**Design Doc**

**OVERVIEW**

• The program, named as sliding puzzle, I designed and developed an interactive

game where users can choose any dimensions up to 10x10, minimum dimension is 3x3. A n\*n square-framed board consists of (n^2-1) square tiles, numbered 1 to (n^2-1), initially placed in rando order. The board has an empty space where an adjacent tile can be slid to. 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), to the currently unoccupied space (the empty space) until all numbers appear sequentially.

**DATA MODEL**

• list, integer, string

• functions, parameters

• while loop, for loop

• if, elif, else, try, except

• import random

**PROGRAM STRUCTURE**

• generate a numbers list and randomize it

• count the inverse number and identify the position of the unoccupied space to make sure that the puzzle is solvable

• print them in a n\*n square

• get four direction keys from the user and move according to the user’s command

• decide whether the puzzle is done and ask whether to play again

**PROCESSING LOGIC (SPECIFIC)**

• Main processing logic:

1. Use get\_dimension() function to get the parameter n\_line.
2. Use get\_four\_direction\_lettters() function to get the parameters left, right, up and down.
3. Use generate\_a\_list() function containing a for loop to list the numbers from 1 to n\_line^2-1.
4. Use generate\_solvable\_puzzle(n\_line) function to get parameters “numbers” and randomize the list “numbers” until the puzzle is solvable. Then use the odevity of inverse number, distance of the empty space from its initial position and “n\_line” to check that whether the puzzle is solvable.
5. Use the printPuzzle() function to transfer the list to a square metrix.
6. The move() function will change the order in the list “number” according to the user’s input.
7. When the puzzle get solved, my program will tell the user total steps he made and ask him whether to play again.

• Initial Puzzle: get the number “n\_line” which represents the “n\_line”\*”n\_line” dimension from the user and use for loop to generate the list. The list is from 1 to (n\_line^2-1). Then, use the random imported to randomize it. Finally, print the puzzle out in the form of square.

**FUNCTIONAL SPEC**

1. generate\_a\_list():

List the numbers from 1 to n\_line^2-1.

Parameter: n\_line.

Return: numbers, numbersInOrder.

1. printPuzzle():

Transfer the list to a square metrix.

Parameter: \_n\_line, \_numbers

1. get\_dimension():

get the parameter n\_line

Return: n-line.

1. get\_four\_direction\_lettters():

Get four symbols which represents for directions to move.

Return: left, right, up and down.

1. countinverseNumber():

Count the inverse number in the “number” list.

Parameters: \_n\_line, \_numbers

Return: inverseNumber

1. generate\_solvable\_puzzle()

Randomize the list “numbers” until the puzzle is solvable. Then use the odevity of inverse number, distance of the empty space from its initial position and “n\_line” to check that whether the puzzle is solvable.

Parameters: \_n\_line

Return: numbers, numbersInOrder

1. move():

This function is to tell legal direction of movement.

Parameters: p: int, character index of ' ' in numbers list

left: string, The left button

right: string, The right button

up: string, The up button

down: string, The down button

n\_line: int, Meaning n\_line rows n\_line columns puzzle

Return: There is no return value

1. run():

Get five parameters: left, right, up, down and n\_line

Package the functions above in this function and arrange them in the right logic using while loop. Also, it can count the steps you move. Meanwhile, it can detect whether the puzzle get solved and then stop the game.

1. main()

Try the game again if nedded.

**SAMPLE OUTPUT**







