ECE 4122/6122 Hmk #6

(100 pts)

Section	Due Date
89313 - ECE 4122 - A	Nov 22nd, 2019 by 11:59 PM
89314 - ECE 6122 - A	Nov 22nd, 2019 by 11:59 PM
89340 - ECE 6122 - Q	Nov 24th, 2019 by 11:59 PM
89706 - ECE 6122 - QSZ	Nov 24th , 2019 by 11:59 PM

Notes:

You can write, debug and test your code locally on your personal computer. However, the code you submit must compile and run correctly on the PACE-ICE server.

Submitting the Assignment:

See Appendix C.

Grading Rubric

AUTOMATIC GRADING POINT DEDUCTIONS :

Element	Percentage Deduction	Details
Does Not Compile	40%	Code does not compile on PACE-ICE!
Chessboard	8% (up to)	Board drawn incorrectly
Chess pieces	32% (up to)	-1 for each piece drawn incorrectly (size and/or location)
Initial lighting/material	10% (up to)	5 for each light not working correctly (includes turning on and off)
Enable features are working	10% (up to)	Double buffering, depth, smooth modeling, etc…
Key press actions	30% (up to)	Divided equally among key presses excluding turning lights on/off.
Clear Self-Documenting Coding Styles	10% (up to)	This can include incorrect indentation, using unclear variable names, unclear/missing comments, or compiling with warnings. (See Appendix A)

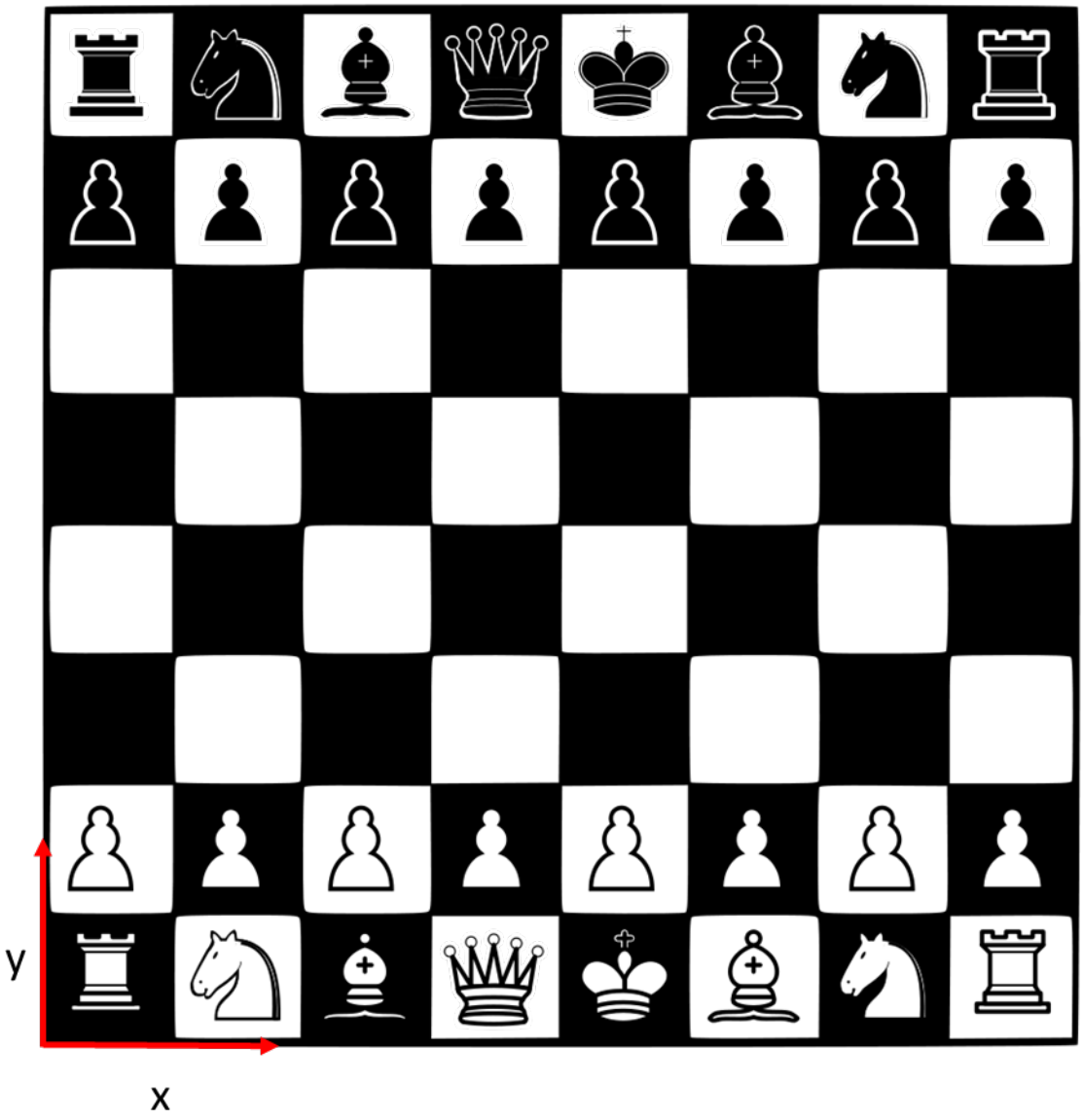
LATE POLICY

Element	Percentage Deduction	Details
Late Deduction Function	score - (20/24)*H	H = number of hours (ceiling function) passed deadline note : Sat/Sun count as one day; therefore H = 0.5*$H_{weekend}$

OpenGL Chess Set

You need to create a 3D chess set model using OpenGL following these rules:

1. Create a chessboard which is a 8 x 8 grid of alternating light and dark squares in the xy plane, as seen below, with each square being 1 m x 1 m. The origin of your model is at the lower left-hand corner. The positive z-axis is up out of the page



2. There are 6 types of chess pieces which are to be represent by 6 different GLUT shapes

   a.  pawn shall be represented using a glutSolidSphere

   b.  rook shall be represented using a glutSolidCube

   c.  knight shall be represented using a glutSolidTeapot

   d.  bishop shall be represented using a glutSolidCone

   e.  queen shall be represented using a glutSolidTetrahedron

   f.  king shall be represented using a glutSolidOctahedron

3. Each piece (light and dark) shall be placed in the correct starting location as shown in the figure for part 1.
4. Each piece shall be sized and/or scaled so that it has a bounding box of 0.75 m width, 0.75 m depth, and 1.0 m height.
5. Each piece needs to be centered in the correct square and sitting flush with the surface of the chessboard
6. The light colored pieces should have a RGB color of RGB (255, 255, 240).
7. The dark colored pieces should have a RGB color of RGB (150, 75, 0).
8. Enable ambient lighting (GL_LIGHT0) and one diffuse light (GL_LIGHT1). GL_LIGHT1 located at (-5, -5, 8)
   a. GLfloat light0_ambient[] = { 0.2, 0.2, 0.2, 1.0 };
   b. GLfloat light0_specular[] = { 0.8, 0.8, 0.8, 1.0 };

9. Setup the following material properties for the pieces
   GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
   GLfloat mat_shininess[] = { 50.0 };

10. Enable depth testing.
11. Use a smooth shading model.
12. The viewport (window size) shall be width of 600 and height of 600.
13. The viewport can be resized and the view of the chess set remain correct.

14. Use double buffering.
15. Use a perspective viewing volume, with the eye located at (4, -5, 10) with a large enough field-of-view to see the whole chess set. The viewing volume should be centered on the center of the chessboard (4, 4, 0).
16. Key press commands (the center of the view volume is always maintained at (4, 4, 0)
    a. Pressing 'r' or 'R' key rotates the chess set by 10 degrees around the z-axis at its center point each time.
    b. Pressing 'd' or 'D' key moves the eye location down z –axis 0.25 m each time.
    c. Pressing 'u' or 'U' key moves the eye location up z-axis 0.25 m each time.
    d. Pressing '0' (zero) key toggles GL_LIGHT0 on and off (enable/disable)
    e. Pressing '1' key toggles GL_LIGHT1 on and off (enable/disable)
    f. **ECE6122 students** also do the extra following additional commands.
        i. Presses a 'k' or "K" causing a random knight to move only one position forward or one position backwards, assuming the move is allowed. If the move is not allowed then a new random knight is chosen until one can move. The piece can disappear and reappear.
        ii. Presses a 'p' or "P" causing a random pawn to move one position forward or backwards the correct amount, assuming the move is allowed. If the move is not allowed then a new random pawn is chosen until one can move. The piece can disappear and reappear.

Extra Credit:

Pressing the 'E' or 'e' key toggles between normal mode and enhanced mode that makes some of the pieces look more realistic (discretion of TAs if it is good enough)

- Pawn (+1), Rook (+1), Bishop (+1)
- Knight (+4), King (+2), Queen (+2)

Appendix A: Coding Standards

Indentation:

When using *if/for/while* statements, make sure you indent 4 spaces for the content inside those. Also make sure that you use spaces to make the code more readable.
For example:

```
for (int i; i < 10; i++)
{
    j = j + i;
}
```

If you have nested statements, you should use multiple indentions. Each { should be on its own line (like the *for* loop) If you have *else* or *else if* statements after your *if* statement, they should be on their own line.

```
for (int i; i < 10; i++)
{
   if (i < 5)
   {
       counter++;
       k -= i;
   }
   else
   {
       k +=1;
   }
   j += i;
}
```

Camel Case:

This naming convention has the first letter of the variable be lower case, and the first letter in each new word be capitalized (e.g. firstSecondThird). This applies for functions and member functions as well! The main exception to this is class names, where the first letter should also be capitalized.

Variable and Function Names:

Your variable and function names should be clear about what that variable or function is. Do not use one letter variables, but use abbreviations when it is appropriate (for example: “imag" instead of “imaginary”). The more descriptive your variable and function names are, the more readable your code will be. This is the idea behind self-documenting code.

File Headers:

Every file should have the following header at the top

```
/*
Author: <your name>
Class: ECE4122 or ECE6122
Last Date Modified: <date>

Description:

What is the purpose of this file?

*/
```

Code Comments:

1. Every function must have a comment section describing the purpose of the function, the input and output parameters, the return value (if any).
2. Every class must have a comment section to describe the purpose of the class.
3. Comments need to be placed inside of functions/loops to assist in the understanding of the flow of the code.

Appendix B: Submitting Assignment

Step-By-Step Submitting a Tar.gz

PLEASE NOTE: All the following instructions are run on PACE. If you are struggling with any of the steps, please come see the TAs during office hours. Better to start and test early than wait until the last minute.

Below is a step-by-step tutorial on how to create the .tgz file for Homework 6 for ECE 4122/6122. Additionally, if you do not use a header file for a solution, you do not need to submit a header file. Please adjust these instructions accordingly. The tarball should contain a folder with your buzzid containing a subfolder for the problem with the corresponding .cpp and .h files.

Please make sure that the problem is in a subfolder with the naming conventions:
<FirstName_LastName>_Hmk6

Create a subdirectory named *buzzid* in the current working directory by executing the following command in the shell:
$ mkdir *buzzed*

Create a text file named **manifest.**

Please make sure that the **manifest** file is a plain text file and not a rich text file. Microsoft word will generate a rich text file. The easiest method to create a plain text file is to use a command line editor such as vi, vim, or nano.

(If you want a tutorial, these can be useful: vi: http://heather.cs.ucdavis.edu/~matloff/UnixAndC/Editors/ViIntro.html, vim: https://openvim.com/, nano: https://www.howtogeek.com/howto/42980/the-beginners-guide-to-nano-the-linux-command-line-text-editor/)

Populate the **manifest** file with the following:

buzzid/<FirstName_LastName>_Hmk6/*.cpp
buzzid/<FirstName_LastName>_Hmk6/*.h

(***PLEASE NOTE***: this is subject to change with depending on your class section. The wildcard (*) character will grab all the .cpp and .h files you generated for each problem.)

Now you need to construct the correct file structure for the submission. This means that you need to put all the homework files into your *buzzid* folder. Execute the following lines in the shell:

$ cp -a <FirstName_LastName>_* *buzzid*

Many people have had issue with this step. You need to make sure the naming conventions are consistent to avoid potential problems.

It is now time to tarball your submission. If all the other steps have gone smoothly execute the following command:

$ tar -zcvf *buzzid*-hmk6.tgz $(cat manifest)

You can now check the new tgz file just generated with the following command:

$ tar -ztvf *buzzid*-hmk6.tgz