Programming Assignment 5

Main Module (game.py):

- Has the main GUI program

ADT Fifteen (fifteen.py):

- Has one class Fifteen

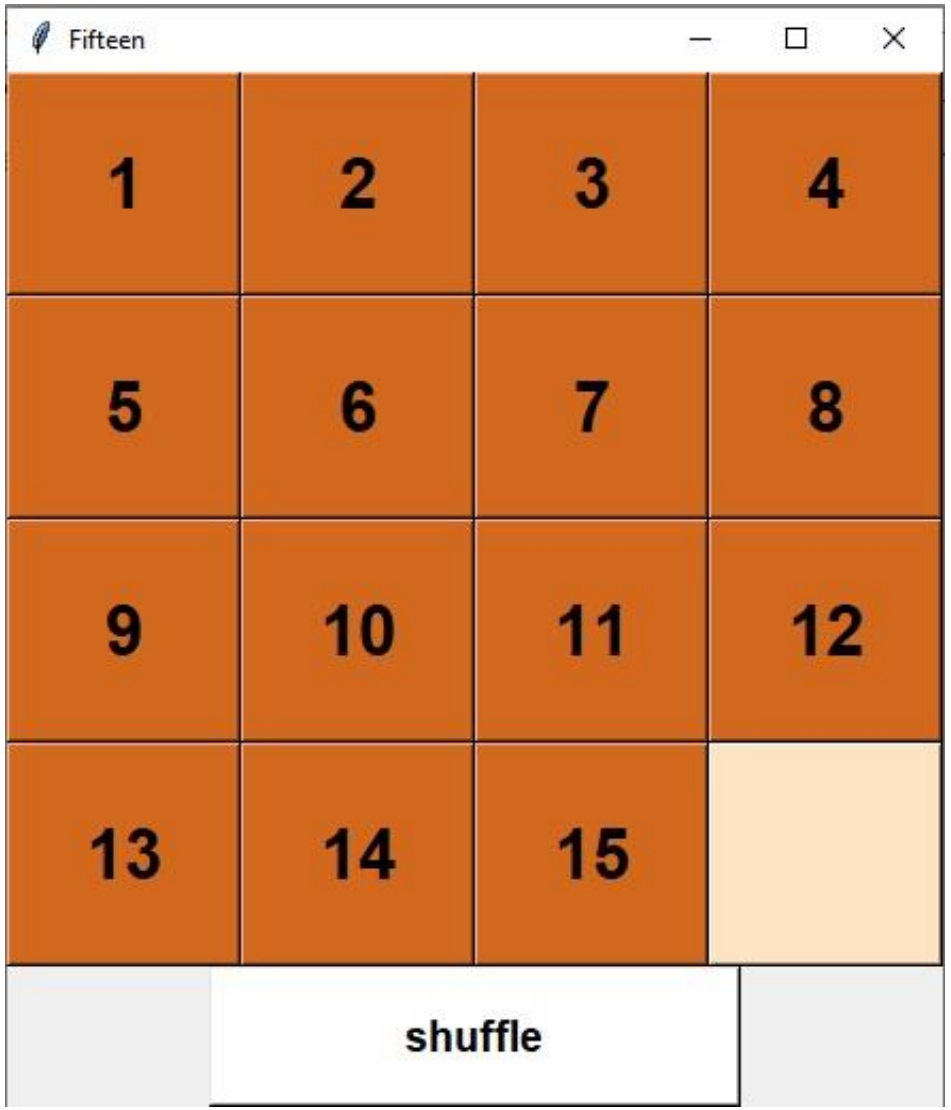
ADT Graph (graph.py):

- Has two classes Vertex and Graph

Main Module: GUI Application

In the main, you should create:

1. a GUI window
2. an object Fifteen
3. 15 buttons (tiles labeled from 1 to 15), one button without a label (empty space), one button to shuffle the tiles
4. 17 labels (StringVar objects) to label the tiles and other buttons
5. fonts
6. functions to shuffle tiles, to add buttons, to transpose a tile with an empty space, etc.



Main Module: GUI Application

You can use the following import statements:

from tkinter import *

import tkinter.font as font

from fifteen import Fifteen

from random import choice

Main Module: GUI Application

```
if __name__ == '__main__':
    # make a board with tiles
    board = Fifteen()
    empty = ?
    # make a GUI window
    gui = Tk()
    gui.title("Fifteen")
    # make fonts
    font1 = font.Font(family='Helveca', size='25', weight='bold')
    # make buttons with labels
    labels = ? # use StringVar()
    # you can update labels
    #  for example, labels[15].set(' ')
    buttons = ? # use method add_button()
    # you can modify buttons
    buttons[15].configure(bg=color2)
    # arrange buttons on the grid
    ?
    # add a button shuffle to shuffle the tiles
    ?
    # update the window
    gui.mainloop()
```

ADT Fifteen: Module fifteen.py

The ADT Fifteen contains the following instance methods:

1. __init__() creates an instance Fifteen with a board (can be a Graph object or array/list)
2. __str__() returns a string representation of the board with 15 tiles
3. draw() draws the boards with 15 tiles to stdout
4. update(move) move the selected tile (move) to the empty space
5. is_valid_move(move) returns True if the move can be done (legitimate), otherwise return False
6. is_solved() returns True if the puzzle is solved
7. shuffle(steps=30) shuffles the tiles using only legitimate moves, the default value of random transpositions should be set to 30

ADT Fifteen: Optional Methods

Additional methods of the ADT Fifteen:

1. transpose(i, j) transposes two tiles (can be a tile and the empty space)
2. solve() solves the puzzle
3. is_solvable() returns True if the puzzle is solvable, otherwise returns False

ADT Fifteen: Testing

Your ADT Fifteen implementation should work with the following code and the drive code in the template file fifteen.py:

```
game.shuffle()
game.draw()
while True:
    move = input('Enter your move or q to quit: ')
    if move == 'q':
        break
    elif not move.isdigit():
        continue
    game.update(int(move))
    game.draw()
    if game.is_solved():
        break
print('Game over!')
```

ADT Fifteen: Output of the Game

```
+---+---+---+---+
| 1 | 2 | 3 | 4 |
+---+---+---+---+
| 5 | 6 | 7 |   |
+---+---+---+---+
| 9 |10 |11 | 8 |
+---+---+---+---+
|13 |14 |15 |12 |
+---+---+---+---+
Enter your move or q to quit: 8
```

```
+---+---+---+---+
| 1 | 2 | 3 | 4 |
+---+---+---+---+
| 5 | 6 | 7 | 8 |
+---+---+---+---+
| 9 |10 |11 |   |
+---+---+---+---+
|13 |14 |15 |12 |
+---+---+---+---+
Enter your move or q to quit: 12
```

```
+---+---+---+---+
| 1 | 2 | 3 | 4 |
+---+---+---+---+
| 5 | 6 | 7 | 8 |
+---+---+---+---+
| 9 |10 |11 |12 |
+---+---+---+---+
|13 |14 |15 |   |
+---+---+---+---+
Game over!
```

ADT Graph

1. Graph and Vertex classes should be written in the graph.py module.
2. The class Vertex is provided to you.
3. To check your graph implementation, run the driver code in the template file graph.py posted under Files/Programming Assignments/ PA5 on Canvas.
4. You **may or may not** use the graph.py module in your application. However, you need to complete it to get the full credit.

ADT Graph: Class Vertex

```
class Vertex:
    def __init__(self,key):
        self.id = key
        self.connectedTo = {}
        self.color = 'white'
    def addNeighbor(self,nbr,weight=0):
        self.connectedTo[nbr] = weight
    def __str__(self):
        return str(self.id) + ' connectedTo: ' +
str([x.id for x in
self.connectedTo])
    def getConnections(self):
        return self.connectedTo.keys()
    def getId(self):
        return self.id
    def getWeight(self,nbr):
        return self.connectedTo[nbr]
```

ADT Graph: Class Graph

1. Graph() creates an object Graph with vertices of the class Vertex given previously
2. addVertex(key) adds a new vertex with the key (label) ‘key’ to the graph
3. addEdge(key1, key2) adds an edge between two vertices with the keys (labels) key1 an key2
4. getVertex(key) returns a vertex with the key (label) ‘key’
5. breadth_first_search(key) returns a list of vertices in order of their discovery by the BFS traversal where ‘key’ is the key of the start vertex
6. depth_first_search() returns a list of vertices in order of their discovery by the DFS traversal that starts at the vertex that has the lowest key value
7. the graph vertices should be sorted by their keys in the incremented order (before using DFS and BFS)