# Design

Design Documentation of Assignment1

* 1. Description of my thoughts and overall approach

My whole blueprint is to separate the program into several functions. These functions are used to (1) generate each type of the puzzle (2) ask the user to input orders when the white space is in different places (3) teach the computer how to move while the user input different instructions and display the updated puzzle (4) display the rules, choice, total steps and congratulations to the user.

I met problems in generating numbers and making my program more efficient and robust. Originally, my plan was to use random.randint to generate each number. However, after I finished the whole program, I played it for several times and found nearly every time I couldn’t solve it. Each time I would end at the place where there were two numbers in the opposite places and the others were in the right order. Soon I realized not every puzzle is solvable if the numbers are generated randomly. Then I searched some information online and used several rules to make sure every puzzle is solvable. For balancing readability and efficiency, I also made many efforts. At first, I wrote nearly 200 lines to simulate each situation. I wrote sentences to tell the user which directions can be chosen and told the computer how to change numbers one by one. And then I tried to pack the sentences of “how to move” into a function and run this function each time the user input an instruction. In making my program more robust, I used try-except and if-else to fix this task. I think the most difficult part is to think what wrong inputs the user may give. And as there are so many wrong inputs, I used a lot of sentences to correct them, thus the program will look a little bit messy.

* 1. Different python objects in my program
     1. list

In 8 number-puzzle, I put 8 numbers ranging from 1 to 8 and a white space in the list and use shuffle to randomly change their sequences. In 15-number puzzle, I put 15 numbers ranging from 1 to 15 and a white space into the list and also use shuffle to change their orders. If the puzzle using numbers from this list is solvable, then the original puzzle will be this list (display 3 numbers per line). When the user input the instruction, the number in list should be moved in some particular directions.

* + 1. list2

I withdraw the white space in the former list and put the rest numbers in list2. List2 is used to check whether this puzzle is solvable or not. If it is solvable, then the output will be “list”, if not, the computer will produce a new list and check the new list2 until the puzzle is solvable. Another use of list2 is to use sort method to align the numbers in order. Then use the new list2 to check whether the final list is the same as list2. If yes, then the user has solved this puzzle.

* + 1. l,r,u,d

The user will choose any characters to represent left, right, up and down. These letters will be stored in l,r,u,d respectively.

* + 1. Game

Ask the user to assign value to the object: game. If game equals to 1, then the computer will display the 8-puzzle. If game equals to 2, then it will display the 15-puzzle. If game equals to q, then it will end the game.

* + 1. Step

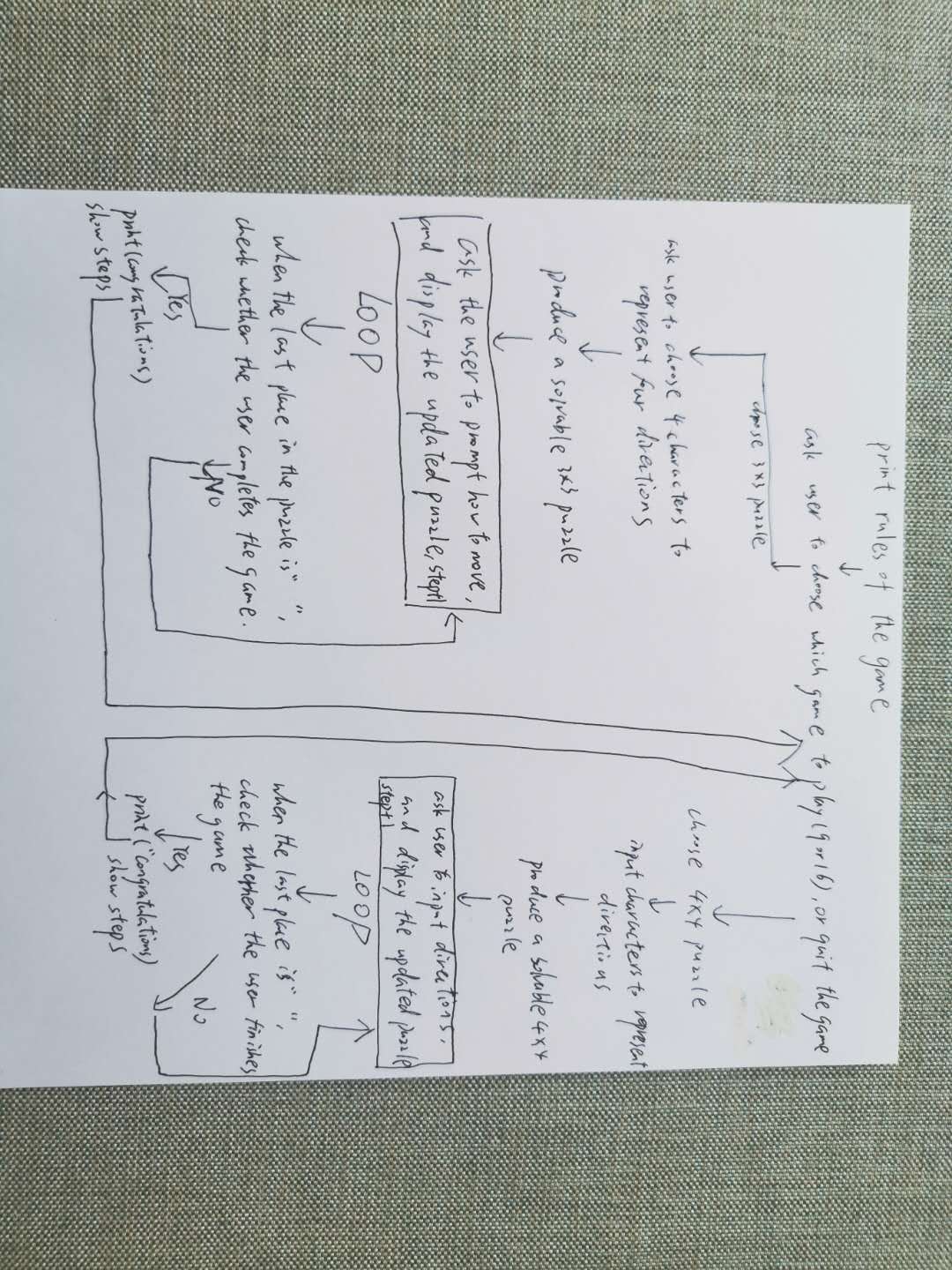
After each move, step will plus one. When the puzzle is solved, it will display how many steps the user uses.

* + 1. Directions and dirention

Directions is a string that contains all the characters the user originally input. Direction is a list that stores each characters that user input to represent directions as one single string .

* 1. description of the program flow

First, display the rules of the game, and let the user choose which puzzle he/she wants to play. Then, ask the user to input the letters that are used to represent the directions. And then ask the user to choose a puzzle to play. After the game starts, the computer will use if statement to check where the white space is and ask the user to input some specific moving orders based on the place of the white space. Based on the input, the computer changes the places of two particular numbers. After each move, the step will plus one. If the user input some wrong directions, the program will ask the user to input again. Specially, If the white place is in the last place of “list”, then the computer needs to check whether the numbers are in the correct order. If yes, then it will display: the number of step, congratulations and ask the user to continue another game or end the game. If no, continue to ask the user to input a direction to move.



* 1. How to generate the randomized puzzle

1. 8-number puzzle

Use for loop and if statement to continuously compare each pair of numbers in list2. Finally, count the inverse number of the randomly generated numbers in the list. If the inverse number is even, then directly display the numbers and white space in list (3 numbers per line). If the inverse number is odd, the reproduce the numbers in list, and check the inverse number until it is even. Only when the inverse number is even can the numbers and white space in list be displayed.

1. 15-number puzzle

First count the inverse number. If the inverse number is even, then the differences between the row number where the white space initially at and the number of the last row must be even. If the inverse number is odd, then the differences between the row number where the white space initially at and the number of the last row must be odd. If the parity doesn’t match each other, then reproduce the list until the rules are satisfied.

# Functions

1. output(number):

This function is used to display the original and updated puzzle. If the user is playing 8-number puzzle, then the parameter number is 9. If the user is playing 15-number puzzle, then the parameter number is 16. User doesn’t need to input the number, the computer will

input 9 or 16 automatically.

1. produce\_9number():

This function is used to generate a randomized, solvable 8-number puzzle.

1. produce\_16number():

This function is used to generate a randomized, solvable 15-number puzzle.

1. order():

This function is used to ask the user to input four letters to represent left, right, up, down respectively.

1. move(p\_number,p\_puzzle\_number):

This function is used to tell the computer to change the places of two particular numbers when the user input a direction. P\_puzzle\_number is the argument of the function output(). Thus p\_puzzle\_number equals to 9 if the user plays the 8-number puzzle. p\_puzzle\_number equals to 16 if the user plays the 15-number puzzle. P\_number is the difference of the index between the number in the same column and in the nearest row. Thus, the p\_number for the 8-number puzzle is 3 and the p\_number for the 15-number puzzle is 4.

1. instruction\_move\_9(): and instruction\_move\_16():

These two functions are to display which direction can be chosen to move and ask the user how to move. If the user input the wrong direction, the computer will ask the user to input again. After each move, the computer will display the updated the puzzle. The step number will plus one after each move. When the puzzle is solved, the computer will ask the user to play another game or quit the game.

1. main\_play\_9(): and main\_play\_16():

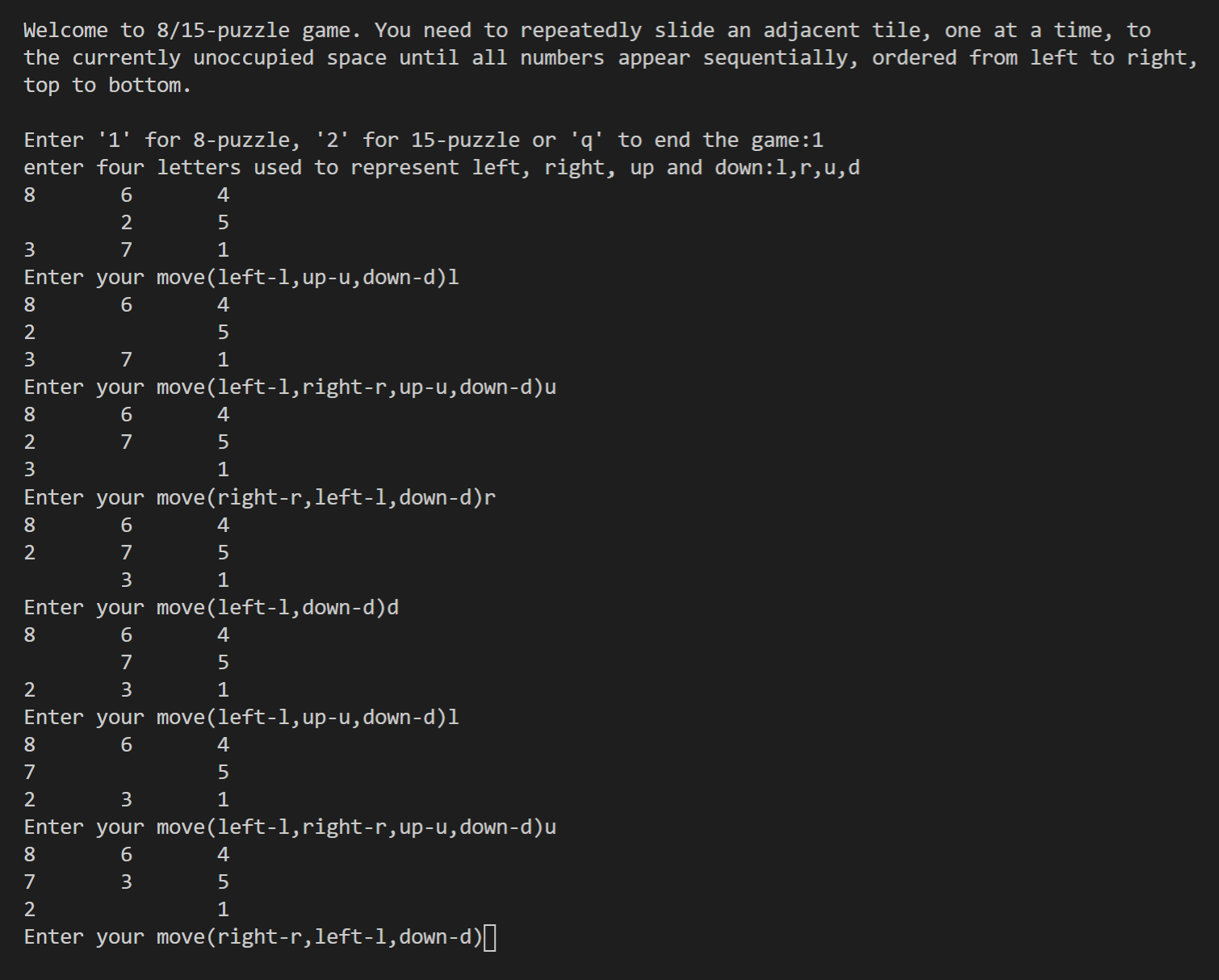
Main\_play\_9() combines order(), produce\_9number(), instruction\_move\_9(). This is the complete process of game: 8-number puzzle.

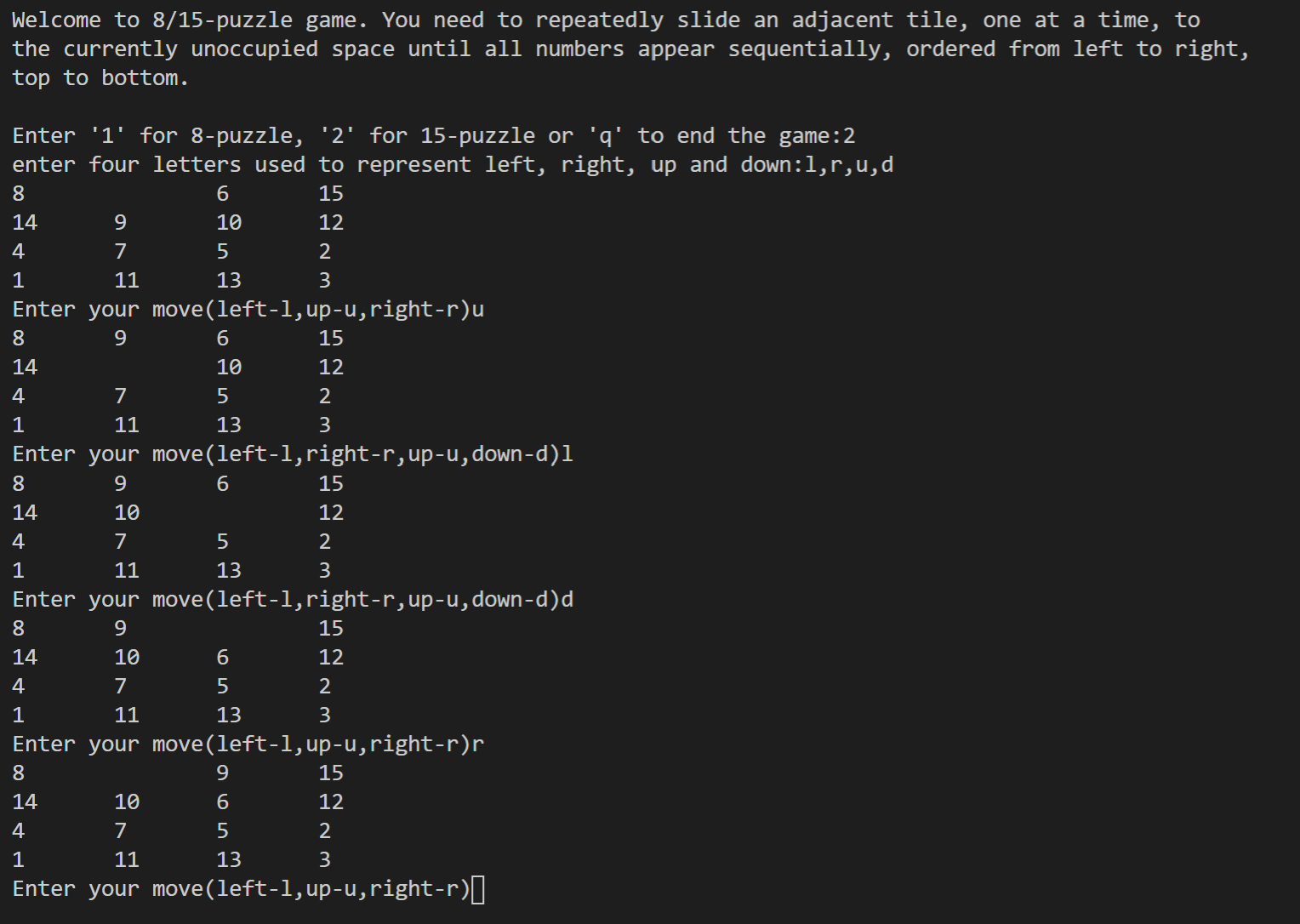
Main\_play\_16() combines order(), produce\_16number(), instruction\_move\_16(). This is the complete process of game: 15-number puzzle.

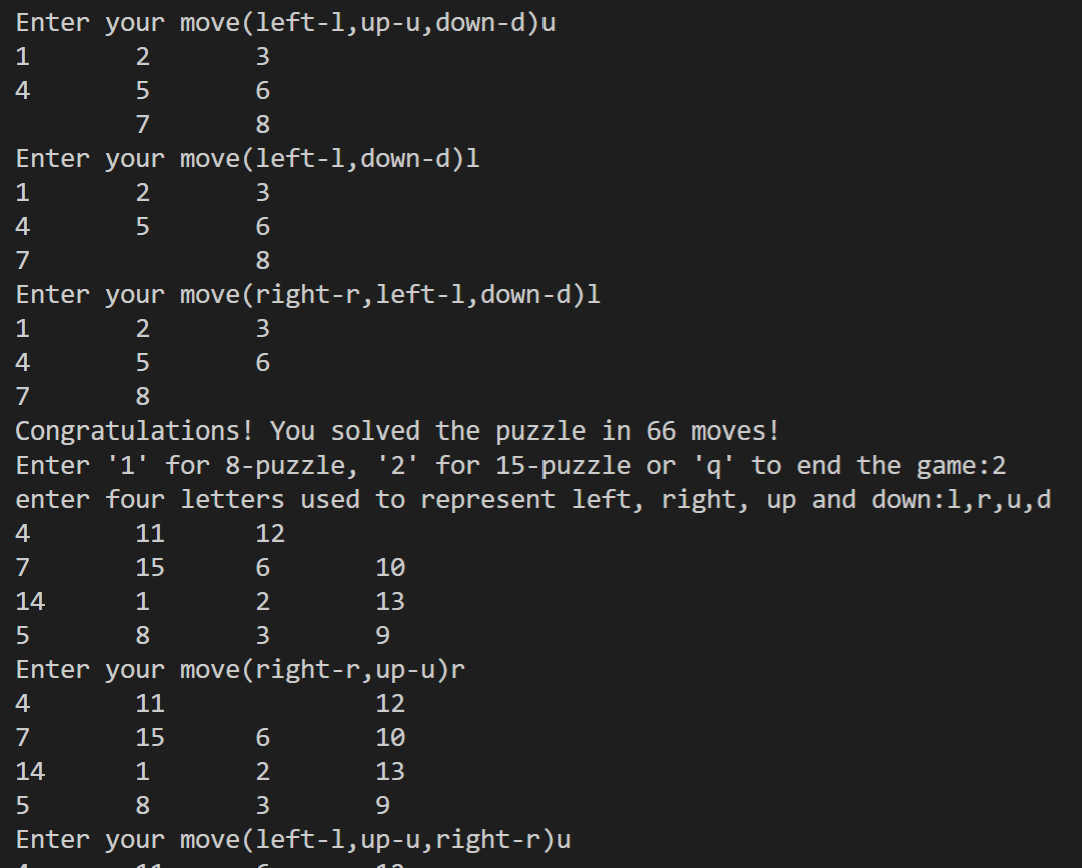
1. main():

Ask the user to play which game and start the game, or quit the game. Main\_play\_9() and main\_play\_16() are in this function.

# Sample Output







Make the program robust enough:

