1. **Design**
   a. **Overview**

      The project is about a game, that is a sliding puzzle itself. A sliding puzzle is combination puzzle that challenges a player to slide (frequently flat) pieces along certain routes to establish a certain end-configuration. The pieces to be moved may consist of simple shapes, or they may be imprinted with colors, patterns, sections of a larger picture (like a jigsaw puzzle), numbers, or letters.

      To design this interactive sliding puzzle game, Python is used. Python is a high-level programming language which was created by Guido van Rossum (and the community) and released in the late of 20th century. There are some reasons why the language is used to develop the game. Those include easy-to-implement code and designation of readability.

   b. **Data Model**
      1) Boolean
         - In Python, boolean variables are defined by the True and False keywords. Though it is also possible to use the integer type of variable (with 0 as False and 1 as True), boolean variables save up the allocated memory. For example, notAvailable = True
         - In the program, boolean is used to evaluate a True/False value. In its example, it is used to determine whether the user wants to reset the game.
      2) Integer
         - In Python, integer is a number data types that store numerical value. For example, num = 5
         - This sliding puzzle is the numbers-pattern type; therefore, it uses integer variable to store the current value of the grid (with the help of list). Not only that, integer is also used to maintain the pointer when the puzzle is randomly generating.
      3) List
         - In Python, list contains items which separated by comma and enclosed within the square brackets [ ]. It is possible to access the value inside using the slice operator [ ] and [:] with indexes starting at 0 in the beginning of the list. For example, lst = [1, 2, [1, 2]]
         - In the program, list is one of the important keys used. List maintains the current grid of puzzle. Not only that, list is also used while generating the random puzzle pattern.
      4) String
         - In Python, strings are identified as a contiguous set of characters represented in the quotation marks. For example, str = “hello!”
         - This program uses string in many aspects. Those are inputs from user, binding the keys to move in any direction available, and some messages.

   c. **Program Structure**
      1) Line 6-11
         - This clearScreen() function is used to clear the terminal / screen.

2) Line 13-22
   - This fakeLoading() function is used to load a specific message from its parameter. It seems that it is not necessary to build-up this function. However, combine this with the clearScreen() function increases the engagement in user interface.
3) Line 24-27
   - This printDelay() function is nothing but only to curtail the whole code without having to repeatedly write the same typical part. For example, print("test") time.sleep(0.5) can be shortened with printDelay("test"). Mainly, to avoid the spamming messages over an instant, delaying message is needed.
4) Line 29-34
   - This blankCoordinate() function is used to search the blank part of the puzzle. It is essential to move the puzzle, as the exact location is needed. It is also possible to maintain 2 variables x and y which takes a little memory.
5) Line 36-38
   - This isComplete() function is used to determine whether the current puzzle is already solved or not.
6) Line 40-42
   - This isComplete1D() function is used to determine whether the 1D version of puzzle is already solved or not. It will be explained later the reason why we need both this function and the isComplete() function.
7) Line 44-55
   - This inputPrompt() function prompts the user to input the integer dimension of puzzle between 3 and 10 (inclusively). The user will be asked to input another integer until the condition is satisfied.
8) Line 57-73
   - This inputKeybindMovePrompt() function prompts the user to input 4 letters to bind them with direction moves. Again, the user will be asked to input another 4 letters until the condition is satisfied.
9) Line 75-93
   - This inputMovePrompt() function prompts the user to input the one of the 4 letters that was previously inputted while binding the keys. Additionally, both reset and exit buttons are provided. The function provides the available current moves. After processing, it returns 0, 1, 2, 3, 4, and 5 for up, right, down, left, exit, and reset respectively. Any random inputs will be ignored and the user will be asked to input another commands until the condition is satisfied.
10) Line 95-117
   - These moveUp(), moveRight(), moveDown(), and moveLeft() functions are used to move the puzzle in the respective direction. It will be called right after the inputMovePrompt() function is called.
11) Line 119-126

- This proceed() function prompts the user to input either y or n as yes and no. This used to proceed the reset and exit buttons.

12) Line 128-143
    - Both totalInversion() and isSolvable() functions are used to determine whether the puzzle is solvable or not. Most of the ideas are implemented through the inversion algorithm.
13) Line 145-152
    - This generateRandomPuzzle() function is used to generate any random puzzle. However, the function will not stop until it generates the solvable puzzle (with the help of isSolvable() function).
14) Line 154-167
    - This printPuzzle() function is used to print the whole grid of puzzle. It contains the clearScreen() function inside.
15) Line 171-184
    - This part provides the “welcome” message for the user and a brief explanatory of the game.
16) Line 186-199
    - This part initializes the whole puzzle. Inputs such as the dimension and keybind settings are included.
17) Line 200-213
    - This is the part of the game. This part uses some important game functions. In case the user wants to reset or exit the game, proceed() function will be called. It will proceed the commands if user inputs “y”, else it will cancel the command and return to the main game.
18) Line 216-220
    - This part is nothing but a congratulation message after the player finishes the game.
19) Line 221-231
    - This part allows the user to either play another game or to exit. It allows P, p buttons to play another game and X, x to quit the game. Another input will be asked until the correct command is found.

d. Processing Logic

1) Main processing logic

    The program will give brief explanatory about the sliding puzzle game itself. It also tells the user how to complete the game. Succinctly, An [Enter] button is needed from the user to confirm that they had already understood the game. Then, the program prompts the user to input N as the dimension for the puzzle (N x N). The dimension is available only in the range of 3 and 10 (inclusive). After that, the user will be asked to input the keybind settings for the direction movements.

    After the inputs, the program will generate the puzzle randomly. An list of 1D is used to take the random.sample() provided from the library. In order to maintain the quality of the game, solvable puzzles are needed; therefore, the isSolvable() function is used. The program generates the random puzzle until it is known to be solvable (using the inversion counts algorithm). Mathematically, the expectation of this part will not take any longer than 10^{-4} seconds (proved with million samples of computations). After generating, the puzzle is converted from 1D list to 2D list (called as a table) using for loops. In the main part, the program maintains some variables including the moves count for the whole game. The count moves will be increased by 1 (countMoves = countMoves + 1). To move, users are allowed to input one of the available moves. While moving, only the provided commands will be executed. The game runs until the condition of the puzzle is the same as the corrected one (formally, gridPuzzle == [[(x + y * N + 1) % (N * N) for x in range(N)] for y in range(N)]). For every moves, some functions will be used to swap the values. The values that will be swapped are the blank one, and the adjacent that share the same side. To maintain the quality of the game, every new moves will clear the current terminal and print the new state of puzzle. After the game is over, this program allows user to choose either to play another game or quit the current game.

2) Technique to generate the randomized puzzle

The program first creates an empty list that will be later assigned some values. The values are randomly ordered from 0 to N^2-1. The random.sample() function is used to generate a permutation from the list. After randoming the puzzle, the program will check whether it is solvable or not. The algorithm follows a lemma which describes the unique interpretation for every N (as the dimension), location of the blank from the bottom of row, and the number of inversions (from the 1D list). The program generates until the condition of solvable is satisfied.

2. Function Specification

1) def clearScreen():
This function clears the terminal using the imported system command from Python library. If the name is equal to ‘nt’ the OS is used Windows which uses ‘cls’. Else, for the Linux and macOS, ‘clear’ commands are used.
2) def fakeLoading():
To make the fake loading screen, a string contains = “…..” is used. While printing, for loops in the string bar allows the tucking of time.sleep() function to the code. Flush is needed to let the function prints every number of seconds determined.
3) def printDelay():
Delaying a message function can be built using only both print() and time.sleep() functions that are provided in Python library.
4) def blankCoordinate():
This function mainly returns the pair of integers x and y. Both x and y are the index where the blank is currently set. To find it, nested for loops is required by checking the contained values in the 2D list. gridPuzlee[i][j] inside for loops j inside the for loops i indicate the number of gridPuzzle in the position i rows from the beginning and j columns from the beginning. If gridPuzzle[i][j] equals to 0, then x = i and y = j.
5) def isComplete(): and def isComplete1D():

This function checks whether the current puzzle is solved or not yet completed. To check that, the program needs only to compare the correct completed puzzle with the current one. The correct puzzle is determined by:

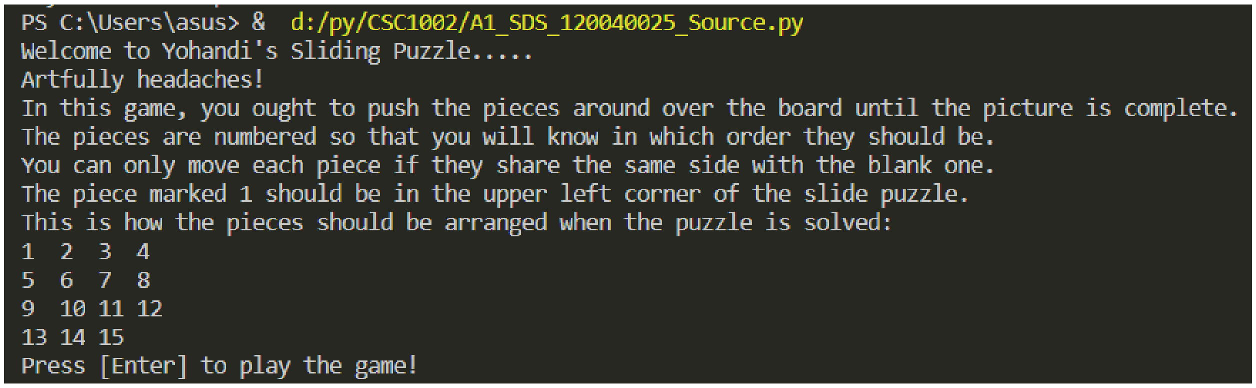
- [[(x + y * N + 1) % (N * N) for x in range(N)] for y in range(N)] for 2D
- [(x + 1) % (N * N) for x in range(N * N)] for 1D

6) def inputPrompt():, def inputKeybindMovePrompt():, and def inputMovePrompt():
Those 3 functions are actually similar, but only in different use. To prompt the users from inputting the wrong constrained inputs, while True is used. While True is an infinite loop. Since, the function is designed to do so, it will only break or return until the users input the exact required input. Therefore, “if” is needed inside the loops in order to determine whether the inputs are correct or not.
7) def moveUp():, def moveRight():, def moveDown():, and def moveLeft():
These 4 functions are used to move the piece towards the blank part. It is essential to note that the blank part is symbolized by number 0. To swap it, the programs need to look up where the 0 is positioned by using the blankCoordinate() function. After that the swap program is as simple as a, b = b, a.
    - $x + 1 < N$ must be satisfied for moveUp
    - $y – 1 >= 0$ must be satisfied for moveRight
    - $x – 1 >= 0$ must be satisfied for moveDown
    - $y + 1 < N$ must be satisfied for moveLeft
8) def proceed():
This function is used to ask the confirmation of users. To do that, the program simply do an infinite loop using while True (as described previously) and using if to check the input “y” and “n”.
9) def totalInversion():
This function returns the number of inversion in the list. The function itself seems does not have any use without any algorithm back it up. To count the number of inversion, the program do a nested loops whereas if gridPuzzle1D[i] > gridPuzzle1D[j] for every j > i (for j loops in for i) the total is added by 1.
10) def isSolvable():
According to a well-known lemma, if N (dimension) is even, then it is solvable if the total inversions + zth row from bottom is odd. If N is odd, then it is solvable if the total inversions is even.
    - (totalInversion() + N – x) is odd, N is even
    - (totalInversion()) is even, N is odd
11) def generateRandomPuzzle():
This function generates a random puzzle from 0 to N^2-1 where 0 denotes the blank where it is possible to swap the number in that spot. To do that, the program uses the random.sample() function that is provided from the Python library. The program keeps generating in the infinite loop and it will break once it is solvable using isSolvable()
12) def printPuzzle():

To print the whole puzzle grid, the program uses nested for loops to print. If it finds the gridPuzzle equals to 0, then it print a blank space only in that position.

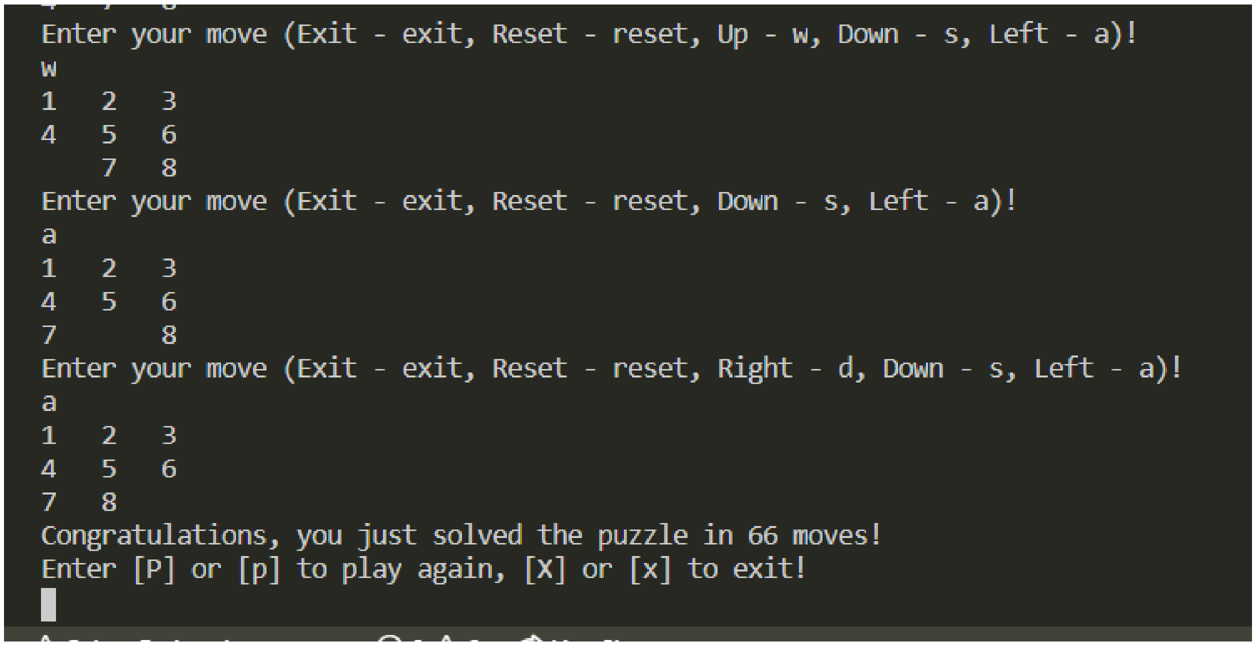
3. **Output**

1) Brief introduction



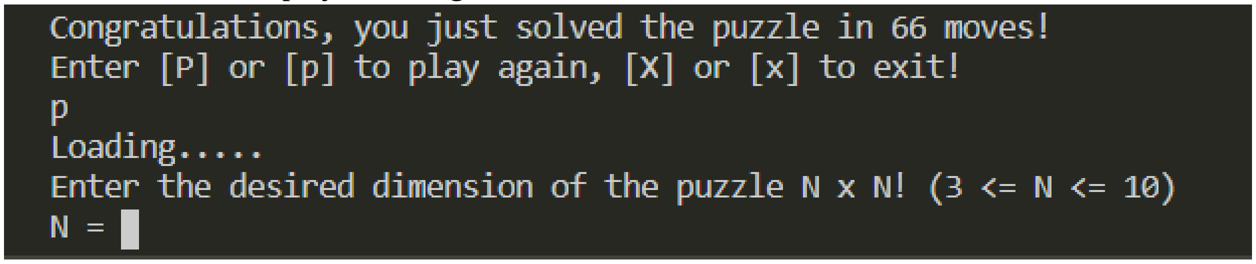
```
PS C:\Users\asus> &  d:/py/CSC1002/A1_SDS_120040025_Source.py
Welcome to Yohandi's Sliding Puzzle.....
Artfully headaches!
In this game, you ought to push the pieces around over the board until the picture is complete.
The pieces are numbered so that you will know in which order they should be.
You can only move each piece if they share the same side with the blank one.
The piece marked 1 should be in the upper left corner of the slide puzzle.
This is how the pieces should be arranged when the puzzle is solved:
1  2  3  4
5  6  7  8
9  10 11 12
13 14 15
Press [Enter] to play the game!
```

2) Process



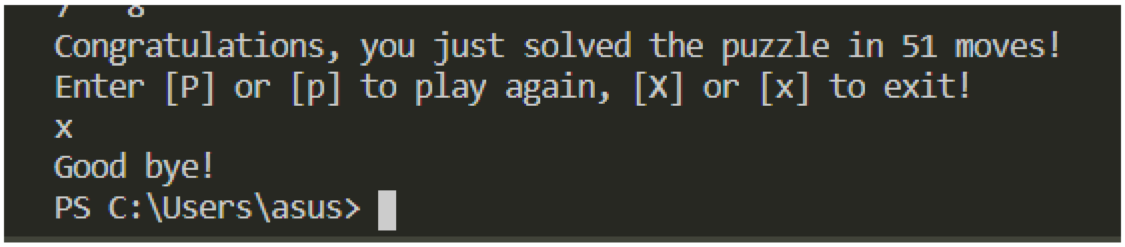
```
Enter your move (Exit - exit, Reset - reset, Up - w, Down - s, Left - a)!
w
1   2   3
4   5   6
    7   8
Enter your move (Exit - exit, Reset - reset, Down - s, Left - a)!
a
1   2   3
4   5   6
7       8
Enter your move (Exit - exit, Reset - reset, Right - d, Down - s, Left - a)!
a
1   2   3
4   5   6
7   8
Congratulations, you just solved the puzzle in 66 moves!
Enter [P] or [p] to play again, [X] or [x] to exit!
```

3) If the user decided to play another game



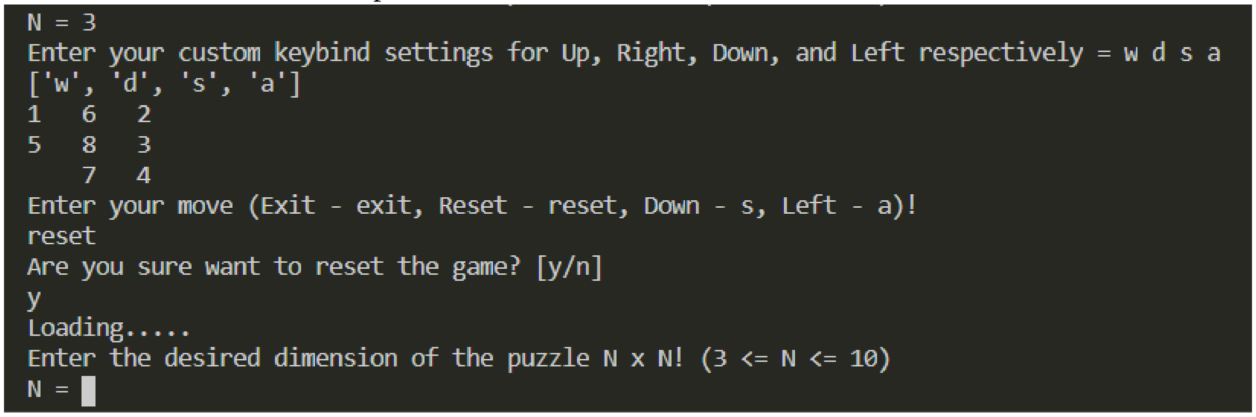
```
Congratulations, you just solved the puzzle in 66 moves!
Enter [P] or [p] to play again, [X] or [x] to exit!
p
Loading.....
Enter the desired dimension of the puzzle N x N! (3 <= N <= 10)
N =
```

4) If the user decided to quit the game



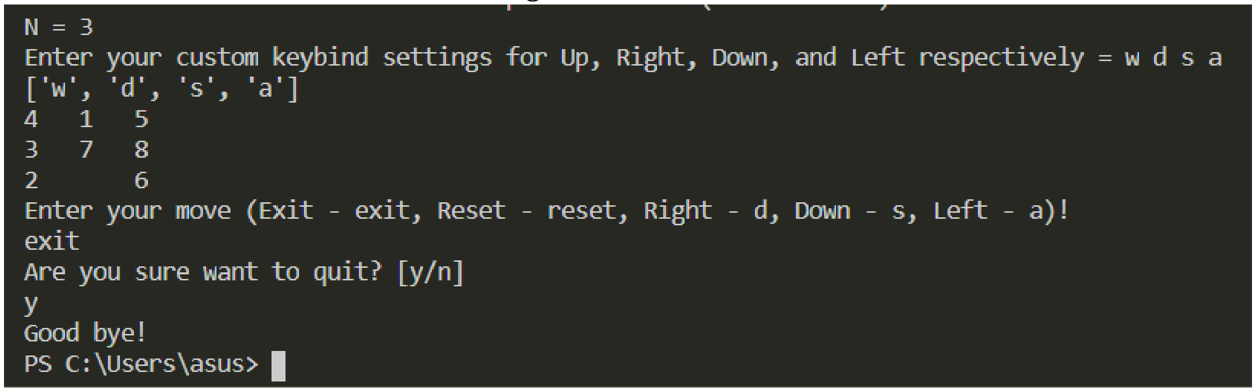
```
7   8
Congratulations, you just solved the puzzle in 51 moves!
Enter [P] or [p] to play again, [X] or [x] to exit!
x
Good bye!
PS C:\Users\asus>
```

5) If the user wants to reset the puzzle



```
N = 3
Enter your custom keybind settings for Up, Right, Down, and Left respectively = w d s a
['w', 'd', 's', 'a']
1   6   2
5   8   3
    7   4
Enter your move (Exit - exit, Reset - reset, Down - s, Left - a)!
reset
Are you sure want to reset the game? [y/n]
y
Loading.....
Enter the desired dimension of the puzzle N x N! (3 <= N <= 10)
N =
```

6) If the user wants to exit in the middle of game



```
N = 3
Enter your custom keybind settings for Up, Right, Down, and Left respectively = w d s a
['w', 'd', 's', 'a']
4   1   5
3   7   8
2       6
Enter your move (Exit - exit, Reset - reset, Right - d, Down - s, Left - a)!
exit
Are you sure want to quit? [y/n]
y
Good bye!
PS C:\Users\asus>
```

7) Example of text output:

```
PS C:\Users\asus> &  d:/py/CSC1002/A1_SDS_120040025_Source.py
Welcome to Yohandi's Sliding Puzzle.....
Artfully headaches!
In this game, you ought to push the pieces around over the board until the picture is
complete.
The pieces are numbered so that you will know in which order they should be.
You can only move each piece if they share the same side with the blank one.
The piece marked 1 should be in the upper left corner of the slide puzzle.
This is how the pieces should be arranged when the puzzle is solved:
1  2  3  4
5  6  7  8
9  10 11 12
13 14 15
Press [Enter] to play the game!
```

```
Loading.....
Enter the desired dimension of the puzzle N x N! (3 <= N <= 10)
N = 3
Enter your custom keybind settings for Up, Right, Down, and Left respectively = w d
s a
['w', 'd', 's', 'a']
3   6   8
4   2   1
5       7
Enter your move (Exit - exit, Reset - reset, Right - d, Down - s, Left - a)!
d
3   6   8
4   2   1
    5   7
Enter your move (Exit - exit, Reset - reset, Down - s, Left - a)!
s
3   6   8
    2   1
4   5   7
Enter your move (Exit - exit, Reset - reset, Up - w, Down - s, Left - a)!
a
3   6   8
2       1
4   5   7
Enter your move (Exit - exit, Reset - reset, Up - w, Right - d, Down - s, Left - a)!
a
3   6   8
2   1
4   5   7

.
.
.

1   2   3
4   5
7   8   6
Enter your move (Exit - exit, Reset - reset, Up - w, Right - d, Down - s)!
w
1   2   3
4   5   6
7   8
Congratulations, you just solved the puzzle in 71 moves!
```

```
Enter [P] or [p] to play again, [X] or [x] to exit!
x
Good bye!
PS C:\Users\asus>
```