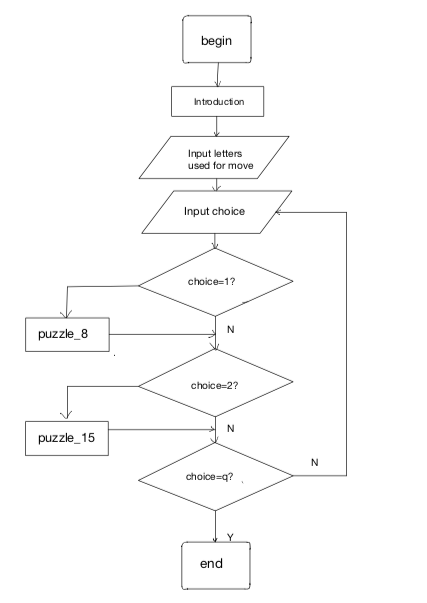
The whole project can be divided into three main parts—puzzle 8, puzzle 15 and main executive code.

1.

Firstly, in the main executive code part, which is the main part to run this game can be divided into Introduction, Direction Determination, and Interaction with Users.



To prompt user to enter four letters to control the direction the tile moving, I defined a “direction\_letters” function. Use “input” function to get users’ input, which return a string. Then consider (1) how a string can assign to four valuables, (2) what if the users enter same letter.

1. Transfer the string into list.

Use iteration to get what the user entered—denote as “letter”.

Then, use “isalpha” function to get all letters (excluding “,” or “ ” …).

Use “len” method to estimate whether the user entered four letters, if not, the user should enter again. Thus, we need a while loop.

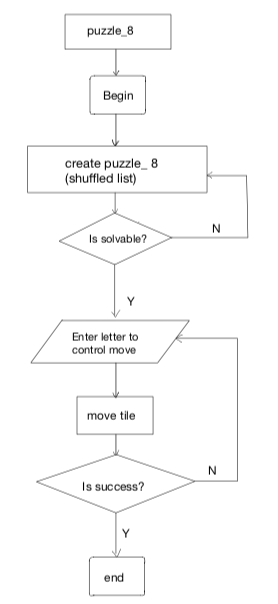
1. Same letter? Transfer the list into set. (list[“a”, “a”, “b”, “c”]🡪set(“a”, “b”, “c”) ) I use length of them to judge this situation. Continue the loop until it is satisfying.

Return l, r, u, d which are assigned with four correct letters.

To interact with user to determine which he choose to play or quit. Thus, I use a while loop to break the game until the user enter “q”. If user enters “1”, call “puzzle\_8” function, “2” for “puzzle\_15” function, “q” for breaking the loop.

2.

Secondly, I designed a puzzle\_8 function which can be divided into “Create puzzle”, “Move tiles”, “Display”, and “Success”.



1. To Create Puzzle, I use a list to contain 0~8, then, use “shuffle” method to shuffle it. Consider it is solvable? I defined an “isSolvable\_8” function with parameter of shuffled list. Use the number of inversions, say, except 0. e.g. [6, 3, 9, 0, 2], for 3, there is ONE number-6 before and bigger than 3; for 9, there is No number before bigger than it; for 2, there is THREE number-6, 3, 9 before and bigger than 2. Thus this list has the number of FOUR inversions. If the number of inversions, denoted as “sum\_reversions”, is even, then return True means this puzzle\_8 is solvable. If not return False.
2. Before move tiles, I should restructure this list. Replace 0 with “ ”. Define a “relist” function with parameter of a solvable list to return a 3 x 3 array [ [], [], [] ]. Then, I should display it–define a “print\_puzzle” function with parameter of a restructured list “out”.
3. To move tiles, I also should know the position of “ ”, denoted as pos\_0\_row and pos\_0\_col, every time printing after-move outcome. Thus, find pos\_0\_row and pos\_0\_col should be included into “print\_puzzle” function. Use iteration to print the 3x3 array and at the same time find “ ” ’s position. Define a “move” function with parameters of restructured list “out”, “pos\_0\_row”, “pos\_0\_col” to move adjacent tiles. {move up – “ ” move down: two tiles change position. e.g. “ ” is set in (0,0) then it should change position with what set in (1,0). The same logic for other direction movement.}

|  |  |  |
| --- | --- | --- |
| (0,0) | (0,1) | (0,2) |
| (1,0) | (1,1) | (1,2) |
| (2,0) | (2,1) | (2,2) |

1. Modify “move” function——

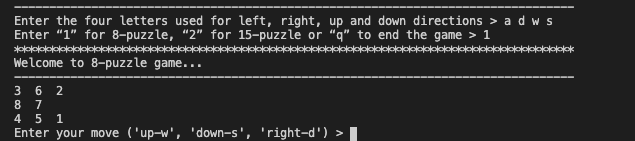
Suppose “ ” is set in (2,2), what if I enter the letter to move up? Obviously, pos\_0\_row+1 is out of range, so it will be an error! Use try…except method to modify it. The same for move light. Suppose “ ” is set in (0,0), what if I enter the letter to move down? pos\_0\_row-1 = -1, so (0,0) and (-1,0)/(2,0) will exchange, which is out of expectation. Thus, I add a conditional judgement pos\_0\_row -1 >= 0. The same for move right. Use will loop to break user’s input until it gets correct input.

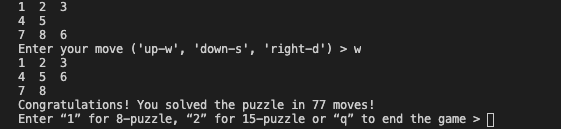
How can the user know which direction is allowed to move? I define another “getValidChoice” function with parameters of “out”, “pos\_0\_row” and “pos\_0\_col”, parameter “out” is the outcoming list changed by the user. Use pos\_0\_row and pos\_0\_col to judge what allowed movement and direction letter will be contained into a list “choice”. Call this function and print “choice” inside of “move” function, for every move should be told this information.

1. After move tile, it should display the result. To count the movement, count + 1.
2. Use while loop until it is solved. How to judge it is solved?

I defined a “isSuccess” function with parameter “out” and “count”, “out” is the outcoming list changed by user, and “count” is used to tell how many steps the user takes to success it. If it returns True, break the loop. In this function, I use match to estimate its success. If and only if match equals to 9, it is success. Match means for n(row, col), n = row\*3 + col +1, and for “ ”, setting in (2,2)

Test it…



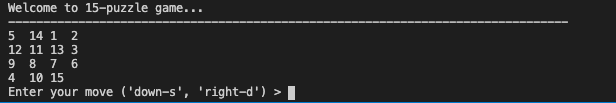


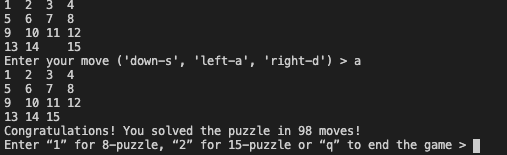
3.

Thirdly, I designed puzzle\_15. The only different of puzzle\_15 and puzzle\_8 is isSolvable function, other part can be modified into public usage.

When the number of inversions and “ ” ’s finally row minus “ ” ’s initial row both even or both odd, this puzzle is solvable.

Test it…





4.

Finally, modify the main code.

Test it…

