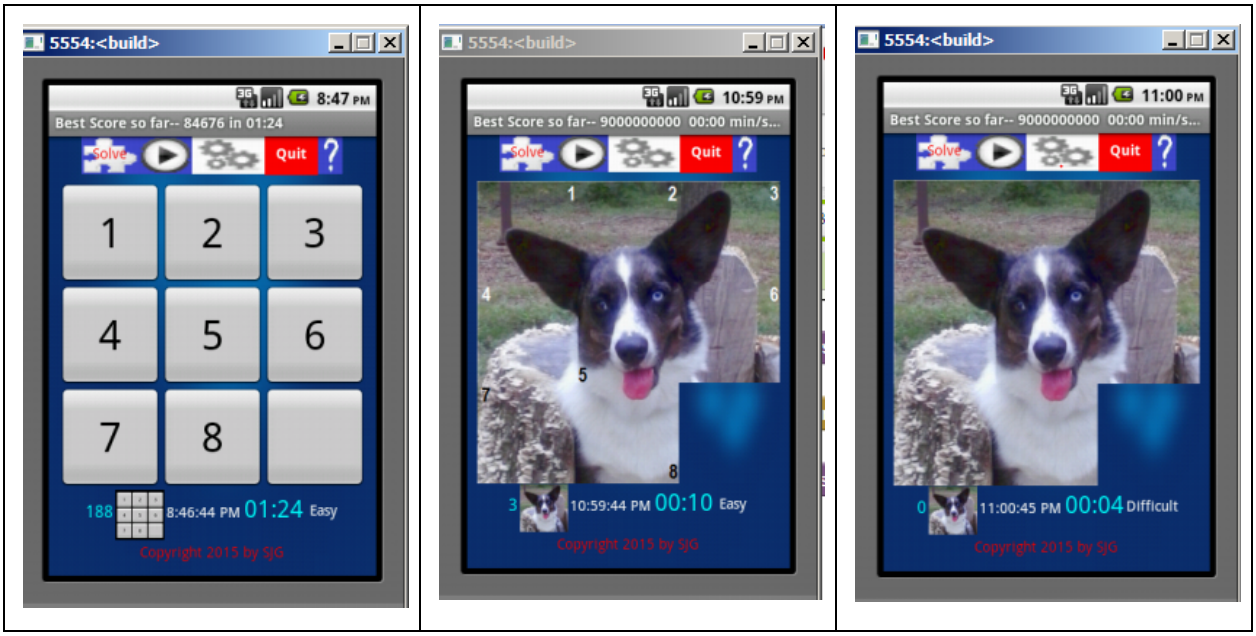
Slide Nine - *a sliding block puzzle game tutorial for App Inventor*

Slide Nine is a timed (and move counted) sliding block puzzle using *Any component* Button Text and Button Image blocks. The example app *Slide Nine* requires the user arrange eight scrambled numerals in a pattern on an Android screen or to unscramble a picture puzzle.

The puzzle blocks on the Android screen appear to slide, touch a block adjacent to the empty slot to trade places with the number or image block next to it. Not all *Slide Nine* puzzles are solvable. Use the Solve button to give up when you have had enough (can not solve a puzzle), then restart the puzzle. Restarting will probably generate a puzzle you can solve. Switch between the numerical and picture puzzle types with the “gear works” menu button. The button sends you to a “screen” (a ListPicker) where you can toggle between the numerical and graphic puzzle options.



Numerical Puzzle | *Magic Easy Puzzle* | *Magic Difficult Puzzle*

Scramble the numbers with the triangle shaped start button, then rearrange them into numerical order by “sliding” the blocks successively into the empty space. You would rather unscramble a picture? Your choice.

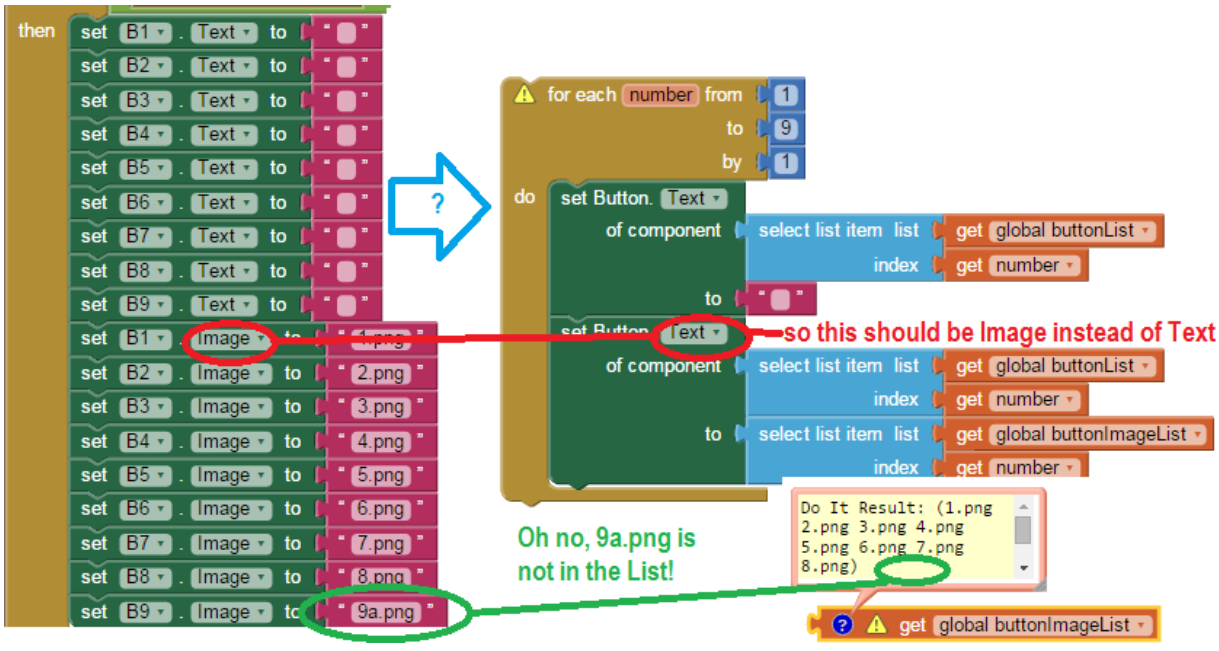
All three *Slide Nine* puzzles are displayed on a single AI2 screen. Displays are changed by hiding/showing layouts.

Program Development

I use the AI2 default object names to label objects in my AI2 projects. Sometimes the sequence of objects is broken; you might find a Label1 and a Label3 but not a Label2 in the final code. I decided not to use an object default name for the *Slide Nine* button 'sliding' objects in the coding of *Slide Nine*. Nine Buttons are named B1 to B9. I needed an awareness of the spatial positions of the buttons and their AI labels to figure out the algorithm to "move" the blocks. How a tile might "move" is restricted by the location of a button within a Table Arrangement. Button B5 is placed in the center Table Layout cell. Button B5 requires the most complex coding to determine what can slide into the space and where a block shown in the button can move. The event handler code of the buttons in the four corners is the least complicated. How each button event handler was coded is described below.

The game logic exists in eighteen Procedures; nine Procedures for the numeral display puzzle, nine similar Procedures for the image puzzles. Other code blocks in the implements a scoring system, orchestrates a shuffling of the "sliding tiles" or blocks, documents the variables, determines what happens when the puzzle is "solved" and provides error control.

Procedures limit redundant code. This tutorial is an exercise to build a game puzzle; if you want to "optimize" the app's code further, there is still opportunity. Here is an example of what could be done to further simplify *Slide Nine* (it was not code,perhaps a project for you):



The *Slide Nine* code snippet on the left could be "simplified" using the *Any component* code blocks on the right if one is VERY careful. The pitfalls of attempting the simplification are

highlighted. Hint: 1) must fix the *Text* to *Image* for the red circled item, 2) when you eliminate the set blocks you will replace on the left, you need to keep the green circled *set B9.Imageto 9a.png* blocks (or you will have issues). Can you find another issue? (Hint: there are only eight items in the *buttonImageList* (there is no reference to B9 in that list). What else in the clever simplify solution above will come back to hurt you? If you fix ALL the issues, the code blocks on the right do what the code on the left does. Try to fix the incomplete example; retain a copy of the code that does work. You will need a copy if you get confused. Sometimes what "looks" messy, might be the best solution. Which code is more readable? Which code is faster? ..and does it matter?

Are you going to build a 15 block puzzle referenced in the section on Algorithms below? The implementation of the code on the right is the way to go.

Slide Nine (or the AI2 Slider Puzzle) is complicated code; the number of blocks might make a novice developer dizzy. This tutorial is primarily the code at the aia source file link. It is not practical to explain all the code blocks, a small book is needed to walk a novice coder through building the app. Writing the tutorial took five times as long as the time required to code the blocks. Really. The functions of the key blocks are explained below; for more insight into the app, peruse the aia file and learn from the block construction.

Housekeeping Blocks

You need a place to store variables:

Game Algorithms

The basic game logic code blocks are placed in the Click event handlers of nine Buttons in a 3 x 3 Table Arrangement. Each 'button' event handler has a slightly different logic. Knowing

which buttons are displaying text adjacent to the button which has no text is the “secret” to successfully writing code to “move” the blocks. The nine buttons are B1..B9. When a button is touched, *App Inventor* searches for an adjacent Table Arrangement cell that contains a button whose text field is empty. If it does not find an empty slot, the button touch does nothing. If an empty slot is located adjacent to the touched button, the text from that button (not the button) transfers to the button that has no text and receives the text from the touched button.

When a slider puzzle is generated completely randomly, exactly 50% of the puzzles will be insoluble. The *Slide Nine* puzzles are generated with AI2’s pseudo-random number generator so expect unsolvable puzzles. You could program the app to allow only solvable puzzles instead of using a random algorithm. I did not attempt to generate a list of solvable distributions.

Each puzzle can be solved in many different ways. Some solutions require more ‘slides’ than others. A move counter is included to keep track of the moves.

Slide Nine is a 3 x 3 puzzle. The original sliding puzzle invented in the 1800’s had 15 blocks on a 4 x 4 grid, was mechanical and much more difficult to solve. You can program a 15 sliding block puzzle too using the techniques here; the larger puzzle just requires more logic and blocks and more patience to code and debug.

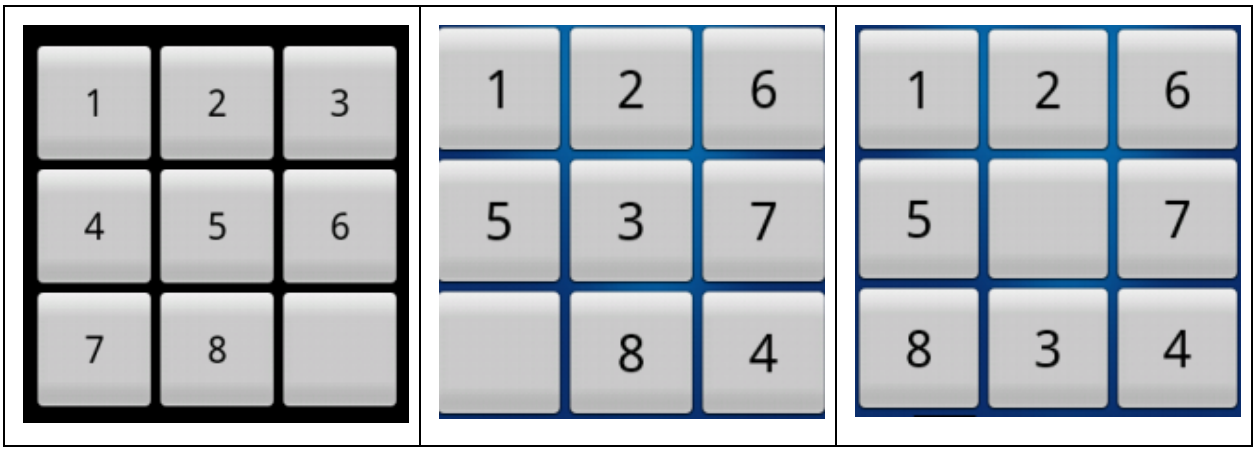
A Little History

Noyes Chapman, a postmaster in Canastota, New York in 1874 invented the first mechanical sliding puzzle. Students in the American School for the Deaf manufactured the puzzle and sold the wooden puzzles in 1879 locally and in Boston, Massachusetts according to *Wikipedia*. Ironic *App Inventor* is maintained by MIT, located in Cambridge, Massachusetts, a Boston “suburb.”

Block Movement

How do the blocks move? Or do buttons move? Or what? The Buttons never move their positions within the Table Layout. The Button.Text changes in the number puzzle. The png image varies as required (i.e. Button.Image changes) in the image puzzles. A significant part of the block “moving” code is possible with the *Any components* Button.Text and Button.Image blocks.

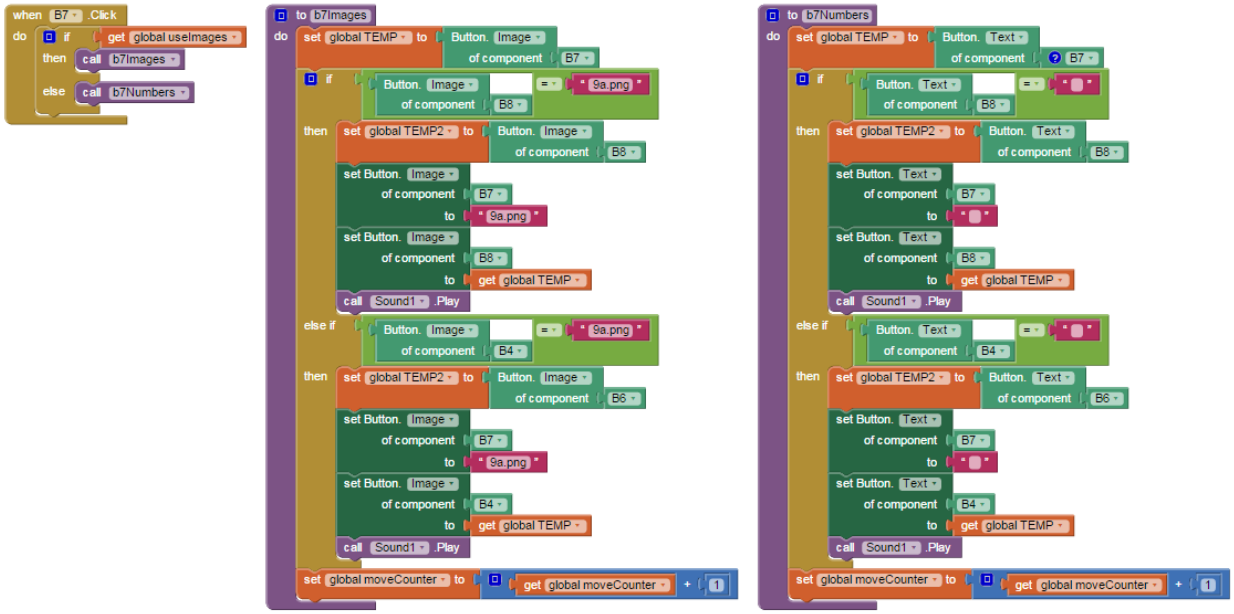
The blocks appear to move because numerical text on Buttons is swapped with the non-text of the ‘empty’ button from the adjacent ‘empty’ button. Likewise, png images are swapped for a blank image.



Unshuffled Buttons *Scrambled lower left corner empty* *Scrambled center empty*

When the numerical puzzle is complete (the Unshuffled Buttons), the lower right corner of the puzzle box is empty, it has no Text numbers. The 8 or the 6 can move into that position. The 5 or the 8 in the first scrambled image above (the one in the center) can 'move' into the empty slot. When the center Table Arrangement cell is 'empty' (it is not really empty, the text of the button is 'empty'), four buttons adjacent to the empty cell display numerical Text, those currently showing 2,5, 7 and 3..

The code blocks needed to determine if the 5 (in button B4) and 8 (in button B8) (the center image) is in *byNumbers* . The numbers 5 and 8 are presently adjacent to an empty block and will move the text from the touched button into the 'empty' block (B7 button). The code blocks in the empty (B7) button and the adjacent buttons work together.



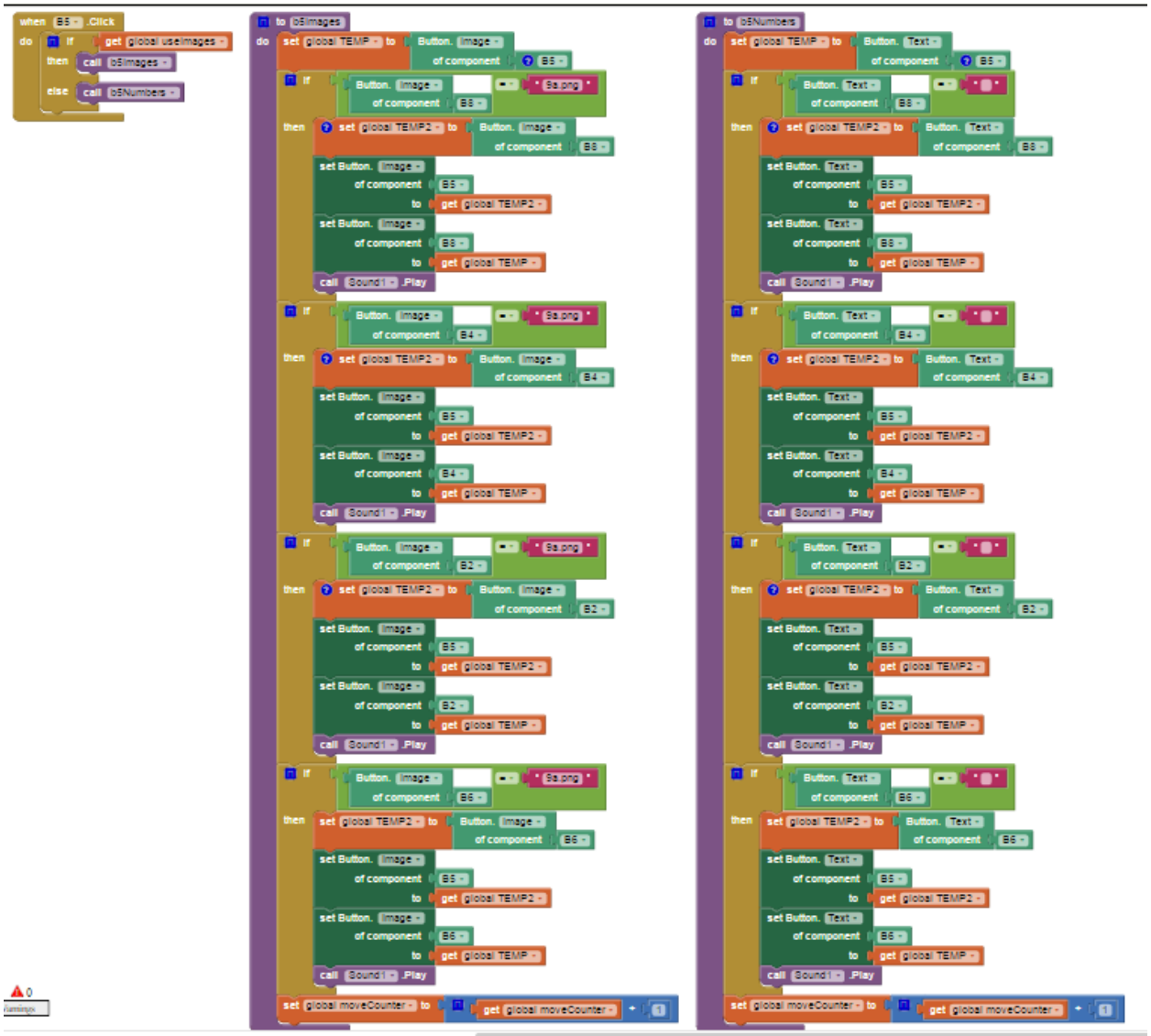
The code in button B7 keeps track of which puzzle is being solved, then selects either the numerical or image puzzle procedures. In this case, a proceduresolves the button relationships for the numerical puzzle (call b7Numbers). If this were an image puzzle the logic would be provided by *call b7Images*.

There are nine puzzle buttons; the code in the Button.Click event is similar for all. The code in the associated procedures is different for each button. All the key procedures follow the form BUTTON_LABEL+Numbers or BUTTON_LABEL+Images.

Is the code images here difficult to read? ALL the code necessary is in the aia file. The tutorial attempts to explain the methodology, you as a developer will fill the gaps by examining the aia.

The empty center block of the puzzle grid shown below is adjacent to four buttons. A slightly different algorithm called in each of the four button event handlers detects the center cell button is empty and the central cell button code recognizes the adjacent blocks contain numeral text. 

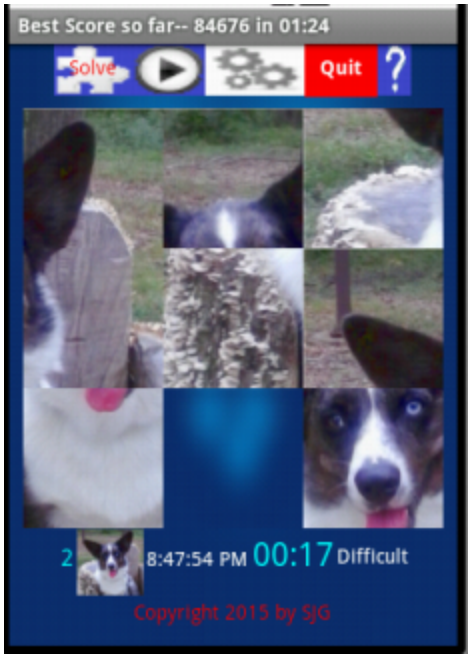
The code blocks that control the central cell of the Table Layout look like this:



The algorithms/procedures used with the numerals to swap text can not work with the puzzle image puzzle. The image puzzle has no text to swap (i.e. the numbers shown on the *Easy* image puzzle are part of the image, not button text). To put *Magic* together again,a second set of procedures is used to track the image png (instead of tracking the text). The procedures *B5Images* and *B5Text* are similar; one procedure uses the *Any component* Button.Image block, the second procedure uses the Button.Text block. The procedures are otherwise identical in structure.

The routines for the remaining tile buttons are shown in the aia file. Right click the block you want to examine and select *Expand Blocks* on these collapsed blocks:

The scrambled *Difficult* image puzzle screen is shown below(Difficult because there are no numerals attached to the eight images of pieces of *Magic's* image). The numerals on the

Easy image puzzle screen (not shown here) help the user arrange the blocks to solve the puzzle. 

The *Slide Nine* Scoring System

We want the fastest time recorded by a user solved a puzzle so far as the score. The **Best Score** is the shortest solving time or lowest value. The puzzle start time is captured (the *Clock.Now*) when a user presses the start button from the MENU at the top of the screen. Pressing the start button also sets other values. When the game is solved (yes, there is logic to determine when the eight 'movable' numbers and images are in their correct solved positions) the app records the time. As soon as the app "realizes" both the start and finish times are available, the Duration (number of ms between the start and finish) is determined. The raw score is the ms required used to solve the current puzzle (1000 ms equals one second).

Slide Nine also records the number of moves (touches of Buttons) The app records even touches to blocks that can not move ...you have to have some kind of penalty for trying the impossible? The counted moves are not part of the score as currently coded. Make the number of moves part of the score in your sliding puzzle app by dividing the ms by the moves to provide a weighted score perhaps?

The **Best** *Slide Nine* score saves to a TinyDB. No leaderboard is necessary. The **Best Score** is displayed the next time *Slide Nine* is loaded (provided you build the apk and load *Slide Nine* on your device).

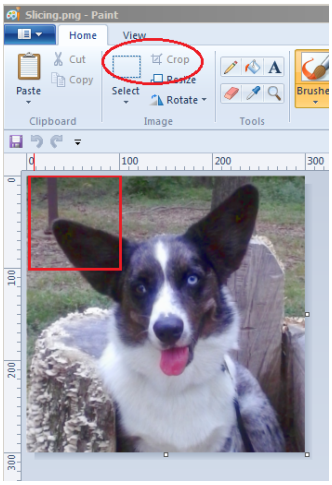
The user can reset the scoring using the "gear works" Menu button at any time.

Random Blocks

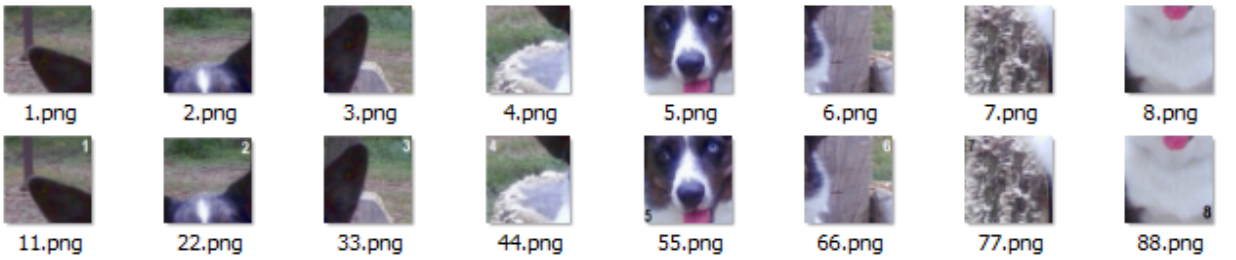
The arrangement of the “puzzle” pieces is different each time the game is played. The random seed is reset when the app is loaded. All puzzles are scrambled by the AI2’s random Math block algorithm We scramble only eight numerical blocks; the lower right corner is not included in the scramble. About half of all puzzles should be solvable. Some puzzles are impossible to solve. If the lower right corner (B9) is included in the scramble, the puzzle is even more difficult. Include the empty block in the shuffle and make finding a solution for the puzzle even more difficult.

How To Make the Images

The puzzle images are created from a 300 x 300 pixel png image sliced into nine pieces. Sophisticated drawing programs can do this using a Slice tool; I use the Windows *Paint* image editor and cut out 100 x 100 images.



The images for the *Difficult* image are named 1.png … 8.png and 9a.png; the images displayed for the *Easy* image are named 11.png...88.png and 9a.png.



The ninth image is a blank 100 x 100 png. The image of *Magic* in this tutorial is the same for both *Easy* and *Difficult*. A numeral is added to the *Easy* image set using the image editor. The numbered blocks makes it easier to solve the graphic slider puzzle.

When you build the app, substitute any images you like for *Magic* (he won’t mind). The images must correspond to the size of the objects in the Table Arrangement. The Table cells

are populated with 100 x 100 pixel images in Buttons. Slice a master 300 x 300 pixel image into nine pieces, discarding the ninth piece. Place the images in the Buttons, not in the Layout cells themselves.

Important Facts

This tutorial and the app are copyrighted. Please do not slightly modify this tutorial and claim it as your own or post it on *Google Play*. Have fun with the puzzle and images for personal use, use the algorithms and ideas in you own app and enjoy coding.

The image of the Cardigan Corgi *Magic* is used with permission of his owners.