15 Piece Analysis - A Look at an Interesting Bug in a gDesklets game.

By: David S. Zimmerman

2010/02/02

Prologue:

I began my understanding of the "15 Piece" puzzle when I met my first Macintosh back in 1984. This little beauty, hid under the Apple logo, relentlessly teased me toward comprehension. Previously, when I had got too frustrated with the physical version, obtained at birthday party or Cracker Jack box, I would simply pop the tiles out and put them in the correct order. This effective strategy springs to mind with all those familiar with James T. Kirk and the Kobayashi Maru scenario. However, seeing that learning the puzzle was more likely to bring me to a solution faster than trying to use the Apple debugger to write my own values into the puzzle tile array... I set myself to finally learn this puzzle from my childhood. Once dedicated, victory was mine!

A Curious Observation:

Decades later, just two days ago, I was happy to again find this little diversion within the gDesklets set of desk applets available with the Gnome windowing environment. I opened it up and proceeded toward the solution until the *impossible* occurred! It did not relinquish the solved state!

As anyone can see from this screen capture, all numbers are in their correct places except the numbers 14 & 15, which are switched. There is no amount of finagling which will place them in the correct position. This becomes evident when one looks at the pre-solved state shown next.

At this point, the top two rows are solved, and the third row is coiled up in the lower left hand corner. All that remains to be done is to rotate the final three (13, 14 & 15) in the lower right corner, around until they are in sequential order with the empty space next to the 12 and the 13 placed next to the 9. The solution then comes from uncoiling the third row, starting with the 12 into the empty space, while bringing the 13-14-15 in after the 9.

As one looks above at the remaining three pieces to spin about in the four available positions, it becomes evident that no amount of rotations will result in a left-to-right sequence of 13-14-15. As this problem gives me additional reason to learn Python (the language that these applets are programmed in), I began to take a look at the source code. It appears that upon first initialization, that the puzzle pieces are simply placed in a random fashion. This insight coupled with the above shown left-to-right sequence of 15-14-13 and a passing thought about enantiomers, and my brain went AHA!!

The puzzle may be thought of as a molecule with two final chiral forms: (1) the solution and (2) the form shown to the left – a mirror image of the solution. A *random* population of the numbers into the grid during reset will result in either one final solved form or the other. Of course, might be tempted to say that this is a feature, not a bug, and change the solution-checking portion to accept the mirror form? Nope. I thought not. It would be much simpler to implement, but certainly *not good*.

Solutions:
How can we provide a randomized and *solvable start state** to the player? If one is determined to use the current randomizer used in the initialization routine then another routine must be devised that can determine the chirality, or handedness, of the eventual solution from this random start state. If a mirror image is detected, then it gets reinitialized until it is solvable. Just thinking about creating this routine, and the complexity of the task, leads me to contemplate other solutions.

This program provides a virtual version of a physical puzzle. The physical puzzle starts solved. Just like a Rubik's Cube, it starts solved, then we mix it up till it looks random, then we solve it. Initialization should start the same way to insure a solvable puzzle. Randomization should mean, in this type of puzzle, application of a series of legal moves, from the solved state, until the field of play appears random. As “appears random” will mean different things to a novice or an expert, a *minimum* requirement that all pieces must be moved from initial state seems fair. We'll call this the Appears Random State (ARS). This means that the empty space must pass at least once over each location.

One way to proceed is to simply make random moves from the solved state until ARS. As the move sequence is random, the number of moves until ARS is indeterminate. To provide determinacy, the space should move to the *next (random) untouched location*. An array of counters may come in handy here to track the number of times a location was touched. Think of the “how many rectangles and squares can you find in this figure” puzzle. Look at our puzzle. Visualize all the rectangles and squares. Any location in the 4x4 playing field may be considered a point that shares in a subset of these rectangles and squares. Any two points will share between them a smaller shared subset of these, each providing a *possible path* that may be traversed *CW* or *CCW* to the next location. Choose the path and direction randomly. We'll call this the Loop method.

If the next location is on, or one off of, a diagonal of the current location, the empty space can be moved by toggling between up, right, down or left as the situation calls for, toward that location. If the space is not on or near a diagonal of the next location, than it can be moved vertically or laterally until it is, and then proceed diagonally. We'll call this the Direct method. Alternating randomly between the Loop and Direct methods in moving the empty space to the next random low/no touched location will provide a robust means of providing a random and solvable starting position in the game.

dsz

---

*At this point I must emphasize how critical it is to supply a puzzle that is solvable ALWAYS. Solving this puzzle is within the grasp of a determined child. Puzzles are brain food. Providing an unsolvable initial state is, in my opinion, like handing a child, whom asked for bread, a rock and told, “Eat”. What this might do to their motivation and world-view if they are able to struggle to the very end and see that it is pointless. That there will be, can be, no victory. Do you want to play again?
I am tempted to walk back through the previous versions over the *years* and see if this puzzle ever actually worked. But this would be pointless except to the maintainers, managers, and financial supporters. I'm not any of those (actually, I have paid for a few distros). I'd rather see a solution first. And soon. Think of the children. The next generation of programmers... or not.