Android: Staff's Choice

n-Puzzle

due by noon ET on Wed 3/7

Ingredients.

- `Activity`
- Android SDK
- `Bitmap`
- `Intent`
- Java

Help.

Help is available throughout the week at `http://help.cs76.net/`! We'll do our best to respond within 24 hours. Be sure, though, to take advantage of lectures and sections as well as videos thereof!

Academic Honesty

All work that you do toward fulfillment of this course's expectations must be your own unless collaboration is explicitly allowed by some project. Viewing, requesting, or copying another individual's work or lifting material from a book, magazine, website, or other source—even in part—and presenting it as your own constitutes academic dishonesty, as does showing or giving your work, even in part, to another student.

Similarly is dual submission academic dishonesty: you may not submit the same or similar work to this course that you have submitted or will submit to another. Nor may you provide or make available your or other students' solutions to projects to individuals who take or may take this course (or CSCI S-76) in the future.

You are welcome to discuss the course's material with others in order to better understand it. You may even discuss problem sets with classmates, but you may not share code. You may also turn to the Web for instruction beyond the course's lectures and sections, for references, and for solutions to technical difficulties, but not for outright solutions to problems on projects. However, failure to cite (as with comments) the origin of any code or technique that you do discover outside of the course's lectures and sections (even while respecting these constraints) and then integrate into your own work may be considered academic dishonesty.

If in doubt as to the appropriateness of some discussion or action, contact the staff.

All forms of academic dishonesty are dealt with harshly.

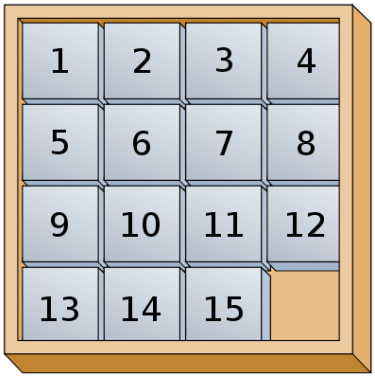
Grades.

Your work on this project will be evaluated along four primary axes.

Correctness. To what extent is your code consistent with our specifications and free of bugs?
Design. To what extent is your code written well (*i.e.*, clearly, efficiently, elegantly, and/or logically)?
Scope. To what extent does your code implement the features required by our specification?
Style. To what extent is your code readable (*i.e.*, commented and indented with variables aptly named)?

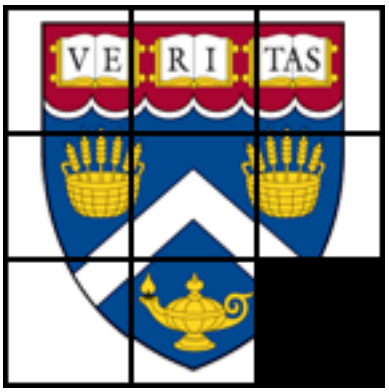
The *n*-puzzle.

The *n*-puzzle is a game known by a variety of names: Game of Fifteen, 8-puzzle, 15-puzzle, Mystic Square, and others. All the names refer to the same game, however: a two-dimensional puzzle with one empty space into which some numbered tiles can slide horizontally or vertically to occupy. The goal is to arrange the board from smallest to largest, as shown in the example below.[1]



A variation of this game is to replace the numbers on the tiles with images cropped from a larger one. The puzzle in a solved state would then appear to be the original image with the corner missing, like the below.

Shuffled

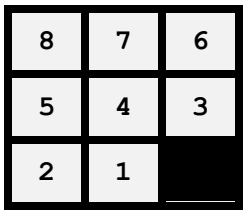


Solved

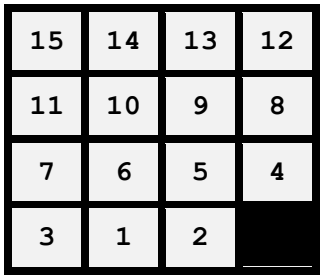
In any case, a valid move is performed when a player slides a tile either horizontally or vertically into the empty space. Any other move, such as a diagonal shift or swapping the place of two arbitrary tiles, is invalid.

[1] Image from: `http://en.wikipedia.org/wiki/File:15-puzzle.svg`

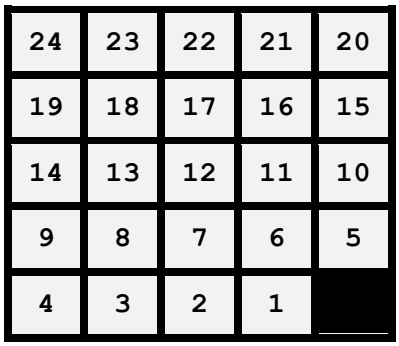
Simply pseudorandomly placing each tile does not guarantee the creation of a shuffled board that is solvable with valid moves. In fact, half of the configurations are found to be impossible![2] To simplify the matters of shuffling, it's perhaps easiest to order the tiles in reverse. However, when *n*, the number of tiles on the board, is odd (don't forget one tile is missing!) then you should swap tiles `1` and `2`. Put another way, if the number of rows and columns is even than tiles `1` and `2` should be swapped. Below are sample boards that are shuffled yet solvable.



8-puzzle



15-puzzle



24-puzzle

Of course, the above applies to the image variant of the *n*-puzzle game, assuming you treat each image tile as a numbered tile.

You might start here if you'd like to do some additional reading on the *n*-puzzle:
`http://en.wikipedia.org/wiki/Fifteen_puzzle`

Armed with this information you can now begin your own implementation of the *n*-puzzle game!

Specification.

☐ By the project's deadline, you'll create an implementation of the *n*-puzzle game as a native Android app. As always, your application must meet some requirements that we've specified below, but its overall design and aesthetics are left up to you. Other unspecified details are left to your own creativity and interpretation.

Features.

- ☐ The game must have three levels of difficulty: "easy", "medium", and "hard". The "easy" level indicates that *n*=8 (in other words, a 3x3 puzzle), "medium" represents *n*=15 or 4x4, and "hard" means *n*=24 or 5x5. The default difficulty should be "medium".
- ☐ Upon opening the application, users must be presented with a list of images included with the application, any one of which will serve as the basis for the *n*-puzzle.

[2] `http://www.jstor.org/stable/2369492`

- ☐ Once the user selects the image, a new activity should appear and display a preview of the solved puzzle using the selected image. That image should be displayed as large as possible on the screen without distorting its aspect ratio and broken up into *n* cropped and equally-sized tiles and displayed with a noticeable border surrounding each. The lowest, right most tile must be blank. To be clear, there should be *n* total tiles in $\sqrt{n+1}$ columns of tiles and $\sqrt{n+1}$ rows of tiles. The quantity *n* depends on the selected difficulty.
- ☐ After three seconds the solution should disappear and, in the same activity, the *n*-puzzle should appear in its place. The puzzle must be made up of the same *n* cropped and equally-sized tiles that appeared briefly, but one of the tiles must remain blank so that the user can perform valid moves. There should again be a noticeable border around each tile so that no two tiles appear to be merged. The *n*-puzzle must be shuffled and must be solvable with valid moves. For simplicity, it's fine if the puzzle's pieces are simply in reverse order as described above. At no time should the puzzle or its pieces change aspect ratio, but the puzzle itself must still remain as large on the screen as possible.
- ☐ To play, a user must be able to tap (or, if using an emulator, click) a tile immediately adjacent (directly on top, to the left, to the right, or below) to the empty space to swap that tile with that empty space. Any other taps or clicks on the puzzle itself must be ignored and should not result in a move.
- ☐ During game play, the user should be allowed to hit the **MENU** button on the Android device (or emulator) to cause a menu to appear and allow the user to reset the puzzle to a shuffled state, change the difficulty, or quit the current game and pick another image. This menu should only appear during game play and not during an image selection.
- ☐ If the user changes the difficulty level it must cause game play to restart. The solution preview must appear and be replaced by the shuffled *n*-puzzle after three seconds, as above. Additionally, the user's preference for difficulty level must be remembered by the app so that the same difficulty is used automatically the next time the game is played. The preference should survive if the user quits and re-opens the app. If no preference is set, the default difficulty should be used.
- ☐ ~~The game's state must also survive if the user quits the app or other activities appear above it. The end result should be that a user can return to the game and continue playing where they left off even if the app is quit or the device is turned off. Don't forget to save such things as the image selection, difficulty, number of moves taken so far, and current tile positions!~~
- ☐ When the user has successfully solved the puzzle, a new activity must appear that congratulates the user on their accomplishment, displays the original image, and lists the number of moves they used while solving it. There must also be a button to return back to the list of images and allow the user to play another game.

Optional Features.

You may also implement one or more of the following additional features. You may safely ignore these if you'd prefer, as you will not have points deducted for not including these. Implementing any will not result in any extra credit. They're simply here for the fun and challenge!

- Ensure that the tiles are arranged pseudorandomly at every shuffle (such as when the game is started or reset) while retaining the puzzle's ability to be solved.[3] This must work for all difficulty levels.
- Allow users to specify a puzzle solution image at runtime via a URL. If the URL is a valid JPG image, the application must download it, store it locally, and add it to the list of images. If the URL is incorrect or the JPG cannot be interpreted the user must be notified.
- Implement a solver that allows a user, by clicking on a new menu option, to watch your app solve the presented puzzle.[4] The solution need not be optimal.

Implementation Details.

- ☐ The application should have a minimum SDK API level of 7 (Android 2.1).
- ☐ You must include at least 3 images of your choosing in the `res/drawable` folder that can be selected as puzzle solutions. The file names must be `puzzle_N.jpg`, where N will begin at 0 and increment sequentially to a maximum value of 9. Keep in mind that we reserve the right to swap out these images or add additional ones (up to a maximum of 10) during testing, so be sure that your tiles are generated on-the-fly by your app and not pre-generated by you and that the quantity of images and sizes of each are not hard-coded. In other words, your image selection activity must dynamically detect the quantity of images and present the user with an appropriate number of choices.
- ☐ Per the features list you should have three activities in this application. The initial list activity should be called `ImageSelection`, the activity where game play occurs should be called `GamePlay`, and the congratulatory activity called `YouWin`. The `.java` files should be named accordingly. You may include other `.java` files if you deem it necessary to implement other classes for your game.
- ☐ Under no circumstances should we be able to cause your program to crash at runtime.
- ☐ Be sure the project and application name are both `nPuzzle########`, where `########` is your 8-digit Harvard ID (HUID), the same credential that you use to log into `help.cs76.net`.
- ☐ Your project's package name should be: `net.cs76.projects.nPuzzle########`
- ☐ You must include your name, email, and HUID in a comment atop every Java source file in the project.

[3] This might help, if you decide to implement the feature: `http://cseweb.ucsd.edu/~ccalabro/essays/15_puzzle.pdf`
[4] As a hint, read up on the A* search algorithm: `http://en.wikipedia.org/wiki/A*_search_algorithm`

How to Submit.

☐ Before the project's due date, export your project in Eclipse for submission. Open Eclipse and click the **File** menu and then **Export**. In the window that appears, click on the triangle next to the **General** section so that you can see the options contained within. Select **Archive File** and click the **Next** button. On the next window, check the box directly to the left of your project you'd like to submit. Be sure no others are selected, or you will export those as well. Click on the **Browse** button to select a location you'd like to export the ZIP file and be sure to name it `########.zip`, where `########` is your 8-digit Harvard ID (HUID), the same credential that you use to log into `help.cs76.net`.

After selecting where the ZIP file will be placed, make sure the export options are correct. Notably that you are saving as a ZIP file (and not tar), that a directory structure is created for files, and that the contents are compressed. When ready, click **Finish** to export your app.

Then head to `https://www.cs76.net/submit`, click the **login** link at top-right, click the link to your TF's dropboxes at top-left, click this project's own folder, click **Upload File**, and upload your ZIP file as prompted; no need to give it a title. Be sure not to click on the wrong TF's dropbox or the wrong project's folder. You may re-submit in this same manner as many times as you'd like. Just take care to delete any prior submissions.

Be sure not to submit or re-submit after this project's deadline.