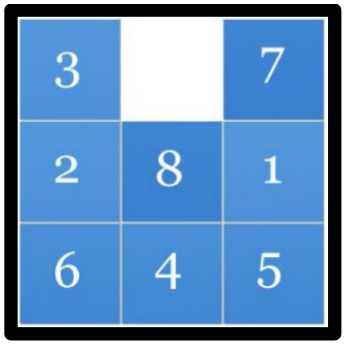
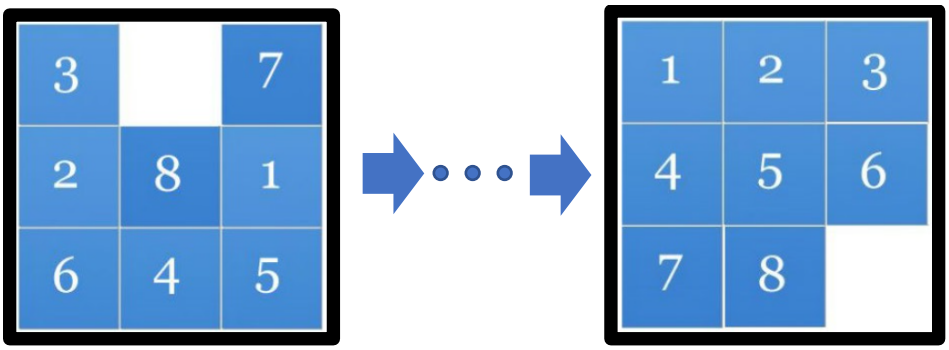
Sliding Puzzle - 2024



OVERVIEW

In this assignment, you are going to design and develop an interactive 8-tile sliding puzzle game. The sliding puzzle game contains a square-framed board consisting of 8 square tiles, numbered 1 to 8, and an empty space. The numbered tiles are placed in a random order. The objective of the game is to re-arrange the numbered tiles into a sequential order by their numbers (left to right, top to bottom) by repeatedly sliding one adjacent tile into the empty space. The following figure shows an example of one 8-number puzzle where "INITIAL" is the starting point of the game, and the player needs to repeatedly slide one adjacent tile, one at a time, to the unoccupied space (the empty space) until all numbers appear sequentially, ordered from left to right, top to bottom, shown as "FINAL".



SCOPE

1. At the start of the game, display a brief introduction about the game.
2. Display a prompt to ask the player to enter 4 letters (a-z), case-insensitive, used for the left, right, up and down moves. The player is free to pick any four letters such as j, k, r, f for left, right, up and down respectively. Your program must perform appropriate input validation such as repeated letters, too many or too few letters, non-letter characters and so on. Spaces are valid in the input such as " l r u d ", "lrud", " l ru d ", " L r U D " and so on.
3. After the successful prompt then generate a randomized, SOLVABLE 8-tile puzzle accordingly; display the puzzle on the screen in a simple text format of rows and columns of numbers (see Sample Output)
4. Repeatedly prompt the player to choose the tile to slide to the empty location (left, right, up or down). In the prompt the valid sliding move(s) are shown together with the designated letter (from step 2 above). For example:
    a. Enter your move (left-j, right-k) >
5. After each move, show the updated puzzle on the screen and prompt further move if needed.
6. Inform player when the puzzle is solved (i.e. the numbered tiles are in sequential order, left to right, top to bottom); then prompt player to continue another game or terminate the program.
7. Track total number of moves made for each game and have it displayed as the puzzle is solved.
8. Validate all input as mentioned above, including whitespaces from the input such as leading and trailing spaces as well as spaces in between. Your program must not crash and must prompt player for input again with a friendly message indicating the detected error (if any).

NOTE:

- Keep your entire source code in ONE SINGLE file.
- Use only standard python modules
- In your design stick ONLY to functions, in other words, no class objects of your own.

STARTUP OPTIONS

Not applicable

SKILLS

In this assignment, you will be trained on the use of the followings:

- Problem Decomposition, Clean Code, Top-Down Design
- Functions (with parameters and return) for program structure and logic decomposition
- Input() - Input Validation & Program Robustness
- Standard objects (strings, numbers & lists)
- Control statements to interact with users
- Variable Scope
- String formatting (F-String, method, etc)

DELIVERABLES

1. Program source code (A1_School_StudentID.py)

where School is SSE, SDS, SME, HSS, FE, LHS, MED and StudentID is your 9-digit student ID.

Submit the python file by due date to the corresponding assignment folder under “Assignment (submission)”

For instances, a SME student with student ID “119010001” will name the python file as follows:

- A1_SME_119010001.py:

5% will be deducted if file is incorrectly named!!!

TIPS & HINTS

- Beware of variable scope as you might keep a few variables as global such as puzzle
- Refer to python website for program styles and naming convention (PEP 8)
- Validate all inputs - if incorrect input is detected then display a friendly response and prompt the input again.

SAMPLE OUTPUT

Welcome to Kinley's puzzle game, you will be prompted …..

….. (skipped)

Enter the four letters used for left, right, up and down move > l r u D

```
   1 3
4  2 5
7  8 6
```

Enter your move (left-l, up-u) > l

```
1    3
4  2 5
7  8 6
```

Enter your move (left-l, right-r, up-u) > u

```
1  2 3
4    5
7  8 6
```

Enter your move (left-l, right-r, up-u, down-d) > l

```
1  2 3
4  5
7  8 6
```

Enter your move (right-r, up-u, down-d) > u

```
1  2 3
4  5 6
7  8
```

Congratulations! You solved the puzzle in 4 moves!

Enter "n" for another game, or "q" to end the game > n

```
 8 1 3
 4   2
 7 6 5
```

Enter your move (left-l, right-r, up-u, down-d) >

.
.
.

MARKING CRITERIA

- Coding Styles – overall program structure including layout, comments, white spaces, naming convention, variables, indentation, functions with appropriate parameters and return.
- Program Correctness – whether or the program works 100% as per Scope.
- User Interaction – how informative and accurate information is exchanged between your program and the player.
- Readability counts – programs that are well structured and easy-to-follow using functions to breakdown complex problems into smaller cleaner generalized functions are preferred over a function embracing a complex logic with too-nested conditions and sub-functions! In other words, a design with clean architecture with high readability is the predilection for the course objectives over efficiency. The logic in each function should be kept simple and short, and it should be designed to perform a single task and be generalized with parameters as needed.
- KISS approach – Keep It Simple and Straightforward.
- Balance approach – you are not required to come up a very optimized solution. However, take a balance between readability and efficiency with good use of program constructs.

ITEMS	PERCENTAGE	REMARKS
CODING STYLES	15%-20%	0% IF PROGRAM DOESN'T RUN
USER INTERFACE	15%-20%	0% IF PROGRAM DOESN'T RUN
FUNCTIONALITY	MAX. 70%	REFER TO SCOPE

DUE DATE

March 3rd, 2024, 11:59:59PM