Project 1
CSC 2053 - Platform Based Computing
Grading: 150 points
Due Date: Oct. 18, 2017

Part 1 - Solve Puzzle using Breadth First Search (100 points)

Graphs and trees have can represent a multitude of problems. We will create a game tree that solves the "sliding block" problem and traverse this tree using a breadth-first search. This project is meant to reinforce the basic data structures, variables, and methods that you have learned as well as complex notions of queues, trees, graphs, and lists. Use the following template code below as a starting point.

File: Play.java

```
import java.util.*;

public class Play{
  public static List<GameState> explored; //explored stores our marked vertices
  public static Deque<GameState> frontier; //frontier is our queue for BFS
  public static void main(String[] args) {
    int[][] tArr = {{1,5,2},{3,4,0},{6,8,7}}; //start state, i.e. tree root
    GameState start = new GameState(tArr,null); // state with tArr state, null parent

    explored = new ArrayList<GameState>();
    frontier = new ArrayDeque<GameState>();

    //Perform a Breadth First Search here
    //Note: We will not create the graph or enumerate the edges
    //The graph is too large to store in memory
    //We will create the vertices of the graph and edge connections on the fly
    }
}
```

File: GameState.java

```
import java.util.*;

public class GameState{
  public int[][] state; //state of the puzzle
  public GameState parent; //parent in the game tree

  public GameState() {
    //initialize state to zeros, parent to null
  }
  public GameState(int[][] state) {
```

```
  //initialize this.state to state, parent to null
 }
 public GameState(int[][] state, GameState parent) {
  //initialize this.state to state, this.parent to parent
 }
 public GameState swapRight(GameState s, int row, int col) {
  //helper function to swap blank space with right block
 }
 public GameState swapLeft(GameState s, int row, int col) {
  //helper function to swap blank space with left block
 }
 public GameState swapUp(GameState s, int row, int col) {
  //helper function to swap blank space with up block
 }
 public GameState swapDown(GameState s, int row, int col) {
  //helper function to swap blank space with down block
 }
 public boolean isEnd() {
  //helper function to check if the GameState is the end state e.g.
  //0 1 2
  //3 4 5
  //6 7 8
 }
 public ArrayList<GameState> getAdjacent() {
  //Use the swap functions to generate the new vertices of the tree
  //Beware of the boundary conditions, i.e. don't swap left when you are
  //already on the left edge
 }
 @Override
  public boolean equals(Object o) {
     //test that 2 GameStates are equal
  }

@Override
 public String toString() {
  // print out the int[][] array in a 3x3 block e.g.
  //0 1 2
  //3 4 5
  //6 7 8
 }
```

Part 2 - PDF write up (50 points) Create a 1-2 page write up that summarizes the interesting parts of your program. Include any problems or insights that you encountered.

Sample solution of the given problem. Your program should output the solution in a similar fashion.

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

3 1 2
0 4 5
6 7 8

3 1 2
6 4 5
0 7 8

3 1 2
6 4 5
7 0 8

3 1 2
6 0 5
7 4 8

3 1 2
6 5 0
7 4 8

3 1 2
6 5 8
7 4 0

3 1 2
6 5 8
7 0 4

3 1 2
6 5 8
0 7 4

3 1 2
0 5 8
6 7 4

0 1 2
3 5 8
6 7 4

1 0 2
3 5 8
6 7 4

1 5 2
3 0 8
6 7 4

1 5 2

```
3 8 0
6 7 4

1 5 2
3 8 4
6 7 0

1 5 2
3 8 4
6 0 7

1 5 2
3 0 4
6 8 7

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

Deliverables: Submit on Blackboard.