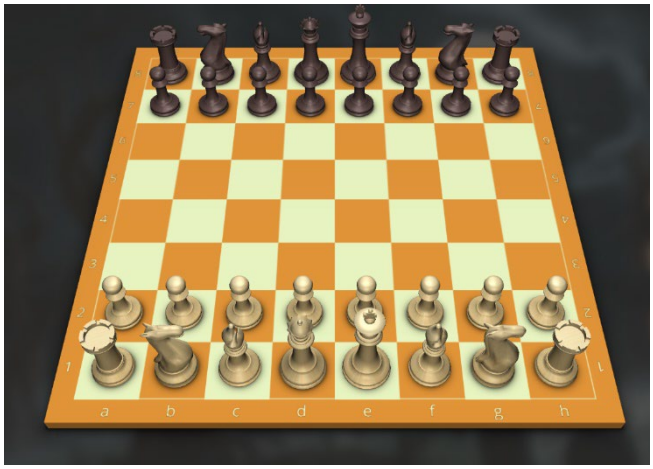
ECE 4122/6122 Final Project: 3D Chess Game
Custom Classes & OpenGL

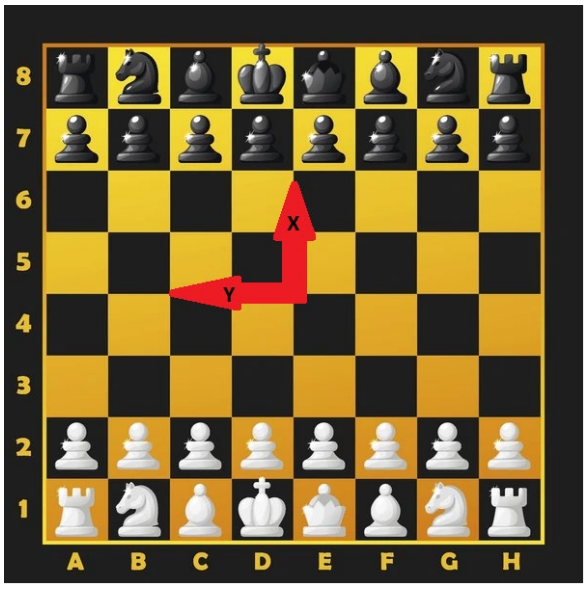
(300 pts)

Due: Tuesday Dec 3rd, 2024 by 11:59 PM



Write a C++ application that uses a custom class(es) with OpenGL, a third-party library (i.e. ASSIMP), and a third-party chess engine (i.e. Komodo, see attached file) to create a fully operational 3D chess game. You can use the OpenGL lab as a starting point. The program should load and render the 3D chess pieces and chessboard in the 3D window. The user enters moves through the command window and your program needs to send the moves to a chess engine and capture the response moves. Your program needs to move the pieces around the board as the moves are entered and received from the chess engine. The chess pieces should slide across the chessboard (~2-3 seconds per move) except for the knight chess piece which should move into the air and over the other pieces to land in the correct location. When pieces are destroyed, they are removed from the game. The user enters move using the UCI (Universal Chess Interface) format (i.e. e2e4).

The user can move the camera location around the board using spherical coordinates. The chess game needs to be setup so that the z-axis point upward out of the board and the xy-axes are aligned as shown:



The user moves the camera using the command: *camera* Θ Φ *R* where Θ & Φ are in degrees and R is unitless. The camera always points towards the origin. The user can move the light source around using the command: *light* Θ Φ *R.*

The following ranges must be enforced.

Θ ranges from 10 to 80 degrees
Φ ranges from 0 to 360 degrees
R > 0.0

The user can change the light power using the command: *power 100.0*
The game ends either when someone wins or the user enters the command: *quit*.
Pressing escape key ends the application.

Below is a sample console input

- Please enter a command: move e2e4
- Please enter a command: camera 10.0 45.0 5.0
- Please enter a command: camera 10.0 45.0 -5.0
- Invalid command or move!!
- Please enter a command: move ee2e4
- Invalid command or move!!
- Please enter a command: move e2e3
- Checkmate you win!!
- Checkmate you lose!!

- Please enter a command: quit
- Thanks for playing!!

1. (50 pts) The chess pieces and board are position so the Z axis points upward out of the board and is centered around the origin.
2. (50 pts) The camera command moves the camera to the correct location and illegal commands are captured.
3. (50 pts) The light position and power are controlled correctly by the commands and illegal commands are captured..
4. (50 pts) The chess pieces move correctly and illegal chess moves are detected. Checkmate is detected and reported.
5. (100 pts) The interface to the external chess engine works correctly and reasonable responsive moves are received. The interface to the external engine is wrapped in a class called ECE_ChessEngine and needs to have at least the following functions:
    - bool InitializeEngine()
    - bool sendMove(const std::string& strMove);
    - bool getResponseMove(std::string& strMove);

Extra Credit (up to 20 pts)

When pieces are destroyed, they are placed along the sides of the chessboard instead of just disappearing.

Turn-in Instructions:

Two methods:

1.
    a. Upload a video of your application running and demonstrate the requirements above. You can include multiple movie files. Zip your movie file(s) into a file called movieFiles.zip
    b. Put all the code files you created into a zip file called ***FinalProject.zip*** and upload to canvas.
2.
    a. The TAs will build and run your code on pace-ice.
    b. Put your code in a folder called FinalProject. You need to include a CMakeLists.txt file that will successfully build your code on pace-ice. Include everything that is needed inside of your folder. Zip this folder in a file called ***FinalProject.zip*** and upload to canvas.

Grading Rubric

If a student's program runs correctly and produces the desired output, the student has the potential to get a 100 on his or her homework; however, TA's will **randomly** look through this set of "perfect-output" programs to look for other elements of meeting the lab requirements. The table below shows typical deductions that could occur.

AUTOMATIC GRADING POINT DEDUCTIONS PER PROBLEM:

Element	Percentage Deduction	Details
Does Not Compile	30%	Code does not compile on PACE-ICE!
Does Not Match Output	Up to 100%	The code compiles but does not produce the correct outputs. See point values above.
Clear Self-Documenting Coding Styles	10%-25%	This can include incorrect indentation, using unclear variable names, unclear/missing comments, or compiling with warnings. (See Appendix A)

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 (section)
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.