```
"""
Clone of 2048 game.
"""

import poc_2048_gui
import random

new_tile_count = 1
# Directions, DO NOT MODIFY
UP = 1
DOWN = 2
LEFT = 3
RIGHT = 4

# Offsets for computing tile indices in each direction.
# DO NOT MODIFY this dictionary.
OFFSETS = {UP: (1, 0),
        DOWN: (-1, 0),
        LEFT: (0, 1),
        RIGHT: (0, -1)}

#COFFSETS = {'UP': (1, 0),
#         'DOWN': (-1, 0),
#         'LEFT': (0, 1),
#         'RIGHT': (0, -1)}



def compress(line):
```

```
    """
    Helper function for implementing merge subroutine
    """
    lis = []
    for i in range(len(line)):
        lis.append(line[i])


    for i in range(0,len(line)-1):
        if lis[i] == 0:
            j = i


            while ( (j <= len(line)-1) and lis[j] == 0 ):
                j += 1
            if j <= len(line)-1:
                lis[i] = lis[j]
                lis[j] = 0
    return lis


def merge(line):
    """
    Helper function that merges a single row or column in 2048
    """
    line = compress(line)
    for i in range(1,len(line)):
        if line[i-1] == line[i]:
            line[i-1]*=2
            line.pop(i)
            line.append(0)
```

```
    return line



class TwentyFortyEight:
    """
    Class to run the game logic.
    """

    def __init__(self, grid_height, grid_width):
        self._height = grid_height
        self._width = grid_width
        self._grid = [[0 for dummy_x in range(self._width)]for dummy_x in range(self._height)]
        self.reset()
        self.dict_for_dir = self.__initialise_dict__()
        self.new_tile_count = 0


    def __initialise_dict__(self):
        list_up=[]
        list_down=[]
        list_left=[]
        list_right=[]
        for dummy_grid_i in range(1):
            for grid_j in range(self._width):
                list_up.append((0,grid_j))

        for dummy_grid_i in range(1):
```

```
        for grid_j in range(self._width):
            list_down.append((self._height-1,grid_j))

    for dummy_grid_i in range(1):
        for grid_j in range(self._height):
            list_left.append((grid_j,0))

    for dummy_grid_i in range(1):
        for grid_j in range(self._height):
            list_right.append((grid_j,self._width-1))

    dictionary = {UP:list_up, DOWN:list_down, LEFT:list_left, RIGHT:list_right}
    return dictionary




def reset(self):
    """
    Reset the game so the grid is empty.
    """
    for dummy_i in range(self._height):
        for dummy_j in range(self._width):
            self._grid[dummy_i][dummy_j]=0



def __str__(self):
```

```
    """
    Return a string representation of the grid for debugging.
    """
    lis = []
    #list which is going to be converted in string
    for dummy_i in range(self._width):
        for dummy_j in range(self._height):
            lis.append(self.get_tile(dummy_i,dummy_j))
    return (str(lis))

def get_grid_height(self):
    """
    Get the height of the board.
    """
    return self._height

def get_grid_width(self):
    """
    Get the width of the board.
    """
    return self._width



def move(self, direction):

    """
    Move all tiles in the given direction and add
    a new tile if any tiles moved.
    """
```

```
stir = direction
length = len(self.dict_for_dir[(direction)])
change = False

for i in range(length):

    lis = []
    row = self.dict_for_dir[stir][i][0]
    col = self.dict_for_dir[stir][i][1]
    while( row < self._height and col < self._width and row >= 0 and col >= 0 ):
        lis.append(self._grid[row][col])
        row += OFFSETS[stir][0]
        col += OFFSETS[stir][1]

    lis_lis = merge(lis)
    row = self.dict_for_dir[stir][i][0]
    col = self.dict_for_dir[stir][i][1]
    count = 0

    while ( row >= 0 and row < self._height and col >= 0 and col < self._width):
        if (self._grid[row][col] != lis_lis[count]):
            change = True
        self._grid[row][col] = lis_lis[count]
        row += OFFSETS[stir][0]
        col += OFFSETS[stir][1]
        count+=1

if ( change == True):
            self.new_tile()
```

```
def new_tile(self):
    """
    Create a new tile in a randomly selected empty
    square.  The tile should be 2 90% of the time and
    4 10% of the time.
    """
    ls_pair=[]
    rand_generator = [2,2,2,2,2,2,2,2,2,4]
    self.new_tile_count += 1
    print (self.new_tile_count)
    for row_i in range(self._height):
        for col_i in range(self._width):
            if ( self._grid[row_i][col_i] == 0):
                k = (row_i,col_i)
                ls_pair.append(k)



    grid_ij = random.choice(ls_pair)


    self._grid[grid_ij[0]][grid_ij[1]] = random.choice(rand_generator)


def set_tile(self, row, col, value):
    """
    Set the tile at position row, col to have the given value.
    """
```

```
        self._grid[row][col] = value



    def get_tile(self, row, col):
        """
        Return the value of the tile at position row, col.
        """
        return self._grid[row][col]



poc_2048_gui.run_gui(TwentyFortyEight(4, 4))
```