COSC 411 Homework 1 (10 points)

The 15-puzzle is a popular sliding puzzle, and has been around for over a hundred years. In this puzzle, there is a 4x4 grid with fifteen tiles (every tile has one number from 1 to 15) in random order and one empty space. The goal is to place the numbers on tiles in order by making sliding moves that use the empty space. For example, figure 1 shows one possible initial configuration of the puzzle, and players target on obtaining the final configuration as shown in figure 2.

14	15	13	5
11	2	10	9
	7	4	6
12	8	3	1

Figure 1: Initial Configuration

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

Figure 2: Final Configuration

Your task is to implement the 15-puzzle using PyQt. Your program should begin by displaying an 4x4 grid with fifteen numbered cells (numbered 1 through 15 in random order) and an empty cell. When the player clicks on a numbered cell located on the immediate left of the empty cell, the numbered cell slides to its right, swapping the position with the empty cell. Similarly, when the player clicks on a numbered cell located on the immediate right of the empty cell, the numbered cell slides to its left. And when the player clicks on a numbered cell located right above/below the empty cell, the numbered cell slides to its lower/upper position respectively. Your program should also support players in sliding more than one numbered cells all together after a single click if those numbered cells are all on the left/right of the empty cell or all above/below the empty cell (see the example below). Clicks on the cells not on the same row/column to the empty cell should be ignored.

Solvability

For the 15-puzzle, there are 2.092279×10^{13} (=16!) possible initial configurations, but not all the configurations are solvable. For example, if players start with the initial configuration below (Figure 3), it's impossible to solve the puzzle. In fact, only 50% of the 2.092279×10^{13} configurations are solvable.

1	2	3	4
5	6	7	8
9	10	11	12
13	15	14	

Figure 3: Unsolvable Configuration

Mathematicians have summarized a simple formula to determine if a configuration is solvable as follows:

- If the grid width is odd, then every solvable state has an even number of inversions.
- If the grid width is even, then every solvable state has
    - an even number of inversions if the empty space is on an odd numbered row counting from the bottom;
    - an odd number of inversions if the empty space is on an even numbered row counting from the bottom.

Here, an inversion is when a tile precedes another tile with a smaller number on it. For example, in figure 1, number 14 is top left, then there will be 13 inversions from this tile, as numbers 1-13 come after it. On the grid of figure 1:

- the 14 gives 13 inversions
- the 15 gives 13 inversions
- the 13 gives 12 inversions
- the 5 gives 4 inversions
- the 11 gives 9 inversions
- the 2 gives 1 inversion
- the 10 gives 7 inversions
- the 9 gives 6 inversions
- the 7 gives 4 inversions
- the 4 gives 2 inversions
- the 6 gives 2 inversions
- the 12 gives 3 inversions
- the 8 gives 2 inversions

- the 3 gives 1 inversion
- the 1 gives none

So there are 79 inversions in this example. Since the grid width is 4 which is even, the empty cell is located in the second row counting from the bottom, and the number of inversions is 79 which is odd, this configuration is solvable.

Suggested Approach

Begin by populating the grid with number 1 to 15 and a 16th value, representing the empty cell, randomly located among the 16 cells. Check the solvability of the configuration first. If the configuration is not solvable, regenerate the configuration and check the solvability again. Repeat the procedure until the configuration is solvable. If the configuration is solvable, reveal the contents of the cells on the screen. When the player clicks in the window, determine if clicked in a cell and if so, which one. If the player clicks the cell who locates on the same row or column to the empty cell, sliding(s) towards the corresponding direction should happen. If the cell clicked is not on the same row nor on the same column, the clicks should be ignored. When all the cells are located on the positions as shown in the figure 2 which is the final configuration, the game is over.

EXAMPLES (NOT PRESCRIPTIONS)

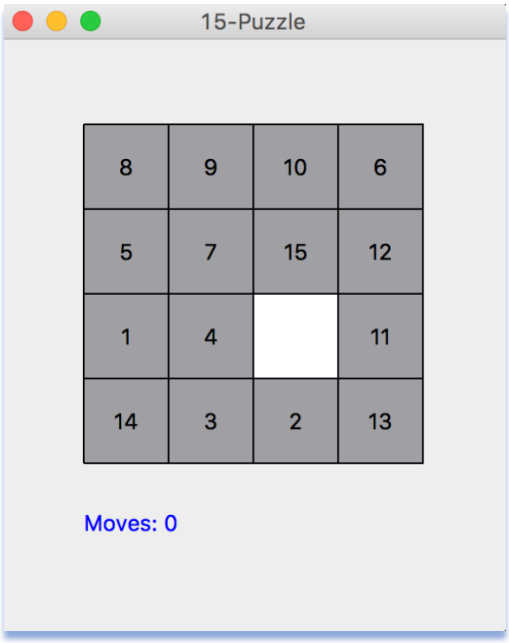

Figure 4: Program upon launch.

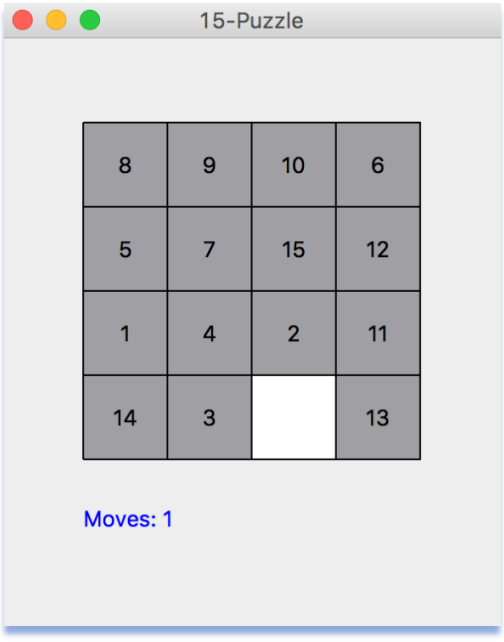

Figure 5: Program after player clicks 2.

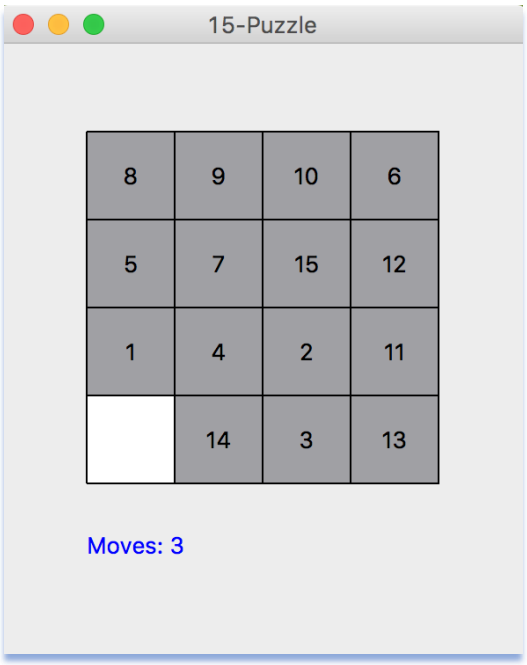

Figure 6: Program after player clicks 14. Note that the tiles 3 and 14 slide to their right positions all together after this click, and the number of moves is increased by 2.

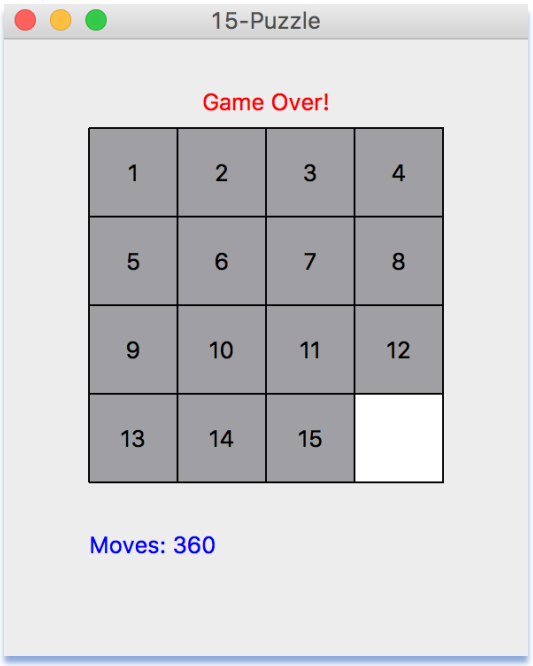

Figure 7: Program after player solved the puzzle. Note that after the game is over, clicks are ignored.

Functional Expectations and Point Values

- **2 points:** The program begins by displaying a 4x4 grid of cells that contain numbers 1 to 15 and an empty space in random order.
- **1 point:** Puzzles shown on the screen should always be solvable.
- **1 point:** The program ignores clicks outside of the grid, on the cells not on the same row or column to the empty cell, and after the game is over.
- **1 point:** When the player clicks the cell located on the immediate left/right of the empty cell, or right above/below the empty cell, the cell swaps its position with the empty cell.
- **2 points:** When the player clicks the cell located on the left of, but not adjacent to the empty cell, all the cells between the left neighbor of the empty cell and the clicked cell (inclusive) slide to their right positions by one all together, leaving the clicked position empty (see the figure 6 as an example). When the player clicks the cell located on the right of, or above, or below, but not adjacent to the empty cell, all the cells between the corresponding neighbor and the clicked cell are moved in the similar way.
- **1 point:** The program updates and displays player's total number of moves on the screen.

- **2 points:** Quality of your report, which includes

  1) description on how your program works, including a walkthrough of exactly what happens when the user clicks in your window;

  2) 10 screenshots for (a) initial configuration, (b) move one cell left, (c) move one cell right, (d) move one cell up, (e) move one cell down, (f) move >= 2 cells left, (g) move >= 2 cells right, (h) move >= 2 cells up, (i) move >= 2 cells down, and (j) final configuration;

  3) Copy and paste your Python source code;

  4) Include references to any sources you consulted;

  5) Who you discussed with or asked help from;

SUBMISSION EXPECTATIONS

`Report.pdf`: A report PDF document introduced above.

`Fifteen.py`: Your main program. Your entire Python implementation should be present within this file.

Policy

1. The students are allowed to discuss ideas with other students enrolled in the class. However, each student MUST finish the coding and report writing independently. TEAM WORK and CODE SHARE are NOT allowed.

2. If you need any help, please feel free to contact the instructor.