3 5
7 0 8

1 4 2
6 3 5
0 7 8

1 4 2
0 3 5
6 7 8

1 4 2
3 0 5
6 7 8

1 0 2
3 4 5
6 7 8

0 1 2
3 4 5
6 7 8
```

```
It has solution
Nodes visited: 65911
Cost of the path: 22
Used memory: 1584 bytes
Running time: 87.437237389 s
Path to goal: [root Right Up Left Down Down Right
Up Right Up Left Down Left Up Right Right Down Down
Left Up Left Up ]

1 8 2
0 4 3
7 6 5

1 8 2
4 0 3
7 6 5
```

```
1 0 2
4 8 3
7 6 5

0 1 2
4 8 3
7 6 5

4 1 2
0 8 3
7 6 5

4 1 2
7 8 3
0 6 5

4 1 2
7 8 3
6 0 5

4 1 2
7 0 3
6 8 5

4 1 2
7 3 0
6 8 5

4 1 0
7 3 2
6 8 5

4 0 1
7 3 2
6 8 5

4 3 1
7 0 2
6 8 5

4 3 1
0 7 2
6 8 5
```

```
0 3 1
4 7 2
6 8 5

3 0 1
4 7 2
6 8 5

3 1 0
4 7 2
6 8 5

3 1 2
4 7 0
6 8 5

3 1 2
4 7 5
6 8 0

3 1 2
4 7 5
6 0 8

3 1 2
4 0 5
6 7 8

3 1 2
0 4 5
6 7 8

0 1 2
3 4 5
6 7 8
```