Design Documentation

120090272 Feng Chuqiao SDS Assignment 1--- Sliding Puzzle

1. Design

a. Overview

The program is designed to output a sliding puzzle game for users. The users will be allowed to input 4 letters they choose, and the dimension of the puzzle in game which could be any integer number between 3 and 10, then the program will generate a random but solvable matrix with a blank. The users are expected to move the blank by using their inputed letters, and finally restore the matrix to its original order, which is from 1 to the largest number and a blank at last. When the users win the game, the program will output the total steps and then they could choose to quit the game or play the game again.

b. Data Model

i. List:

(a)**Current_matrix**: Use a list to store the current puzzle, and the order of the numbers in the list represent the position of the puzzle.

```
current_matrix = []
current_matrix = list(range(size ** 2)) # the initial order
```

(b)**User_op_letter**: Put the user input letter in a list in order to control the move.

```
user_op_letter = []
for i in range(4):
    name_to_number[user_op_letter[i]] = i
```

(c)**Move**: Store the current possible direction be numbers based on the current position of the blank puzzle. Use number to represent the direction of move.

```
move = []
if g_line != size-1: move.append(0)
if g_line != 0: move.append(1)
if g_row != size-1: move.append(2)
if g_row != 0: move.append(3)
```

(d)**Standard_name**: Store all the possible direction of movement.

```
standard_name = ['left', 'right', 'up', 'down']
```

ii. Dictionary:

(a)**name_to_number**: Use dictionary to connect the keywords to the number in the move list. '0' is left, '1' is right, '2' is up, '3' is down.

```
name_to_number = dict.fromkeys(user_op_letter)
```

iii. String:

(a)**op_letter**: In case that the user may input the letter in capitals, so the input command is designed to make the letter at a lower case to run the program right.

```
op_letter = input('Enter 4 letters for left, right, up, and down directions:
').lower()
```

iv. Variable:

(a)**g_row, g_line**: Use number '0' to represent the blank puzzle' s location, index the '0' in current_matrix list, and use 'divmod' function and integer 'g_row, g_line' to show the 'row * line' in the 'n * n' matrix.

```
g_row, g_line = 0, 0
s = current_matrix.index(0)
g_row, g_line = divmod(s, size)
```

(b)**op_number**: For link of number op_number as key in the dictionary name_to_number, to move the blank puzzle, if op_number = 0, 1, 2, 3, the blank will move left, right, up and down.

```
a = input()
if a in name_to_number and name_to_number[a] in move:
    op_number = name_to_number[a]
    move_blank(op_number)
```

(c)**command**: Use as the variable of function move_blank in the process of generating the matrix. Put the originally ordered number list, shuffle the list by moving the blank randomly. The command variable is randomly chose to move blank.

```
command = random.choice(move)
move_blank(command)
```

(d)**<size**: Size means the size of the matrix. If size = n, then the matrix is in 'n' dimension. Also, the program will check whether the dimension is in 3 to 10. If n out of the range of 3 and 10, the user will be warned and expected to input the size of matrix again. And if they input q, they could quit the game.

```
n = input("\nEnter a number n in range 3 to 10 to choose the dimenssion, enter
'q' to quit game: ").lower()
if n == 'q':
    print('quit')
    break
if n.isdigit() == False or int(n) < 3 or int(n) > 10:
    print('The input is invalid, please input a number in 3 to 10! ')
    continue
    size = int(n)
```

(e)**cnt_move**: The variable is for counting the time of move by the user and finally tell the user at the end of the game.

```
cnt_move = 0
if a in name_to_number and name_to_number[a] in move:
    cnt_move += 1
```

c. Program Structure.

1. Define the way to control the blank' s move.

i. Create an operate number list for judging what movement to take

First, define a function to confirm the possible direction of move, use number to represent the move, each step, append the possible direction into the 'move' list.

ii. For user to input his letter as controller

Then design a function for users to put in the letters they choose to relate the direction. And the program allow the users' wrong input. If the user input non-letter, or they input 5 or 3 letters, also if they input the same letter, the program will warn the users and ask them to input the 4 letters again.

iii.Use number to operate the movement of blank

'0' represent moving left, '1' represent moving right, '2' represent moving up, '3' represent moving down.

2. Generate the original matrix

First, generate the a correct order of number list and make them to be in the list 'current_matrix'. Then use the well defined moving method to move the blank randomly for a random times. The times of moving depends on the size and dimension of the matrix, So that it will not cost much time, and it will be enough in a higher dimension case. Then print out the current matrix line by line, and the number '0' will be printed as a blank.

3. The main body of playing this game.

i. Ask users to choose the dimension they want to play.

Ask the user to choose the dimension of game, and check whether the input is right. The input should be number and in the range of 3 and 10, and if the input is invalid, the users are expected to input the number again.

```
n = input("\nEnter a number n in range 3 to 10 to choose the dimenssion, enter
'q' to quit game: ").lower()
if n == 'q':
    print('quit')
    break
if n.isdigit() == False or int(n) < 3 or int(n) > 10:
    print('the input is invalid, please input a number in 3 to 10! ')
    continue
```

ii. To change the input letter into operation number.

Use dictionary to store the number of direction and letter to operate.

```
name_to_number = dict.fromkeys(user_op_letter)
for i in range(4):
    name_to_number[user_op_letter[i]] = i
```

iii. To inform the user what to do at each step.

Notice that the puzzle can only be certain moves in different position, different position has different way of moving. This step is designed for users to enter certain moves at certain places.

iv. Allow the users to play the game.

Then this part is for users to play and count how many moves they generated.

```
if a in name_to_number and name_to_number[a] in move:
    cnt_move += 1
    op_number = name_to_number[a]
    move_blank(op_number)
```

v. To check whether the user win or not .

This step is to check whether the user win the game or not. All the number should be at their positions, and the blank number should be at the last of the matrix.

vi. Ask the users whether to quit the game or start a new one.

```
opt = input('Enter 'q' to quit or enter any other letter to start a new game :
')
if opt == 'q':
    flg = 0
```

```
if flg == 0:
    print("quit")
    break
```

d. Processing Logic (the details are in process structure)

i. The main processing logic.

1. Define the way of moving how use letter to put in list and use dictionary to link the operate number and user-input operate letters.
2. Generate the original matrix and inform the users how to play the game, then the users are expected to play the game.
3. Then check whether the game is win, if the game win, tell the users their number of moves, and ask the users to quit the game or restart a new one.

ii. The technique developed to generate the randomized puzzle.

Define the moving first. Generate an already ordered matrix, locate the blank puzzle, then move the blank randomly at several and enough times. Then output the shuffled matrix.

2. Function Specification

1.**The function to create an operate number list - 'move' for judging what move to take**

The details are as below, clear the move list at the beginning of each step and return the possible move list at the end of each step. The variables used here are the size of matrix, g_row, g_line in global.

```
def operate_move_puzzle():
    global size, g_row, g_line, move
    move.clear()
    if g_line != size-1: # when the blank is not at the most left
        move.append(0)  # means the blank 'can' move left
    if g_line != 0: # when the blank is not at the most right
        move.append(1)  # means the blank 'can' move right
    if g_row != size-1: # when the blank is not at the top
        move.append(2)  # means the blank 'can' move up
    if g_row != 0: # when the blank is not at the most bottom
        move.append(3)  # means the blank 'can' move down
    return move
```

2.The function for user to input his letter as controller

The function below uses variables of global size, g_row, g_line, also the list of move and current_matrix, it will allow the users to input several letters as directions' controllers. Finally it return the list of user_op_letter.

```
def input_directions():
    global size, g_row, g_line, move, current_matrix
    user_op_letter.clear()
    while True:
        op_letter = input('Enter 4 letters for left, right, up, and down
directions: ').lower()
        for item in op_letter: # make sure the input is alpha and not same
alpha.
            if item.isalpha() == True and item not in user_op_letter:
                user_op_letter.append(item)
                continue
            if len(user_op_letter) == 4: # make sure the number of letters is 4.
                break
            if len(user_op_letter) != 4:
                print('The input is invalid, please try it again! ')
                continue
            return user_op_letter
```

3.The function use number to operate the movement of blank

The function use number (op_number) to operate the movement of blank by swapping the position of numbers in a list of current matrix, and the details are as below. In this function we use global g_row, g_line, and list of current matrix. And the function will return the moved matrix each step.

```
def move_blank(op_number):
    global size, g_row, g_line, move, current_matrix
    a = g_row * size + g_line # locate the current blank' s poition
    if op_number == 0: b = g_row * size + g_line + 1 #left
    if op_number == 1: b = g_row * size + g_line - 1 #right
    if op_number == 2: b = (g_row + 1) * size + g_line #up
    if op_number == 3: b = (g_row - 1) * size + g_line #down
    current_matrix[a], current_matrix[b] = current_matrix[b], current_matrix[a]
    # swap the position of two number
    return current_matrix # return the changed matrix
```

4.The function to generate the original matrix

The function is to generate the original matrix which is not in order for users to play. This function uses the variables of size, g_row, g_line and the list of move and current_matrix. The function will return the print of original matrix.

```
def matrix_generate():
    global size, g_row, g_line, move, current_matrix
    current_matrix = list(range(size ** 2))
    # Create an list to put in number of matrix in order
    for m in range(random.randint(3*(size**2), 6*(size**2))): # The times of
move
        s = current_matrix.index(0)
        g_row, g_line = divmod(s, size)
        operate_move_puzzle()
        command = random.choice(move)  # Randomly move the matrix
        move_blank(command)
        zero = current_matrix.index(0)
        g_row, g_line = divmod(zero, size)  # Locate the current '0'
    for i in range(size):
        for j in range(size):
            if i == g_row and j == g_line:
                print("   ", end=" ")  # Print blank at 0
             else:
                print("%3d" % current_matrix[i * size + j], end=" ")
        print('\n') # Print out the matrix
```

5. The function to print the step information

The function is to print the step information. Here it uses the variables of g_row, g_line and the list of move, current_matrix, user_op_letter. And finally the function will return and print the step information at end of each step to inform the user how to play the game.

```
def step_information():
    global size, g_row, g_line, move, current_matrix, user_op_letter
    print('enter your move:', end='')
    for i in move:  # Different position has different way of moving
        print(' [{}]-{} '.format(user_op_letter[i], standard_name[i]), end='')
    print(": ", end="")
    return
```

6. The function to check whether the game win

The function is to check whether the user win the game or not. It uses the global list current_matrix and global variable size. Finally it will return true if the game win.

```
def win():
    global size
    for i in range(size**2 - 1):
        g_row, g_line = divmod(i, size)
        if current_matrix[g_row * size + g_line] != i + 1:
            return False
    return True # Because here I use flag and True or False to stop or continue
the game.
```

3. Sample Output

```
Microsoft Windows [版本 10.0.18363.1441]
(c) 2019 Microsoft Corporation。保留所有权利。

C:\Users\pc\Desktop\CSC1002>C:/Users/pc/AppData/Local/Programs/Python/Python38/python.exe "c:/Users/pc/Desktop/CSC1002/# Sliding_Puzzle_202
1.py"
Welcome!

This is 120090272's sliding puzzle game! First you can enter 4 letters to denote move 'left', 'right', 'up' and 'down'.
(Such as 'j', 'k', 'r', 'f' for 'left', 'right', 'up' and 'down')
Then choose a dimension n between 3 and 10 to start your game (in an n * n matrix) to play.
Game 'win' when you move the puzzles into its correct orders.

Enter 4 letters for left, right, up, and down directions: adws
Now you can play the game:

Enter a number n in range 3 to 10 to choose the dimenssion, enter 'q' to quit game: 3
  4   3   2

  6   8   1

  7       5

enter your move: [a]-left  [d]-right  [s]-down :
```

```
enter your move: [a]-left  [w]-up : a
  3       2

  4   6   1

  7   8   5

enter your move: [a]-left  [d]-right  [w]-up : a
  3   2

  4   6   1

  7   8   5

enter your move: [d]-right  [w]-up : w
  3   2   1

  4   6

  7   8   5

enter your move: [d]-right  [w]-up  [s]-down : d
  3   2   1

  4       6

  7   8   5

enter your move: [a]-left  [d]-right  [w]-up  [s]-down : d
  3   2   1

      4   6

  7   8   5

enter your move: [a]-left  [w]-up  [s]-down :
```

```
enter your move: [a]-left  [d]-right  [s]-down : a
  1   2   3

  4   5   6

  7   8


Congratulations! You win in 50 moves !

Enter [q] to quit or enter any other letter to start a new game : q
quit
Thank you for playing this game!
```

```
enter your move: [a]-left  [d]-right  [s]-down : a
  1   2   3

  4   5   6

  7   8


Congratulations! You win in 29 moves !

Enter [q] to quit or enter any other letter to start a new game : n

Enter a number n in range 3 to 10 to choose the dimenssion, enter 'q' to quit game: 4
  1   4   2   6

  3   7      11

  8  12   5  10

 13   9  14  15

enter your move: [a]-left  [d]-right  [w]-up  [s]-down :
```