A L E X M A L A V E

Assignment: Homework #8

Problem 1:

Part I:
slidingpuzzle.py,

slidingpuzzle.py is intended to be used as a module by the driver program *game.py*. Within *slidingpuzzle.py,* a class, **SlidingPuzzle**, is defined. The constructor takes two arguments, **rows** and **columns**, out of which it creates lists within lists, with the length of each list corresponding to the number of columns, and the number of lists created within the main instance variable **self.__answer** corresponding to the number or rows.

The method **displayPuzzle** prints the puzzle onto the terminal in a legible, neat way. Some attempts are made in order to make single-digit numbers and two-digit numbers fit better within their respective columns.

The method **move** swaps the 0 from whichever list it is stored within with the value of the index provided by the user. To do so, it iterates through the puzzle's list of numbers, **self.__answer**, and finds the 0's index (relative to the list it lies within). The other number's index is assumed to exist within the board's limits due to the next method's use.

The method **legalMoves** ensures that the move the user is attempting is "legal," that is, it ensures that the "tile" the user wishes to swap the 0 for is directly adjacent to the 0's position at the given time. To do this, it iterates through self.__answer twice: once to find the 0's position indexes, and the second time to compare its indexes to those of adjacent "tiles." Returns a list of tuples that correspond to possible moves (possible indexes in the format (row, column)).

The method **scramble** does exactly that: it scrambles the board in order to make the puzzle, well, a puzzle. The way it does this is by moving the 0 around using the methods **move** and **legalMoves** an arbitrary number of times. The more times the puzzle is shuffled, the harder it becomes.

Finally, the method **isSolved** checks if the puzzle has been solved. If so, it returns true, otherwise, it returns false. To check that the puzzle is solved, it iterates through **self.__answer** and checks whether the previous term is smaller than the next.

Note: this code is not to be run individually, rather it is meant to be imported into a driver program, such as game.py.

Part II:
game.py

This file is meant to be used as the driver program for the module *slidingpuzzle.py*. It has one function, main, which initiates a game with the user using *slidingpuzzle.py*, the goal of which is to organize a board of numbers in such a way that all numbers are in order from least to greatest. The user can choose arbitrary values for the row and column sizes, and they can select their difficulty level as well (which refers to the number of times the puzzle is scrambled).

main uses a nested while loop to keep the game running, checking for illegal moves and whether the puzzle has been solved or not. The user is asked for a position for which that of the 0 can be swapped, and here, the indexes the user inputs are compared to those of the tuples within the list returned by the method **legalMoves** in *slidingpuzzle.py*. Should the user win, or solve the puzzle, the while loop ends, and “You win!” is displayed on the terminal. The code should be run as follows: *python3 game.py*

Extra Credit:

Part I:
pictureslidingpuzzle.py

Since much of this module is similar to *slidingpuzzle.py*, I'm discouraged from repeating myself. Instead, I'll point out the differences.

The constructor now takes an imageWindow object and an image filename, which it fragments depending on the number of rows/columns the user chooses. An image object is created for each “tile” and stored within **self.__tiles**. On the other hand, **self.__bounds** contains a list of tuples that denote the upper-left and lower-right coordinates, or bounds, of each tile accordingly. The “tile” that corresponds to the 0 is made white for clarity's sake.

The method **displayPuzzle** now draws each “tile” according to its position on the **self.__answer** list.

The method **returnTileIdx** returns the “tile” index of the “tile” within the bounds provided. This is used by the driver program in order to make moving “tiles” possible.

Finally, the method **returnDefaultSolution** simply prints out what would be the puzzle's solution. This is also utilized by the driver program.

Part II:
picturegame.py

As was the case with Part I of this extra credit, much of this file shares properties with the original *game.py* file. However, a notable difference is: asking the user to click on a “tile” rather than input numbers in order to make the “move” selection. The function **main** first grabs the “tile” number using **returnTileIdx**, and then checks for that “tile's” position in the default solution returned by **returnDefaultSolution**. Once the correct tile is found, its index is made into a tuple, which is broken down into row and column indexes, and then processed with the methods **legalMoves** and **move**. The rest of the function behaves much like before.