OVERVIEW:

This program can create an interactive sliding puzzle game where users can choose any dimensions up to 10x10, minimum dimension is 3x3. The board has an empty space where an adjacent tile can be slid to. Users can choose dimension and the 4 buttons of controlling the sliding direction by themselves. And invalid input will be asked to be retyped. The objective of the game is to rearrange the tiles into a sequential order by their numbers (left to right, top to bottom) by repeatedly making sliding moves (left, right, up or down).

When the user enters the direction of movement, the program will remind the user which direction is valid. Invalid input will be reminded to retype. When the user successfully solves the puzzle, the program will congratulate user and tell user how many times he had moved (Invalid input is not included). After that, the program will ask user whether they want to continue playing or not. If user input “n”, it will regenerate a new puzzle. If user input “q”, it will be closed. If user input something else, it will prompt user only can input “n” or “q”, and let user retype.

DATA MODEL:

The dimension: n

It is an integer between 3 and 10(including 3 and 10). Type of it is integer.

Control move: l, r, u, d

Each of them is a letter. And they must be different letters. Types of them are all string.

List: li, list1, list2

Types of them are all list.

Continue or exit: n, q

Types of them are all string.

PROGRAM STRUCTURE:

First, give user a brief introduction of this game. Just print the introduction.

Second, prompt user to input the dimension which he wants. Use the WHILE and TRY to break out of the loop until the user enters a reasonable value.

Third, let the user decide which four letters to control move. Use the WHILE and TRY to break out of the loop until the user enters reasonable values.

Fourth, generate a randomized and solvable puzzle for the user to solve.

Define a function that gets a randomized and solvable puzzle:

get\_valid\_list(li: list)

Define a function that print the list in the screen with each row of n: display\_list(li: list)

Define a function that update the list when user input direction:

update\_list(li: list, m: str)

Define a function that prompt the player the valid sliding direction and enter it:

input\_move(li: list)

Define a function that integrate other functions to start the game:

start\_game()

Fifth, when the user has solved the puzzle, congratulate him and ask whether he wants to continue playing or not. Use WHILE loop to decide whether to continue playing or not according to the input.

PROCESSING LOGIC (SPECIFIC):

Main processing logic:

Create an ordered list of numbers with n\*\*2 elements based on the size entered by the user. At the same time, create another list for final verification. Let user input 4 letters to control the movement: left, right, up and down. Regard the 0 in the list as the blank space in the game. The movement of user input is the opposite of the movement of position 0, for example, when user input move left, the 0 will move right. Define get\_valid\_list(li: list) function, let 0 randomly move left, right, up or down 1000 times, scrambling the list in this way ensures that the puzzle is randomized and solvable. Then, define display\_list(li: list) function to print the list as game’s form in the screen. After that, define input\_move(li: list) function to prompt the user which moves are valid and let user input a valid move. Then, define start\_game() function integrate the previous functions together and count the effective movement of the user input, when user succeed, tell user the number of moves. Last, use WHILE loop and start\_game() function to ask whether user want to continue game or not.

Initial Puzzle:

After user input dimension n and control buttons, I use list1 = [i for i in range(n \*\* 2)] to get initial list and use get\_valid\_list(li:list) function to create a randomized and solvable puzzple. Then, use display\_list(li: list) function to print the list as game’s form in the screen. Last, use input\_move(li: list) function to prompt the user which moves are valid and let user input a valid move.

FUNCTIONAL SPEC:

get\_valid\_list(li: list)

The parameter li is a list. The return is a list. This function scrambles the list by randomly moving the position of 0 left, right, up or down 1000 times. This ensures that the scrambled list is solvable. When 0 is in the upper left corner, only can move left or up at random. When 0 is in the upper right corner, only can move right or up at random. When 0 is in the bottom left corner, only can move right or up at random. When 0 is in the bottom right corner, only can move right or down at random. When 0 is in the upper except corner, only can move left, right or up at random. When 0 is in the bottom except corner, only can move left, right or down at random. When 0 is in the left-most except corner, only can move left, up or down at random. When 0 is in the right-most except corner, only can move right, up or down at random. When 0 is not in the outermost shell, can move left, right, up or down at random.

display\_list(li: list)

The parameter li is a list. It has no return, and it is used to print the list as game’s form in the screen. Use FOR loop to print n elements per line, and print spaces when the element is 0.

update\_list(li: list, m: str)

The parameter li is a list, the parameter m is a string. This function moves 0 in the list by the value of m. Because the m is the opposite of the movement of 0's index. So, when m is l, 0 move right. When m is r, 0 move left. When m is u, 0 move down. When m is d, 0 move up. The function returns the list after moving.

input\_move(li: list)

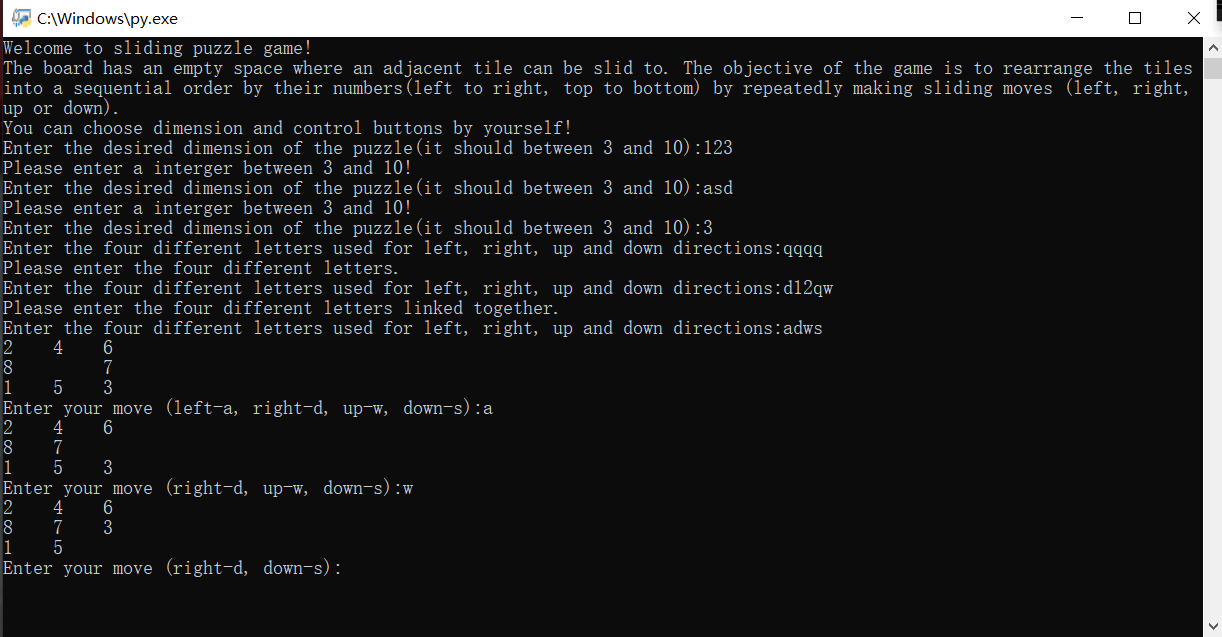
The parameter li is a list. The return is a string which user input. This function is to prompt the user what sliding direction is valid and let user input a string which representing the move direction. If user input an invalid string, the function will let user retype until it is valid. The return string is which the user input.

start\_game()

This function has no parameters and returns. It initializes the game and integrate other functions to start the game. Also, it counts the number of moves the user has made and tells the user when they have succeeded.

SAMPLE OUTPUT:

1



2



3

