MARIST College **Computer Graphics**
Fall 2013 **CMSC446:MSCS555**

Team Project II: Chess Game Visualizer

Demonstration Date: Wednesday, December 11 @ 1530

Submission Format: Electronic via iLearn

Notes: Each team shall comprise two students. Only one submission is required for each team.

Summary

The goal of this application is to implement a Chess visualizer that presents a realistic rendering of a chess game in action.

Requirements

The application must communicate with a remote server to receive each move in the game and subsequently animate the Board and Pieces appropriately. The details of the communication protocol and message format will be given later.

You should plan to include detailed models for each object in your visualization – at minimum the board and two complete sets of game pieces, but possibly including additional visual elements.

Each change in the state of the chess game should be accompanied by an appropriate smooth animation – the precise nature of this animation is up to you. It may be a simple sliding or floating of a chess piece from one square to another, or it may be a more complex animation sequence if time permits. In any event, you should be sure to avoid having one piece move *through* another – that is, the pieces should act as solid objects.

User controls should permit opening/closing a connection to the server, changing the camera properties, selecting a *theme* for the board and pieces, and quitting the application. Camera changes should be smooth (not instant), and at least two visual themes should be provided.

You must have detailed shading of your scene, with at least two distinct types of light sources. The precise details of each light (type, location, color, etc.) is left to your artistic discretion. Basic smooth shading should be performed in the fragment stage using the typical Blinn-Phong method. If you wish to implement some form of artistic shader (e.g., Toon) instead, you are free to do so, but please clear it with your instructor first. You must use at least two texture images applied to at least two objects – one for each of the "themes" mentioned above.

All objects should be managed using a scene graph via the THREE.JS library.

In addition to these basic modeling/shading/animation requirements, you must also incorporate a variety of other techniques, including but not necessarily limited to – secondary shaders (e.g., bump), additional textures, collision detection/physics, complex animations, shadow mapping, particle effects, etc. The highest grades are not attainable without some combination of these additional features.

Of course, coding style, modularity, and other best practices also factor into the project grade.

Documentation

All software must include extensive and appropriate embedded documentation (i.e., regular and *Javadoc* or similar comments). A brief but complete user guide must also accompany your project.

Each class and public member must have Javadoc or similar style comments; each global function or variable must have regular comments.

Every control structure or block of code in excess of 4 lines must have at least a one-line comment associated with it.

The User Guide must cover all methods of input, any rules of operation (input restrictions, input sequences, game logic, etc.), file formats, and how to interpret the output. The document must be no more than 4 pages in length.

Grading

A breakdown of the individual components of the application and their weights is given below.

◆ Modeling & Materials	15%
◆ Transformations & Animation	20%
◆ Cameras & Viewing	10%
◆ Lights & Shaders	10%
◆ Texture Mapping	10%
◆ User Interaction	10%
◆ Query Server & Handle Response	10%
◆ Overall Design & Quality	10%
◆ Additional/Advanced Features	5%

Running Your Application

When using Chrome, you may need to launch the browser with the `—disable-web-security` flag if you are simply loading the files from your local disk. You can run a local Web server if you wish in order to avoid this problem.

For our final demonstrations, we will run all the projects from a central Web server located at `10.11.18.65` on the Marist network. Each team has a single user account via which they can upload all their project files to the server using SFTP.

Each team's login name and the default password – which you should change right away – are provided under Resources on iLearn. If you should have any trouble connecting to the server, please let me know immediately.

Each account has a `www` directory for the purpose of hosting your THREE.JS applications. Please upload your files into this `www` directory.

Client-Server Communications

Your program should use a text input field to obtain a GAMEID number from the user. You can query the Chess Server's REST API via AJAX in order to obtain the game history. I have reproduced one of the most famous chess matches in history, Kasparov vs Topalov 1999, in order to provide you with a sample game against which you can test your graphical application.

Given an HTTP GET request with the following URL: `http://www.bencarle.com/chess/cg/340`

you will receive a response in the form of a JSON string, such as

```
{
    "blacktime": 108.887688,
    "gameover": true,
    "lastmovenumber": 87,
    "moves": [
        "Pe2e4", "Pd7d6", "Pd2d4", "Ng8f6", "Nb1c3", "Pg7g6", "Bc1e3", "Bf8g7", "Qd1d2", "Pc7c6",
        "Pf2f3", "Pb7b5", "Ng1e2", "Nb8d7", "Be3h6", "Bg7h6", "Qd2h6", "Bc8b7", "Pa2a3", "Pe7e5",
        "Ke1c1", "Qd8e7", "Kc1b1", "Pa7a6", "Ne2c1", "Ke8c8", "Nc1b3", "Pe5d4", "Rd1d4", "Pc6c5",
        "Rd4d1", "Nd7b6", "Pg2g3", "Kc8b8", "Nb3a5", "Bb7a8", "Bf1h3", "Pd6d5", "Qh6f4", "Kb8a7",
        "Rh1e1", "Pd5d4", "Nc3d5", "Nb6d5", "Pe4d5", "Qe7d6", "Rd1d4", "Pc5d4", "Re1e7", "Ka7b6",
        "Qf4d4", "Kb6a5", "Pb2b4", "Ka5a4", "Qd4c3", "Qd6d5", "Re7a7", "Ba8b7", "Ra7b7", "Qd5c4",
        "Qc3f6", "Ka4a3", "Qf6a6", "Ka3b4", "Pc2c3", "Kb4c3", "Qa6a1", "Kc3d2", "Qa1b2", "Kd2d1",
        "Bh3f1", "Rd8d2", "Rb7d7", "Rd2d7", "Bf1c4", "Pb5c4", "Qb2h8", "Rd7d3", "Qh8a8", "Pc4c3",
        "Qa8a4", "Kd1e1", "Pf3f4", "Pf7f5", "Kb1c1", "Rd3d2", "Qa4a7"
    ],
    "whitesturn": false,
    "whitetime": 120.588292
}
```

Your application must read and parse this response and be able to reproduce the entire sequence of moves in the game. You must also display a countdown of the active player's remaining time.

It will be necessary to poll the game server at some short, regular interval in order to determine whether any new moves have been made. Your application should update live while a game is being played, and it should also support the ability to replay an entire game from its beginning.

Note that some moves have consequences that are not encoded in the response. For example:

- Black's 8th move `Bg7h6` *captures* White's bishop. Your application should detect this and remove the captured bishop from the board. (You may choose to display it off to the side.)
- White's 11th move `Ke1c1` is a special *castling* move that allows two pieces to move at once. In this case, the King castles *queenside* by moving two spaces while the Queen's Rook jumps over the King. Note that Black's 13th move `Ke8c8` also a castling move.
- Two other special Chess moves that are not demonstrated in this sample game include *promotion* and *en passant*. Your program should be prepared to handle these moves.

On AJAX, REST, & JSON

AJAX is a relatively simple technique that can be implemented from scratch or provide via framework such as *jquery* or *mootools*.

http://eloquentjavascript.net/chapter14.html

http://api.jquery.com/jQuery.ajax/

http://mootools.net/docs/core/Request/Request

We will be using AJAX to send a simple HTTP request to the Chess Server, which provides a RESTful API for querying the state of a particular game. Just make sure that you send your request using the GET method of HTTP and that you parse the response as plain text (instead of HTML or XML).

The response will be a JSON string, which can be parsed converted into a JavaScript object easily enough via the `JSON.parse()` function. Once you have the resulting program object, you can operate on it in the same way you would any other JavaScript object. The list of game moves is simply a JavaScript Array, so you can iterate over all the moves in the usual way.

http://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/parse

Proposed Timeline

Oct 23 – Form/select project teams

Oct 30 – Determine work plan and delegate tasks

Nov 6 – Begin implementation with board and hard-code pieces

Nov 13 – Begin testing client-server communication and parsing response

Nov 20 – Animate the movement of pieces/lighting/view

Dec 4 – Add texture mapping and any, additional features

Dec 11 – Tested, debugged, ready for demo