CISC106 - Fall 2019
Packing Project

Part 1 - Due Friday Nov 15, 11:59 pm

Low and medium-level functions for placing shapes on grid and scoring.

Part 2 - Due Friday Nov 22, 11:59 pm

High-level problem solving and graphics.

Part 3 - Due Tuesday Dec 3, 11:59 pm

Specification updates and heuristic improvement.

Description

The packing problem is an important problem in the real world. Companies like FedEx, UPS, and container shipping firms all equate efficient packing with money. It is also very interesting to mathemeticians and computer scientists:

`https://en.wikipedia.org/wiki/Packing_problems`

Finding a general and optimal solution is very difficult, but as this project shows, we can solve limited versions of the problem with reasonable efficiency in reasonable time.

You will be part of a coding team. Two thirds of your project grade will be based on the team project grade, which includes execution, coding style, testing, and enhancements. The team grade will be apportioned among team members by the team, using a blind rating system where each team member rates the contributions of their peers.[1]

The final third will be based on project questions given in the final exam.

Work on the project within your team only. Seek help and explanations from your team members, not from members of other teams. In particular, DO NOT communicate with another team or outside person about how they wrote a particular function or test. Feel free to send your code or thoughts to other members of your team, but not outside the team. Violations of this policy will be considered academic dishonesty (see the class web page).

- For this project, you may work with zero, one, or two partners of your choosing. Submit your list of team members in the text box on Canvas for lab 10 if you haven't done so already.
- Write tests for the following functions, then write the functions (You have already written much of this code!). Ten tests per function is probably a good minimum, but you'll need more than that to test some well. I have given you some very simple examples for some functions, but you need to test all functions except grid.print(). There are lots of boundary cases. See the professor if you are unable to think of good test cases.
- Strategy: write tests first, thinking about special cases that might break your code. Then write low-level functions that do not depend on other functions, and run your tests. Slowly work up to higher level functions. (Design goes the other direction, high-level first, but this project has been designed for you.)

```
from cisc106 import *
import time
from tkinter import *
DEBUG = True

# A grid is a 2D list containing ones or zeros. A "shape" is an
# object containing a nested tuple, where the real world shape is
# represented as True values. For example, an "L" shape could be
```

[1] The Professor reserves the right to override team evaluations if s/he doesn't think they accurately represent contributions.

```
# represented as
#
#     ((False, False, True),(True, True, True))
#
# Read a sequence of letters from a string (later from a text
# file). Each letter represents a shape: T, L, or Z (all 2x3), or I
# (1x4). Use functions  to find a place on the grid where
# the next shape in the sequence doesn't overlap any ones. Rotate the
# shape to find the "best" fit. Then modify the grid to show the
# shape in place, and repeat with the next shape in the sequence.

# We will add the graphics and high level functions in the next part.

class Grid:
    """
    Has attributes numRows, numCols, and squares (a 2D list containing True/False).
    """
    def __init__(self,rows,cols,squares):
        self.numRows = rows
        self.numCols = cols
        self.squares = squares if squares else [cols * [False] for r in range(rows)]

    def updateGrid(self, shape):
        """
        Top-level def for placing a single shape on grid. Finds best
        position and rotation for shape, update grid squares (mutates).
        Calls: find_max_score_location
        Returns boolean: fits or not
        """

    def print(self):
        """
        Print the grid, using an asterisk for each true square, underscore for false.
        Use a newline after each row, and no spaces in rows.
        """

class Shape:
    """
    A "shape" is a nested tuple, where the real world shape is
    represented as ones. For example, an "L" shape could be
    represented as

     ((False, False, True),(True, True, True))

    Has attributes x, y, letter, squares (a nested list of type boolean),
        color, num_rotations
    """
    def __init__(self, letter, squares, color):
        self.x = 0 # will be modified later to indicate position
        self.y = 0 # will be modified later to indicate position
```

```
        self.letter = letter
        self.squares = squares
        self.color = color
        self.num_rotations = 0


    def rotate90(self):
        """
        Rotates this shape 90 degrees clockwise (direction
        matters). Mutates squares and num_rotations
        Returns None
        """


def generate_all_locations(grid, shape):
    """
    Takes a single shape in one position and finds all the places it could fit on the
    grid, given its dimensions.
    Returns: a list of row,col tuples
    """

def get_valid_locations(location_list, grid, shape):
    """
    Returns list of locations where shape does not overlap shapes
    already on grid. Assumes that all locations in parameter list are
    potential fits for this shape.
    Calls: fits
    """

def fits(location, grid, shape):
    """
    Returns True if shape placed at location does not overlap shapes
    already on grid.
    location: row,col tuple
    """

def get_max_score(location_list, grid, shape):
    """
    Finds highest scoring location from list, given shape.

    When scores are equal, the lowest row (highest row number), right end (highest
    column) should be preferred.

    Return: nested tuple of (location_tuple,number)
    Calls: get_score
    """

def get_score(location, grid, shape):
    """
    Computes the score for a shape placed at a single location on grid.
```

```
    Scores are positive, higher is better. For now, code the heuristic discussed in class.
    location: row,col tuple
    Returns: number
    """

def find_max_score_location(grid, shape):
    """
    Takes a single shape, finds best position on grid. Tries original
    and three possible 90 degree rotations. Mutates shape for each rotation.

    When scores are equal, the lowest row (highest row number), right end (highest
    column) should be preferred.

    Returns tuple:  (fits numberRotations location)
    fits: bool
    maxScoreRow, maxScoreCol: upper left coordinates of best position for shape on grid
    numberRotations: 0-3 rotations required for best fit.
    Calls: rotate90, generate_all_locations, get_valid_locations, get_max_score
    """

def get_shape(letter):
    """
    Returns the Shape corresponding to the letter parameter: I
    for a line; T for a T; L for an L on its back, foot to right; Z
    for a Z. More may be added.
    """

if DEBUG:
    assertEqual(generate_all_locations(Grid(2,4,[]), get_shape('L')), [(0,0),(0,1)])

    assertEqual(get_max_score([(0,0),(0,1)], Grid(2,4,[]), get_shape('L')),  ((0,1),0))

    assertEqual(get_valid_locations([(0,0),(0,1)], Grid(2,4,[]), get_shape('L')),
                [(0,0),(0,1)])
    b = [[1,0,0],[0,0,0]]
    assertEqual(get_valid_locations([(0,0)], Grid(2,3,b), get_shape('L')), [(0,0)])

    assertEqual(find_max_score_location(Grid(2,4,[]), get_shape('L')), (True, 0, (0,1)))

    assertEqual(find_max_score_location(Grid(3,4,[[0,0,0,0],[0,0,0,0],[0,1,1,1]]),
                 get_shape('L')), (True, 2, (1,0)))

    g = Grid(2,4,[[True,False,False,False],[False,False,False,False]])
    assertEqual(find_max_score_location( g, get_shape('L')), (True, 0, (0,0)))
    g.print()
```

Assignment Submission:

Make sure all names of your programming team are in every file.
Upload all materials to Canvas by the deadline to avoid late point deductions.
Follow the TA's instructions if they vary from this document.