CSCI 141 Computational Problem Solving
Term Project: 15-Puzzle

Due Friday, 8 May 2020 at 2200

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 for this project is to implement the 15-puzzle using PyQt5. 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.

In addition, your program should also display a leaderboard that contains the top 5 results from the previous plays. After each play, if the new result beats any of the results on the leaderboard, the program should pop up a dialog window to ask for entering the player's name and update the leaderboard with the new result.

Solvability

For the 15-puzzle, there are 2.092279x10^{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 2.092279x10^{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.

You could find more detailed explanations and mathematical proofs about the solvability of the puzzle on this website.

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 she/he 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.

In order to maintain the records on the leaderboard, we provide a txt file that initially contains top 5 results with the players' names, and their total moves. Your program should show the leaderboard by retrieving the information from that provided txt file (see Figure 4). At the end of the play, if the number of moves is less than any of the results on the leaderboard, the program should ask for entering the player's name and update both the txt file and the leaderboard on the window.

Functional Expectations and Point Values

There are in total 30 points for the required part, and there are up to 2 dazzle points which are optional.

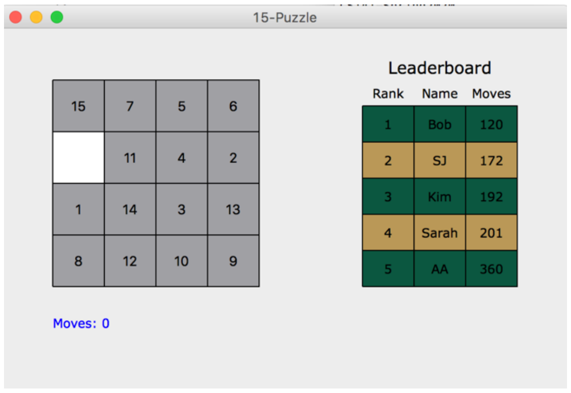
- **3 points:** The program begins by displaying a 4x4 grid of cells that contain numbers 1 to 15 and an empty space in random order.
- **3 points:** The program displays the leaderboard by retrieving the information from the provided txt file once the program is launched.
- **3 points:** Puzzles shown on the screen should always be solvable.
- **3 points:** 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.
- **3 points:** 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.
- **6 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.

- **3 points:** The program updates and displays player's total number of moves on the screen.
- **3 points:** The program asks for entering the player's name and updates the leaderboard if the new result beats any of the results on the leaderboard, and this update should be permanently saved in the provided txt file.
- **3 points:** Quality of your writeup, including references to items you consulted and a detailed explanation of exactly what happens when a user clicks in the window.
- **2 points:** Dazzle us with something else that you can do with your program that personalizes it without changing the basic functionality described above. Some examples might include adding a button to start a new game, supporting players in retracting false moves, adding sound effects for clicks, etc. Use your imagination and consult the documentation to discover how to implement your ideas.

Here are some screen captures of the reference implementation. These are examples, not prescriptions. You are free to style your program however you like. We chose to use William & Mary's official colors, specifically William & Mary Green and Spirit Gold, for the leaderboard.

EXAMPLES (NOT PRESCRIPTIONS)



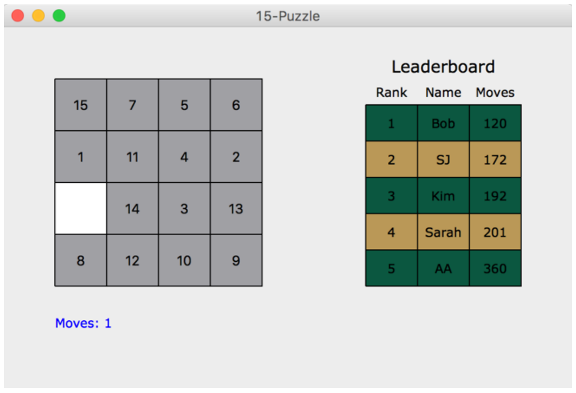
15	7	5	6
	11	4	2
1	14	3	13
8	12	10	9

Leaderboard

Rank	Name	Moves
1	Bob	120
2	SJ	172
3	Kim	192
4	Sarah	201
5	AA	360

Moves: 0

Figure 4: Program upon launch.



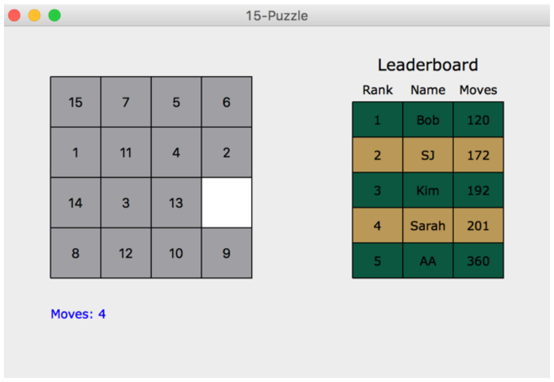
15	7	5	6
1	11	4	2
	14	3	13
8	12	10	9

Leaderboard

Rank	Name	Moves
1	Bob	120
2	SJ	172
3	Kim	192
4	Sarah	201
5	AA	360

Moves: 1

Figure 5: Program after player clicks in space 2, 0.


15-Puzzle

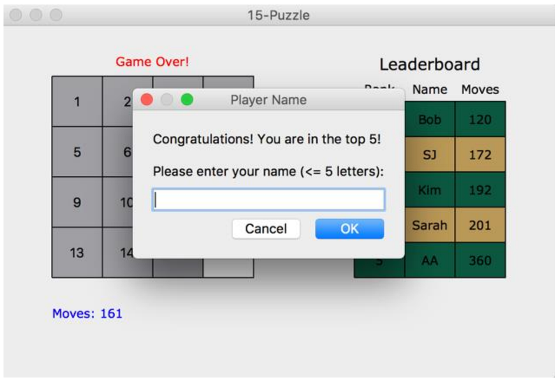
15	7	5	6
1	11	4	2
14	3	13	
8	12	10	9

Moves: 4

Leaderboard

Rank	Name	Moves
1	Bob	120
2	SJ	172
3	Kim	192
4	Sarah	201
5	AA	360

Figure 6: Program after player clicks in space 2, 3. Note that the tiles 14, 3 and 13 slide to their left positions all together after this single click, and the number of moves is increased by 3.


15-Puzzle

Game Over!

1 2
5 6
9 10
13 14

Moves: 161

Leaderboard

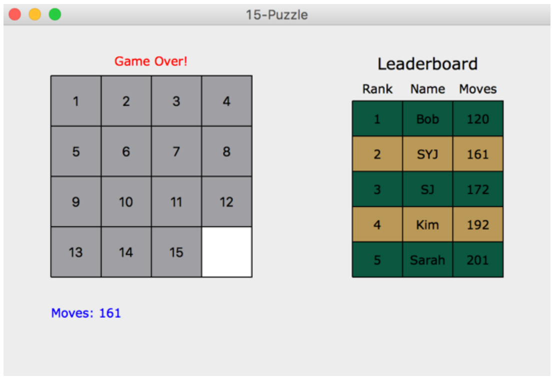
Rank	Name	Moves
	Bob	120
	SJ	172
	Kim	192
	Sarah	201
	AA	360

Player Name

Congratulations! You are in the top 5!

Please enter your name (<= 5 letters):

Figure 7: Program after player solved the puzzle. Since the total number of moves is 161, which beats some of the results on the leaderboard, another window pops up asking for entering the player's name.


15-Puzzle

Game Over!

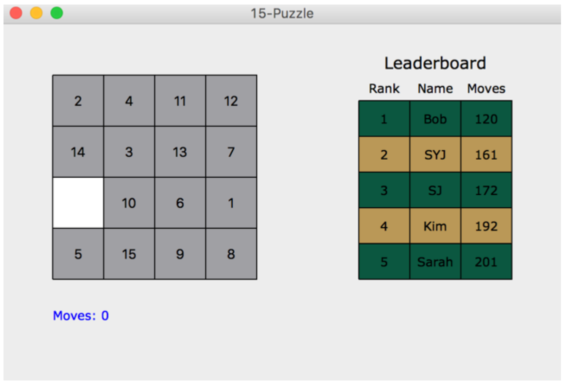
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

Moves: 161

Leaderboard

Rank	Name	Moves
1	Bob	120
2	SYJ	161
3	SJ	172
4	Kim	192
5	Sarah	201

Figure 8: Program after player entered her name SYJ on the pop-up window and hit the OK button. Note that the ranking on the leaderboard has been updated. After the game is over, clicks are ignored.


15-Puzzle

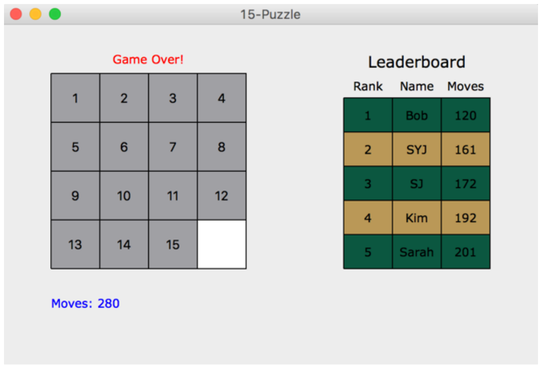
2	4	11	12
14	3	13	7
	10	6	1
5	15	9	8

Moves: 0

Leaderboard

Rank	Name	Moves
1	Bob	120
2	SYJ	161
3	SJ	172
4	Kim	192
5	Sarah	201

Figure 9: Program upon relaunch following the last play. Note that the program is able to retrieve the same leaderboard maintained in the file 'leaderboard.txt'.


15-Puzzle

Game Over!

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

Moves: 280

Leaderboard

Rank	Name	Moves
1	Bob	120
2	SYJ	161
3	SJ	172
4	Kim	192
5	Sarah	201

Figure 10: Program after player solved the puzzle. Since the total number of moves is 280, which does not beat any result on the leaderboard, no window pops up in this case.

SUBMISSION EXPECTATIONS

Files that do not conform to these expectations will not receive credit. This includes case and spelling.

`93xxxxxxxx.pdf`: A document that outlines how your program works, including a walkthrough of exactly what happens when the user clicks in your window. Your writeup must also explain what you implemented for the project's dazzle point and why it is worth 2 points. Also include references to any sources you consulted, regardless of format. Use your W&M ID number for the name of the file.

`Fifteen.py`: Your main program. We will launch this program and expect a window to appear with a grid in which we can click to play the game. A skeleton file is not provided, but you can use files from previous projects and lectures to build from. You may need to import additional modules. **If you encounter trouble importing a particular module, post the problem on Piazza, and we will direct you to the correct import line.**

`Dazzle.zip`: A zip file containing any other files your program needs in support of your dazzle points. This file is not required for the specified behavior; you may need it depending on your dazzle approach. We will place the contents of this zip archive in the same folder as your Fifteen.py. Your program should assume that any additional files are in its same directory.