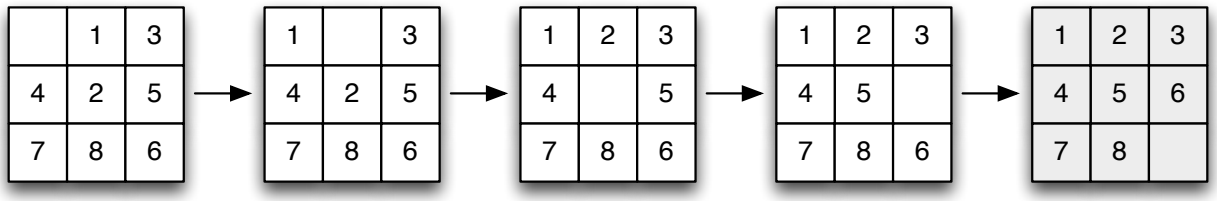
ITIS/ITCS 4180/5180 Mobile Application Development
Homework 3
Date Posted: 09/24/2012 at 02:50am
Due Date: 10/02/2012 at 11:55pm

Basic Instructions:

1. In every file submitted you MUST place the following comments:
   a. Assignment #.
   b. File Name.
   c. Full name of all students in your group.
2. Each group should submit only one assignment. Only the group leader is supposed to submit the assignment on behalf of all the other group members.
3. Your assignment will be graded for functional requirements and efficiency of your submitted solution. You will loose points if your code is not efficient, does unnecessary processing or blocks the UI thread.
4. Please download the support files provided with this assignment and use them when implementing your project.
5. Export your Android project as follows:
   a. From eclipse, choose *"Export…"* from the File menu.
   b. From the Export window, choose *General* then *File System*. Click *Next*.
   c. Make sure that your Android project for this assignment is selected. Make sure that all of its subfolders are also selected.
   d. Choose the location you want to save the exported project directory to. For example, your *Desktop* or *Documents* folder.
   e. When exporting make sure you select *Create directory structure for files*.
   f. Click Finish, and then go to the directory you exported the project to. Make sure the exported directory contains all necessary files, such as the .java and resource files.
6. Submission details:
   a. When you submit the assignment, compress your exported Android project into a single zip file. The format of compressed file name is HW#.zip
   b. You should submit the assignment through Moodle: Submit the zip file.
7. **Failure to follow the above instructions will result in point deductions.**

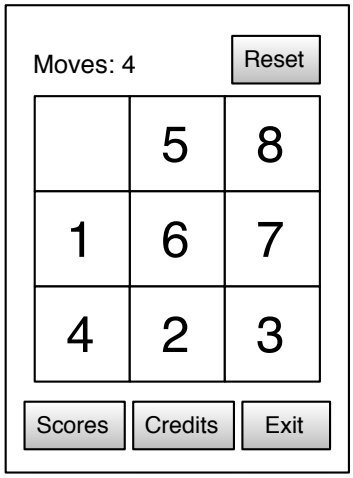
Homework 3 (200 Points)

In this assignment you will develop a 8-puzzle game for Android. The 8-puzzle problem is a puzzle played on a 3-by-3 grid with 8 square blocks labeled 1 through 8 and a blank square. The player's goal is to rearrange the blocks so that they are in order. This is a single player game. The player is permitted to slide blocks horizontally or vertically into the blank square. The following shows a sequence of legal moves from an initial board position (left) to the goal position (right).



Note that it has been proven that not all the initial block permutations are solvable. So it is important when generating a random initial state to ensure that it is solvable. This can be easily done by starting from the goal state, and randomly permuting the empty block using legal moves to generate a solvable initial state.

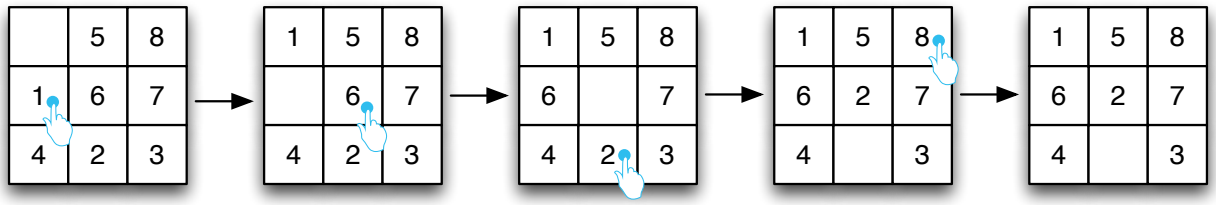
Part 1: Game Activity (150 Points)



The Game Activity contains the main elements of the 8-puzzle game. The Figure shows a wireframe of the game's interface which includes:

1. You are provided with photos to be used for the different number blocks (blockX.png). Please check the PNG files provided. Use ImageViews for displaying these number blocks. You are also provided with a background image (background.png) to be used as the Activity's background. The empty is provided as and empty image (empty.png), use ImageView to display this block. You are also provided with an application image (app_image.png) to be used as your application icon. You should use the provided photos to implement this application.
2. Upon starting the game, the initial board should be generated randomly by starting from the goal board and then randomly permuting the empty block using the legal moves to guarantee the generation of a solvable puzzle. Note that if you simply shuffle the game blocks it is not guaranteed to generate a solvable puzzle, so follow the simple recommended approach. If this operation takes a lot of time, then it should be included in a separate thread or AsyncTask to avoid blocking the UI thread. If you are interested, there is another approach that is based on using parity checking to guarantee the generation of a solvable solution.
3. Upon clicking a block adjacent horizontally or vertically to the empty block, the clicked block should appear in the position of the empty block. No blocks should move if the user clicks a block that is not horizontally or vertically adjacent to the empty block. The figure below shows a demonstration of these requirements. As a Hint, you can

maintain the state of the game in an integer array, which indicates which blocks are stored at which location. For example, the game status array for the below initial state would be {9,5,8,1,6,7,4,2,3}, and also store the index of the empty block which is 0. This will enable you to maintain the game state and to investigate if the clicked ImageView is above, below, to the right, or to the left of the empty block. Also you should store information in the ImageViews using tags to maintain any other required information.

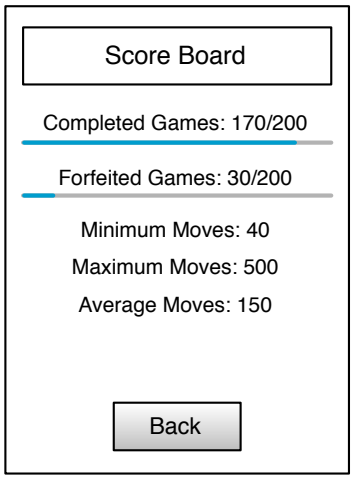


4. Upon reaching the goal state {1,2,3,4,5,6,7,8,9} the user should be notified by displaying a Toast message congratulating the user for reaching the goal state. In addition, any further clicks on the blocks should not move any of the blocks. In other words, the game should not respond to any block on click events after reaching the goal state.
5. The TextView "Moves:" indicates the number of moves performed by the player so far for the current puzzle instance. The "Moves:" should be updated after each move performed by the player.
6. The "Exit" button should quit the Game activity and exits the application.
7. The "Reset" button should reset the game to a new randomly generated initial state. This only resets the game UI but should not reset the score statistics. If the reset is pressed after the game is successfully completed then this game is counted as a successful game. If it the reset is pressed before the completion of the game then the user should be alerted through an AlertDialog indicating that the game will be forfeited, if the user agrees then the game is counted as a forfeited game. If the user does not agree then the user should continue to play the current game.
8. The "Scores" button should show the ScoreBoard activity. Use an explicit intent to show the ScoreBoard activity.

Part 2: ScoreBoard Activity (40 Points)

The ScoreBoard activity shows the overall scores for all the played games. It is called if a user clicks the Scores button. A wireframe of the ScoreBoard interface is indicated in the side figure. The scores and moves should be passed to the ScoreBoard via the putExtra() method. Show the score board based on the provided wireframe.

1. The "Back" button should finish the ScoreBoard activity and take the player back to the Game activity.



Part 3: Credits Activity (10 Points)
The Credits activity shows the names of the project group members. It also has a "Back" button to return to the previous activity. The Credits activity should be started using an implicit Intent object.

Part 4 (Bonus): Both 8-Puzzle and 15-Puzzle (30 Points)
Extend the game to enable the player to choose between playing a 8-Puzzle (3x3) and a 15-Puzzle (4x4). The 15-Puzzle should fit on the same screen and should also randomly generate the initial state. Your code should not simply be a copy paste of the 8-Puzzle game, instead your code should accommodate both puzzle sizes by changing the puzzle size.

Part 5 (Bonus): Gesture Based Game (70 Points)
Extend the game to enable the player to swipe the screen indicating the direction of block to move into the empty block instead of simply clicking on the that block. The figure below shows an example of the required swipe mechanism. You should investigate using the Android GestureDetector class. The GestureDetector should be used to identify the direction of the swipe and accordingly select the block to move and the direction of the move relative to the empty block.