Design Doc

Overview

This program allows users to enter an integer for dimension, and the system automatically create a (dimension*dimension) square. It contains number :1~dimension**2-1 and an empty string. The square will be randomly rearranged but it is solvable (which means via moving the number, it can finally be ordered from up to down, left to right).

The users will also input 4 different characters to represent moving left, right, up, and down. Each time by input in the keyboard, the number adjacent to the blank would move towards the input direction. When the users finally arrange the puzzle orderly, they can choose to start again or quit. Once they choose start again, a new random puzzle will be given.

Data Model

The core object in the program is g_list1. It is a list. It contains dimension**2 elements. One of them is an empty string and the others are different numbers from 1 ~ dimension**2 – 1. In the later operations, g_list1 was turned into a random list. It helps to generate sliding puzzle. In the whole process, it changes as the program goes.

Another essential object is g_list2. It is a list. It contains dimension**2 elements. One of them is an empty string and the others are different numbers from 1 ~ dimension**2 – 1. Different from g_list1, it is well-ordered all the time. It is used to check whether g_list1 was turned into the ordered list. If so, the game is successful.

There is an object “s” in the program. It is a string. It is used to tell the position of the space. s is also used to generate a statement of an input() function. Via the use of s, and “+” marks, we can print the moves which the space can go accordingly.

Program Structure

Basically, the program can be divided into four components.

The first component: generate the randomized, solvable sliding puzzle.

To assure its solvable character, the puzzle must be made from its well-ordered form. By moving its adjacent numbers randomly, as long as there are enough times, we can obtain a highly random list. By print the list with a certain method, we can obtain the required table.

The function used: (with corresponding to the conclusion above)

generate_dimensional_list(): this function is used to obtain g_list1 and g_list2, both of them are well-ordered at this time.

up_move(): this function is used to make the space move upwards one unit.

down_move(): this function is used to make the space move downwards one unit.

right_move(): this function is used to make the space move rightwards one unit.

left_move(): this function is used to make the space move leftwards one unit.

generate_a_random_list(): this function will utilize the up_move(), left_move(), right_move(), and down_move() function to generate a randomized but solvable list.

generate_a_puzzle(): this function is used to print the puzzle in a required form.

The second component: interact with the user. (playing the game)

To play the game, we must know where the blank is. If it is at the edge of the puzzle, there exists some moves that cannot go. For instance, if the blank is in the left-up corner, then the user cannot input to move the number right or down (numbers go oppositely to the blank). And the input statement will adjust accordingly. After the input is adjusted, the program must to be set to be able to go to the instructed position. All settings finished, the player can go infinitely moves until the puzzle (g_list1) was arranged orderly.

<u>The function used: (with corresponding to the conclusion above)</u>

input_the_statement(): according to the position of the blank, it will generate different strings as input statement.

input_the_choice(): the function is used to prompt the statements generated from the front function. It will let the user know which movement can they conduct, and require the player to input the right letter which represents the position of move.

one_move(): according to the player's input, the function will enforce the numbers move to the determined direction.

play(): this is the function to repeat the movement until the puzzle (g_list1) is arranged orderly. It can also record the number of moves

The third component: ask the player to input the parameters

In this part, the players are required to enter values for dimension and four letters four left\right\up\down moves. All the inputs should be robust enough. If inputting wrongly, the program will automatically ask the player to input over and over again, until all the inputs meet the requirement.

<u>The function used: (only one)</u>

obtain_the_dimension(): this function is used to obtain the value for dimension. It is designed to prevent the crash caused by invalid input. The function will return the value of dimension

obtain_the_keys():the function is to obtain the keys of moves for left\right\up\down. It is designed to prevent the crash caused by invalid input. The function will return the values of keys

The fourth component: the conduct of the functions above

This part has no defined functions.

This part has two objects. The first is to implement the functions above. Besides, it uses "while True…if…break" structure to control the terminal of the program. Before the player successfully rearrange the puzzle, the loop will not break. After that, the player is required to choose whether continue or leave.

Processing Logic (Specific)

First of all, the program uses list g_list1, which is invisible to the user, to link the first two parts. By using function generate_a_puzzle(), the puzzle will be printed on the screen. Seeing the puzzle, the user can determine which move should go, in order to make the puzzle (g_list1) in order (==g_list2). g_list1 plays an essential role. It is the "real puzzle" and all the so-called "moves" are from exchanging the position of elements in g_list1.

To check whether g_list1 is ordered, which means, the puzzle is solved, the list g_list2 plays a crucial role. It is unchangeable since its creation. It is a ordered list, from left to right, from 1 to dimension**2-1. It passes the whole program.

To obtain the input statement of move, the use of s-string can enormously simplify the program. s-string is initially a string with no direction instructions. By the sign "+", it represents the moves it can conduct. For instance, if the blank is **not** on the left edge of the puzzle, then the there exists a number in its left to go right. Therefore, the statement "right-{}" .format() is add into the string.

The whole program is united by the last component, where each of the three components conduct in this part. First, it conducts the input instructions. Then it creates the randomized and solvable puzzle. Then it interacts with the player. And last, asking the player whether to continue or not. This component makes the process smooth and organized.

To generate the initial, randomized puzzle:

Firstly, we create a list (g_list1) to contain the elements which will appear in the puzzle. We have to make sure g_list1 has elements of numbers from 1~dimension**2 – 1 and an empty string. By using generate_dimensional_list() can obtain this object. This will make g_list1 a well-ordered list

Secondly, in order to make it solvable, we have to design the way it is randomized. We design four move functions to ensure it can move back to the well-ordered list. This step is to set up the randomizing method.

Thirdly, by import random numbers from 1 to 4, each represents a moving direction, we can make the blank space move. We can repeat this operation sufficiently large times, to get a highly upset list. In the program, I set step =10000, which means the operation repeats 10000 times.

Functional specifications

generate_dimensional_list(p_dimension): p_dimension is the parameter for the dimension of the puzzle. The form of puzzle is just the square of dimension**2. This function will create two global variables, which are g_list1 and g_list2. Both of them are well organized by this function.

up_move(p_dimension): p_dimension is the parameter for the dimension of the puzzle. Firstly, it locates where the empty string is. Then judge whether it can move upwards, by calculating the index of the empty string compared with p_dimension. If p_dimension > the index, it cannot move upwards. In this situation, it will print ('you cannot go down) ('go down' because the movement of numbers is opposite to the movement of the blank space). Otherwise, it will switch the position of g_list1[index] with g_list1[index-p_dimension]

right_move(p_dimension): p_dimension is the parameter for the dimension of the puzzle. Firstly, it locates where the empty string is. Then judge whether it can move rightwards, by calculating the index of the empty string compared with p_dimension. If (the index+1)%p_dimension ==0, it cannot move rightwards. In this situation, it will print ('you cannot go to the left') ('go to the left' because the movement of numbers is opposite to the movement of the blank space). Otherwise, it will switch the position of g_list1[index] with g_list1[index+1]

left_move(p_dimension): p_dimension is the parameter for the dimension of the puzzle. Firstly, it locates where the empty string is. Then judge whether it can move leftwards, by calculating the index of the empty string compared with p_dimension. If the index%p_dimension ==0, it cannot move leftwards. In this situation, it will print ('you cannot go to the right') ('go to the right' because the movement of numbers is opposite to the movement of the blank space). Otherwise, it will switch the position of g_list1[index] with g_list1[index-1]

down_move(p_dimension): p_dimension is the parameter for the dimension of the puzzle. Firstly, it locates where the empty string is. Then judge whether it can move downwards, by calculating the index of the empty string compared with p_dimension. If p_dimension **2-p_dimension<= the index, it cannot move downwards. In this situation, it will print ('you cannot go up) ('go up' because the movement of numbers is opposite to the movement of the blank space). Otherwise, it will switch the position of g_list1[index] with g_list1[index-p_dimension]

generate_a_random_list(p_dimension): p_dimension is the parameter for the dimension of the puzzle. This function imports random. it will randomly and repeated move the blank space. It has a local variable 'step' to represent the repeat times. It uses a for-loop. Finally, it will return the randomized g_list1.

generate_a_puzzle(p_dimension): p_dimension is the parameter for the dimension of the puzzle. The function will go through all the values in g_list1 and print them one by one. for the ith element, i is in range(1,len(g_list1)+1). If i% p_dimension != 0, it will print(g_list[i-1],end = '\t'); else, it will just print(g_list[i-1])

input_statement(p_left,p_right,p_up,p_down,p_dimension): p_dimension is the parameter for the dimension of the puzzle. p_left\right\up\down all represent the directions the number will move respectively. In this function, it will check where your blank space is, and will generate a statement for the following input function. This function will return the s-string, which represents where the blank space is.

Input_the_choice((p_left,p_right,p_up,p_down,p_dimension) : p_dimension is the parameter for the dimension of the puzzle. p_left\right\up\down all represent the directions the number will move respectively. This function will let the user to input his\her moving choice. It will provide the moves which the user can choose. The user must input the correct letters. It uses while True loop. It will break when the user inputs the right value. Finally, it will return the user's choice

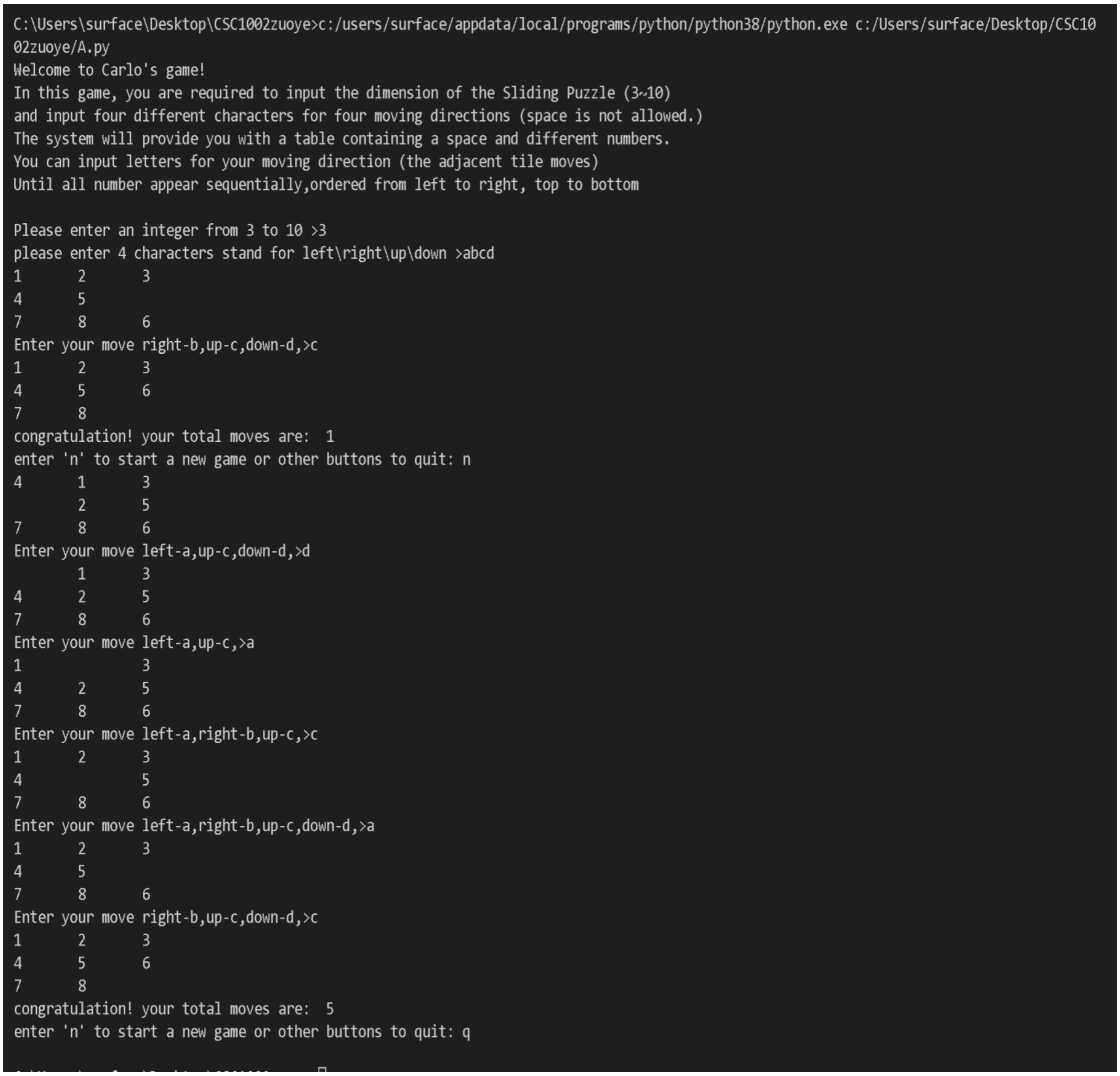
one_move(p_left,p_right,p_up,p_down,p_dimension) : p_dimension is the parameter for the dimension of the puzzle. p_left\right\up\down all represent the directions the number will move respectively. This function is used to move the blank space. It will follow the input of what the user enters. The inputs is generated by the function above.

play(p_left,p_right,p_up,down,p_dimension): p_dimension is the parameter for the dimension of the puzzle. p_left\right\up\down all represent the directions the number will move respectively. This function is used to repeat the move. It uses while True loop. Only when the g_list1 is well organized can the loop break. On top of that, it will automatically records how many times you have tried in the game. When you succeed in the puzzle, it will print("congratulations!", "your total moves are", the moves).

Obtain_the_dimension(): No parameters. It is used to generate the parameter. The user must input a integer from 3 to 10, 10 included, for dimension. If they input a string, a negative number, or a float number, will be invalidate and required to input again, until they are right.

Obtain_the_keys(): the user is required to input four distinguishable characters to represent their move to left, right, up, and down. If they input any same numbers, it will print 'you cannot input the same numbers', and ask the user to input again, until it meets the requirement.

Sample Output (for simplicity, setting step =10)



```
C:\Users\surface\Desktop\CSC1002zuoye>c:/users/surface/appdata/local/programs/python/python38/python.exe c:/Users/surface/Desktop/CSC10
02zuoye/A.py
Welcome to Carlo's game!
In this game, you are required to input the dimension of the Sliding Puzzle (3~10)
and input four different characters for four moving directions (space is not allowed.)
The system will provide you with a table containing a space and different numbers.
You can input letters for your moving direction (the adjacent tile moves)
Until all number appear sequentially,ordered from left to right, top to bottom

Please enter an integer from 3 to 10 >3
please enter 4 characters stand for left\right\up\down >abcd
1       2       3
4       5
7       8       6
Enter your move right-b,up-c,down-d,>c
1       2       3
4       5       6
7       8
congratulation! your total moves are:  1
enter 'n' to start a new game or other buttons to quit: n
4       1       3
        2       5
7       8       6
Enter your move left-a,up-c,down-d,>d
        1       3
4       2       5
7       8       6
Enter your move left-a,up-c,>a
1               3
4       2       5
7       8       6
Enter your move left-a,right-b,up-c,>c
1       2       3
4               5
7       8       6
Enter your move left-a,right-b,up-c,down-d,>a
1       2       3
4       5
7       8       6
Enter your move right-b,up-c,down-d,>c
1       2       3
4       5       6
7       8
congratulation! your total moves are:  5
enter 'n' to start a new game or other buttons to quit: q
```

In the program, the step = 20000, which is more randomized. This is just easy to succeed!