Assignment 3 – A Complete 2D Tetris (10% + 2% Bonus)

Due: 11:59pm, Sunday, 12 July 2020

Q1(7%): Complete your Tetris program (with your Assignment 2 submission) to include the following additional steps and functionality:

- Add the following constants which could be adjusted as needed in the indicated ranges:
  - *M* – scoring factor (range: 1-15).
  - *N* – number of rows required for each *Level* of difficulty (range: 20-40).
  - *S* – speed factor (range: 0.1-1.0).
- When any horizontal row of squares *R* has no hole, i.e. all the squares in *R* are parts of some shapes with colors, *R* is removed and all the rows above *R* move one square down, *Lines = Lines + 1; Score = Score + Level x M*.
- If the number of removed rows in the current *Level* reaches *N, Level = Level + 1*, the falling speed *FS = FS x (1 + Level x S).*
- When a new shape has no space to fall, i.e. existing shapes in "Main area" pile up to near the top, the game terminates.
- If the cursor is inside the falling shape *F* (in PAUSE mode), *F* will be changed to one of the shapes different from *F* and also NOT currently inside "Next shape", *Score = Score - Level x M.* You should use "Point-Inside-Polygon" test algorithm to detect the cursor.
- Create a user-friendly interface so that various parameters could be adjusted, using GUI widgets of your choice (e.g. a slider for *N*), to suit different user groups:
  - Constants *M, N,* and *S* are individually adjustable.
  - The width and height of "Main area" can be adjusted (beyond 10 x 20 squares).
  - The size of the square is adjustable (e.g. enlarged for elderly or visually impaired players).

Note: The above updating functions are very simple (by multiplying a constant). You may introduce more realistic and complex functions and may vary the ranges of *M, N,* and *S* to reflect more interesting playing experiences. Describe your changes, as well as the general implementation method, in the comments on top of the source code.

Q2(3%): As a computer graphics expert, you are approached by an architect with a small contract to draw a spiral staircase that is customizable with different parameters. You then notice that the requested spiral staircase looks very similar to that in Exercise 5.11 of the textbook. Here are the architect's requirements: Java program file is named as Stairs.java. The program should be able to produce a data file that can be viewed using Painter.java, ZBuff.java, or HLines.java in Chapter 6 and look VERY similar to the one in the book. There should be no gap between the stairs and the central cylinder. A sample data file called stairs.dat may be downloaded from eLearning for you to view it in action. You can run the compiled Painter.java, ZBuff.java, or HLines.java and then load stairs.dat.

The Stairs.java program should take 3 command-line arguments for the three customizable parameters. In other words, after compiling your program, you may run it by

java Stair 25 15 stairs.dat

where 25 is the number of stairs, 15 is the rotation degree between every two consecutive stairs, and stairs.dat is the output data file.

Q3(2% Bonus): Fix the bug in CGDemo's Brehsenham Line Drawing Algorithm (i.e. the two Y choices could both be under the ideal line. They instead should be on each side of the ideal line).

Submit your Java program in one file on eLearning before the due time. Late submissions should be submitted to the TA through email rather than through eLearning, and will be penalized with a 2% deduction for every 1-24 hour delay.