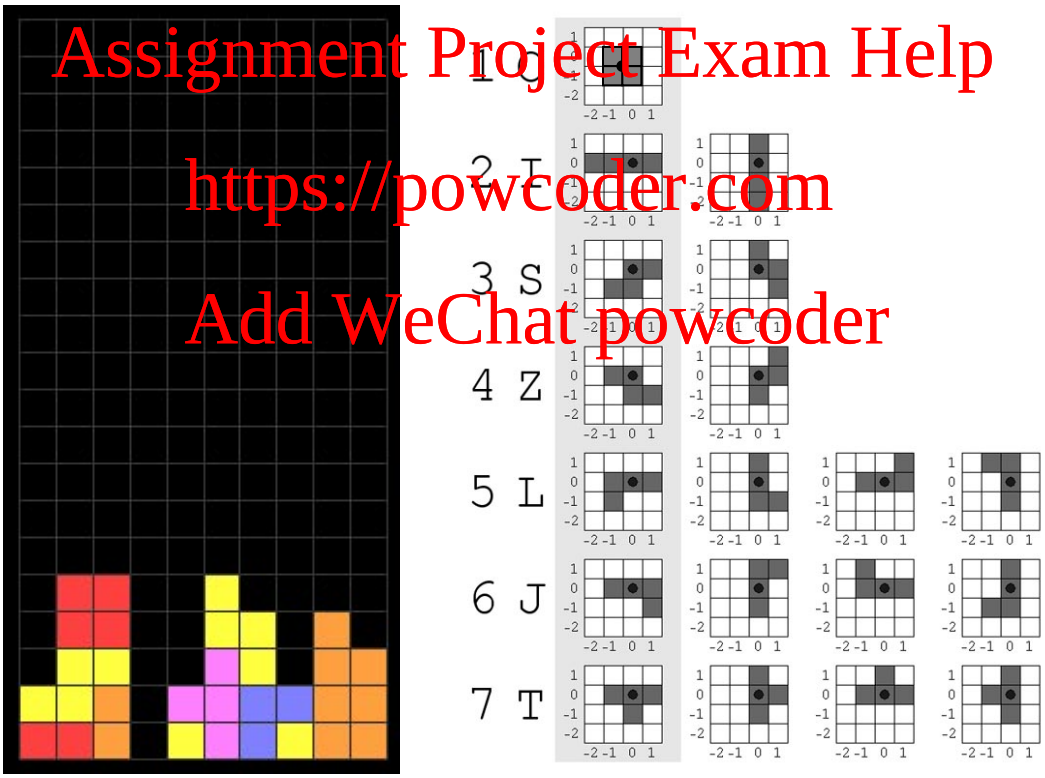
Computing Science CMPT 361 **Fall 2020**
Instructor: Richard (Hao) Zhang **Simon Fraser University**

Assignment #1 (8 marks)

Written parts are **exercises, no submission;** solutions posted progressively by **Nov. 17.**
Programming part due: **Tuesday, Nov. 17, 11:45 p.m. via electronic submission.**

Programming (8 marks): Tetris!

You are to implement a simplified version of the game Tetris as described below. Any visual flare that you wish to add to the appearance of your game will be judged by the grader and may be credited at his discretion. The game window consists of a 20 × 10 square grid of appropriate size, e.g., so that the window will fit in the screen comfortably. There are seven standard Tetris pieces (or *tiles*), as shown below, with pivot of rotation indicated by a black dot. You are advised to complete this problem in several steps.



(a) [3 marks] Tile and grid rendering and tile downward movement

Set up the game window with grid lines and randomly select Tetris tiles with distinguishing colors one at a time and drop them from the top of the game window. The starting position and orientation of the tile is chosen randomly. You can control the speed of the tile movement to suit your game playing. Movement of the tiles'

will be aligned with the grids and at uniform speed. For this step, the tiles can drop straight through the bottom. After one tile disappears, a new tile is dropped.

(b) **[1 marks] Tile stack-up**

In this step, the tiles will stack up on top of each other and the bottom of the game window will offer ground support, as in the Tetris game you are familiar with.

(c) **[3 marks] Key stroke interaction and tile movements**

The four arrow keys will be used to move the tiles. A pressing of the "up" key rotates a tile *counterclockwise* about its pivot, 90° at a time. The "left" and "right" key presses result in lateral movements of a tile, one grid at a time. The "down" key accelerates the downward movement. At no time should you allow a tile piece to collide with any existing Tetris pieces or the border of the game window.

(d) **[1 marks] Additional game logic**

When the bottom row is completely filled, it is removed and the tile stack above it will be moved one row down. Game terminates when a new tile piece cannot be fit within the game window. Press 'q' to quit and 'r' to restart. *Pressing any of the arrow keys should not slow down the downward movement of a tile.*

Note that the above steps build on top of each other, in order. You need not submit individual programs to correspond to these steps. If you can implement all the required parts, a single, complete program is sufficient. **No skeleton code** is provided.

Submission: All source codes and a README file that documents any steps not completed, additional features, and any extra instructions for your TA.

Exercise 1: "Why HD DVD failed?"

The so-called "full HDTV" mode supports a screen resolution of 1920 × 1080. Suppose that the video content is to be displayed at 30 Hz and that we wish to provide 24 bits of color (8 bits per R, G, B). Then how many seconds of video of this type could fit on a single-layer HD DVD, which has a storage capacity of 15 Gigabytes? Does your answer contradict with common-sense? Should the demise of the HD DVD format against its rival Blue-Ray be attributed to this? Offer an explanation.

Exercise 2: Line drawing

Compare the *diamond exit rule* and the *Bresenham's algorithm*. Specially, given two end points (x_1, y_1) and (x_2, y_2) with integer coordinates, will the pixels returned by the two algorithms, except for the two end pixels, be the same in general? Explain your answer.

Exercise 3: Rigid-body transformations

Let us recall that the general form of a 3D transformation matrix M in homogeneous coordinates is

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Assume that the upper 3 by 3 submatrix R of M is *orthonormal*, i.e., $R^T = R^{-1}$.

1. What is the inverse of M? Note that you should not use brute force or a package such as Maple or Matlab to answer this question.
2. Prove that the transformation M preserves both lengths and angles in 3D. Note here that when we talk about the angle between two vectors, the order in which the two vectors are given would be irrelevant.

Exercise 4: gluLookAt()

The GL utility (GLU) library had a function called gluLookAt which has the following specification:

gluLookAt(*ex*, *ey*, *ez*, *rx*, *ry*, *rz*, *ux*, *uy*, *uz*)

Write out the transformation matrix T which transforms points in the world coordinate system (WCS) into the view coordinate system (VCS) specified by gluLookAt(). The VCS has origin at the eye point (*ex*, *ey*, *ez*), and its positive *z*-axis is aligned with the vector $\mathbf{v} = (ex, ey, ez) - (rx, ry, rz)$. Also remember that the up vector (*ux*, *uy*, *uz*) is not necessarily perpendicular to the vector $\mathbf{v}$. *You may leave your final answer as a product of two or more matrices*.

Exercise 5: Clipping with convex polygons

Prove that clipping a convex polygon against another convex polygon will yield at most one convex polygon.

Exercise 6: BSP vs. depth-sort

Show that the back-to-front display order determined by traversing a BSP tree is not necessarily the same as the back-to-front order determined by depth-sort, even when no polygons are split. To receive full mark on this problem, the number of polygons you use for your example must be the smallest possible and you also need to prove that the number of polygons you used is the smallest possible. **Hint: think about an example.**

Exercise 7: Silhouettes

Consider a scene consisting of a set of closed convex objects represented by triangle meshes and a viewpoint, device a simple method which returns the set of all silhouette edges of the objects with respect to the viewpoint, assuming perspective projection. Prove that the set of edges returned will be forming a set of closed loops.

Exercise 8: Radiosity

Explain in what ways the radiosity method covered in class is designed to model global illumination of a scene composed of perfect diffuse reflectors.

Exercise 9: Ray tracing

Explain why the classical ray tracing algorithm, the one covered in class, it best suited to render "glossy" scenes, i.e., scenes that are composed mostly of highly reflective and shinny surfaces? Can you propose a simply modification to that algorithm to also render dull surfaces, i.e., surfaces that are diffuse rather than reflective?