The Basics of the Fifteen Puzzle

The original Fifteen puzzle

For our final mini-project, we will build a program that solves the Fifteen puzzle which was popularized in the 1800's by Loyd. The original version of the Fifteen puzzle consists of fifteen tiles numbered one to fifteen that lie on a 4×4 grid. Since there are sixteen spaces in the grid, one of the spaces is empty.

This version of the puzzle starts in an initial configuration where the tiles numbered one to four lie on the top row (left to right), tiles five to eight lie on the next row down, tiles nine to twelve lie on the third row, and tiles thirteen to fifteen lie on the bottom row with the blank space starting in the lower right corner.

To manipulate the puzzle, the player can slide a tile adjacent to the blank space into the blank space. Repeating this process scrambles the order of the tiles. The challenge in the Fifteen puzzle is to bring the tiles back into its initial configuration.

A modified version of the Fifteen puzzle

Before discussing how to solve the Fifteen puzzle, first, we will make a minor modification to the game design of the puzzle that simplifies reasoning about the game. In the original game, the tiles are numbered one to fifteen with the blank space lying at the lower right. In this version, the blank space can be viewed as logically corresponding to the number sixteen.

From a computational point of view, this design has two drawbacks: the tile numbers start at one and the number logically associated with the blank space depends on the size of the puzzle. A better design would be to start with the blank space in the upper left-hand corner and associate the number zero with it. Now, the remaining fifteen tiles are placed as usual in ascending order from left to right and top to bottom.

This **program template** implements an interactive implementation of this modified version of the Fifteen puzzle. Running the template initializes the puzzle in its *solved* configuration with the blank space (represented by the light blue tile with number zero) in the upper left corner. You can use the arrows keys to swap tile zero with its neighboring tiles. With a fairly small number of key presses, one can scramble the board so that it is difficult to return the board to the solved configuration. Your task for this week is to implement a solver for this modified version of the Fifteen puzzle.

To begin, we suggest that you spend a few minutes examining the template in detail. The main component of the template is a `Puzzle` class that provides the functionality for our template. The contents of the board are represented as a 2D list of numbers. In particular, note that the number of the tile currently positioned in the row `row` and column `col` of the puzzle is `self._grid[row][col]`. Here, row zero is the topmost row and column zero is the leftmost column with indices increasing from left to right and top to bottom.

Modeling solutions to the Fifteen puzzle

Our goal for both the homework and mini-project will be to add several methods to the `Puzzle` class that automatically solve the puzzle after it has been scrambled. The solution will be a sequence of tile swaps between tile zero and its neighbors. At this point, we must make a critical decision: *How will we represent solutions to the puzzle?*

One possibility would be to record the path of tile zero as it moves through the puzzle during the solution process. This choice is awkward for two reasons: the position of tile zero is a pair of indices, which requires more indexing. Also, building an inconsistent path for the tile is possible (i.e; a bug could lead to a path that is not physically possible).

Instead, we have chosen to encode the path of tile zero as a string whose entries are either `"l"`, `"r"`, `"u"`, and `"d"`. These letters correspond to swaps of tile zero with its left, right, up, and down neighbors, respectively. This design choice will allow us to specify relative movements of tile zero without worrying about its position and record these movements more compactly as a string.

Moreover, illegal movements such as attempting to swap tile zero across the border of the puzzle can be handled easily, allowing for simpler debugging. To experiment with this model, use the arrow keys to move tile zero around the puzzle and then click the "Print moves" button. The corresponding movement string will be printed in the console. We recommend that you use this functionality when creating your own puzzle configurations for testing.

Mark as completed