Chessgames Deliverable

March 15, 2015

**Prepared For:**

Daniel Freeman

Todd Breedlove

**Prepared By:**

Stewart Taylor

Jacob Neal

Aaron Costner

Lewis Sanchez

Table of Contents

[Introduction 2](#_Toc414212915)

[Backlog Items Covered So Far 2](#_Toc414212916)

[Business Rules 3](#_Toc414212917)

[ER Diagram 5](#_Toc414212918)

[SQL Script for Chessgames Database 6](#_Toc414212919)

[Use Cases 7](#_Toc414212920)

[Sequence Diagrams 16](#_Toc414212921)

[User selects either login or register button 16](#_Toc414212922)

[Login the user if the user info is correct 17](#_Toc414212923)

[Register the user if the user does not exist and the username, password and email pass validation 18](#_Toc414212924)

[User selects chess game pool 19](#_Toc414212925)

[User transmits move to other user 20](#_Toc414212926)

[User connects to dedicated server 21](#_Toc414212927)

[Class Diagrams 22](#_Toc414212928)

[CG\_Server 23](#_Toc414212929)

[CG\_PlayerThread 24](#_Toc414212930)

[CG\_Match 25](#_Toc414212931)

[C++ Code 26](#_Toc414212932)

[Main.cpp 26](#_Toc414212933)

[CG\_user.h 28](#_Toc414212934)

[CG\_user.cpp 29](#_Toc414212935)

[CG\_dbManager.h 31](#_Toc414212936)

[CG\_dbManager.cpp 32](#_Toc414212937)

[CG\_board.h 36](#_Toc414212938)

[CG\_board.cpp 37](#_Toc414212939)

[CG\_history.h 46](#_Toc414212940)

[CG\_history.cpp 47](#_Toc414212941)

[CG\_square.h 49](#_Toc414212942)

[CG\_square.cpp 50](#_Toc414212943)

[CG\_piece.h 53](#_Toc414212944)

[CG\_piece.cpp 54](#_Toc414212945)

[CG\_pawn.h 55](#_Toc414212946)

[CG\_pawn.cpp 56](#_Toc414212947)

[CG\_rook.h 57](#_Toc414212948)

[CG\_rook.cpp 58](#_Toc414212949)

[CG\_bishop.h 59](#_Toc414212950)

[CG\_bishop.cpp 60](#_Toc414212951)

[CG\_knight.h 61](#_Toc414212952)

[CG\_knight.cpp 62](#_Toc414212953)

[CG\_queen.h 63](#_Toc414212954)

[CG\_queen.cpp 64](#_Toc414212955)

[CG\_king.h 65](#_Toc414212956)

[CG\_king.cpp 66](#_Toc414212957)

[QML Code 67](#_Toc414212958)

[Main.qml 67](#_Toc414212959)

[CG\_definitions.js 71](#_Toc414212960)

[CG\_login.qml 71](#_Toc414212961)

[CG\_login.js 79](#_Toc414212962)

[CG\_lobby.qml 80](#_Toc414212963)

[CG\_settings.qml 85](#_Toc414212964)

[CG\_loading.qml 90](#_Toc414212965)

[CG\_game.qml 93](#_Toc414212966)

[CG\_game.js 100](#_Toc414212967)

[CG\_board.qml 102](#_Toc414212968)

[CG\_board.js 108](#_Toc414212969)

[CG\_piece.qml 114](#_Toc414212970)

[CG\_banner.qml 116](#_Toc414212971)

[CG\_clock.qml 119](#_Toc414212972)

[CG\_image.qml 121](#_Toc414212973)

[Screenshots 122](#_Toc414212974)

[Server 122](#_Toc414212975)

[Start state 123](#_Toc414212976)

[Username Validation 124](#_Toc414212977)

[Password Validation 125](#_Toc414212978)

[Email Validation 126](#_Toc414212979)

[Lobby 127](#_Toc414212980)

[Settings 128](#_Toc414212981)

[Loading 129](#_Toc414212982)

[White player makes move 130](#_Toc414212983)

[Black player makes move 131](#_Toc414212984)

[QML Scalability 132](#_Toc414212985)

[Conclusion 133](#_Toc414212986)

# **Introduction**

This deliverable lists all design documents, code and screenshots associated with this project. The table of contents lists the order in which these documents were produced. For example -- our group went from the requirements, to the use cases, to the sequence diagrams, to the class diagrams and finally to the code itself.

# Backlog Items Covered So Far

* We worked on and mostly completed our database design with an Entity Relationship Diagram, SQL script for creating the database which we later converted into SQLite and sample inserts.
* Connected user to login system and deployed a “Hello World!” application to and Android device.
* Set up the game lobby and the server. This had a foundation for different game pools (1 minute, 5 minute, or 30 minute games).
* Converted over to QML for our presentation layer.
* Created a game window with a board being displayed and the basic layout of the board itself, with piece movements and sounds.
* Created a loading screen to transition into game.
* Validated piece movements, added piece dragging / zooming, cleaned up layout.
* Implemented chess clocks.
* Created a history class to review moves at the end of a game.
* Created a settings page currently to change country flags in the game window.

# Business Rules

General Assertions:

* A game can have two, and only two, unique users.
* A user can have many games they have played.
* Many users can have many different roles.

User Assertions:

* A flag will be kept for each user to see if a user is banned.
* ELO rating is an attribute of user.
  + ELO is calculated using the number of games played by a user, how many of those games won by the user, the total ELO rating of every user played, and the user’s current ELO.
* Username is an attribute of user.
  + A user must have a username that must be unique to all other users and follow the language rules for a valid username.
* Password is an attribute of user.
  + A user must have a password.
* Join Date is an attribute of user.

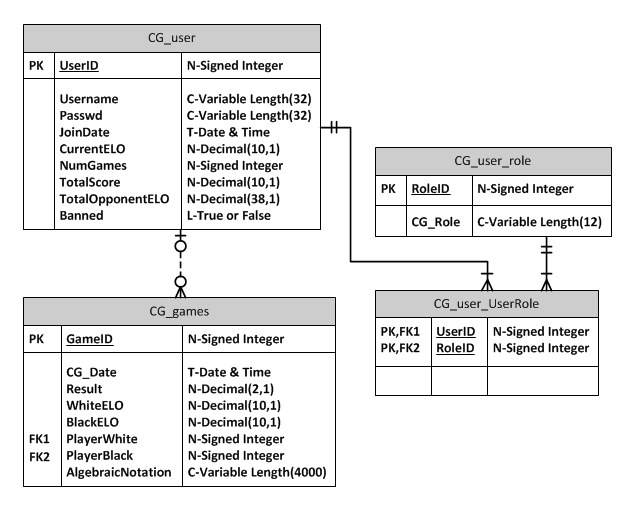
Role Assertions:

* Role Name is an attribute of role. Example Role Names are:
  + Standard (the basic user)
  + Premium (those who have invested more into chessgames.com)
  + Admin
* (In the future, we’d like to expand on these roles and apply them to a user like decorators.)

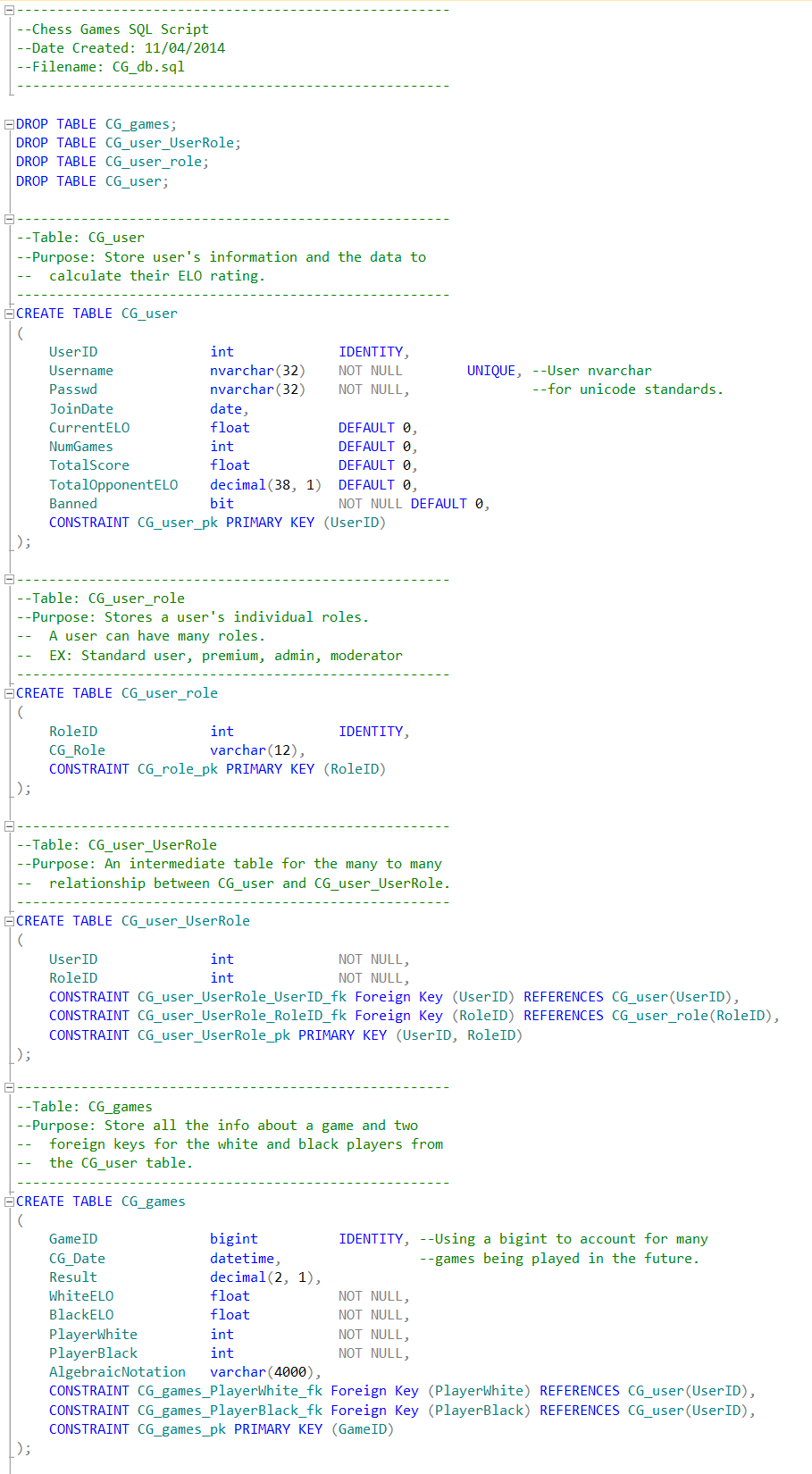
Game Assertions:

* Game Date is an attribute of game.
* Result is an attribute of game.
  + If a game is started between two users, then the game must have a result.
* PlayerWhite and PlayerBlack are attributes of game.
  + PlayerWhite and PlayerBlack must not be null and must not be the same.
  + The ELO ratings of both PlayerWhite and PlayerBlack will be attributes of a game to represent their ELO at the time the game was played.
* AlgebraicNotation is an attribute of game.
  + (A way to describe chess moves made in a game.)

# ER Diagram



# SQL Script for Chessgames Database



# Use Cases

Below is our only use case for our sprint. The outline follows the same format as the use case outline on the SharePoint site. Some parts of SharePoint outline were omitted because they are not applicable to our project (i.e. Timing execution).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **General Information** | | | | |
| Use Case ID Number : UC001  Subject Area : Chessgames Login  Description : Customer uses login to get into game account | | | | Responsible Analyst : Chessgames |
| **Requirements/Feature Trace** | | | | |
| REQ# | Requirements Name and / or Short Description | | | |
| 001 | Login allows customers to login to Chessgames account to play against other users. | | | |
| **Revision History** | | | | |
| **Author** | | **Date** | **Comments** | |
| Lewis Sanchez | | 11/15/14 | First draft | |
| Jacob Neal | | 11/18/14 | Updated | |
| Aaron Costner | | 12/03/14 | Added Exceptions to accommodate invalid passwords and email addresses for logging in. | |
| **Insertion Points in other Use Cases** | | | | |
| Use Case Name | | **Use Case Number** | **Step Inserted After** | |
| **N/A** | |  |  | |
| **Actors** | | | | |
| **Actor Name** | | **P/S** | **Brief Description** | |
| Chessgames user | | P | Chessgames user can login to their account. | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Pre-Conditions** | | | | | | | | | | | | | | |
| # | | | Description | | | | | | | | | | | |
|  | | | Chessgames displays user account login page | | | | | | | | | | | |
| **Start Stimulus** | | | | | | | | | | | | | | |
| Chessgames user opens the application. | | | | | | | | | | | | | | |
| **Use Case Main Course Steps** | | | | | | | | | | | | | | |
| **Number** | | | | **Description** | | | | | **Adds/Alt UC Name/Number** | | | **Bus Rule(s)#** | | |
| 01 | | | | User inputs username and password | | | | |  | | |  | | |
| 02 | | | | User selects either the “Login” or “Register” button. | | | | |  | | |  | | |
| 03 | | | | Validate the username and password fields. | | | | |  | | |  | | |
| **Exception Conditions** | | | | | | | | | | | | | | |
| **Exception Situations** | | | | | | **Action(s) on Exception** | | | | | | | | **Adds/Alt Use Case #** |
| Invalid Username | | | | | | Chessgames app displays message about the username not existing and returns to the login screen. | | | | | | | |  |
| Invalid password | | | | | | Chessgames app displays message about the password not being correct and returns to the login screen. | | | | | | | |  |
| Invalid length | | | | | | Chessgames app displays whether or not the username is too short or too long and returns to login screen. | | | | | | | |  |
| Invalid characters | | | | | | Chessgames app displays whether or not there are invalid characters in the username and returns to login screen. | | | | | | | |  |
| Invalid periods | | | | | | Chessgames app displays whether or not they have a period in the beginning of their username, or the end of the username and returns to login screen. | | | | | | | |  |
| Check for website in username | | | | | | Chessgames app displays an error message for invalid username and returns to login screen. | | | | | | | |  |
| Check password length | | | | | | Chessgames app displays whether or not the pass is too short or too long and returns to login screen. | | | | | | | |  |
| Check Required Characters in Password | | | | | | Chessgames app displays an error message if the password does not contain necessary characters and returns to login screen. | | | | | | | |  |
| Check for valid looking email address | | | | | | Chessgames app displays whether or not the email address inputted takes on the appearance of a valid email address and returns to login screen. | | | | | | | |  |
| **Post-Conditions** | | | | | | | | | | | | | | |
| **#** | | **Description** | | | | | | | | | | | | |
| 1 | | User is logged into account | | | | | | | | | | | | |
| 2 | | Displays a welcome message saying, “Hello “ and the username. | | | | | | | | | | | | |
| 3 | | Chessgames app returns to login screen | | | | | | | | | | | | |
| **Candidate Objects** | | | | | | | | | | | | | | |
| **Class/Object Name** | | | | | **Descriptions** | | | | | | **Possible**  **attributes** | | | |
| CG\_user | | | | | Chessgames user | | | | | | Name, Elo, Country, etc | | | |
|  | | | | |  | | | | | |  | | | |
| **Assumptions** | | | | | | | | | | | | | | |
| **#** | **Assumption** | | | | | | **Date**  **Raised** | **Raised**  **By** | | **Date**  **Verified** | | | **Verified By** | |
| 1 | Customers only speak English. | | | | | | 11/18/14 | Lewis Sanchez | |  | | |  | |
|  |  | | | | | |  |  | |  | | |  | |

|  |  |
| --- | --- |
| **General Information** | |
| Use Case ID Number : UC002  Subject Area : Navigating Application Lobby  Description : Selecting a queue for a timed game of chess. | Responsible Analyst : Todd Breedlove |

|  |  |
| --- | --- |
| **Requirements/Feature Trace** | |
| REQ# | Requirements Name and / or Short Description |
| 5ai | Queues divided into 1 minute, 5 minute, and 30 minute games |
|  |  |
|  |  |
|  |  |

|  |  |  |
| --- | --- | --- |
| **Actors** | | |
| **Actor Name** | **P/S** | **Brief Description** |
| User | P | Someone who wishes to play a 1 minute, 5 minute, or 30 minute game of chess. |
| Server | S | Places a user in a pool and allocates users into a game of chess based on their Elo rating. |
|  |  |  |

|  |  |
| --- | --- |
| **Pre-Conditions** | |
| # | Description |
| 1 | Needs to be logged into the application. |

|  |
| --- |
| **Start Stimulus** |
| Successfully logging in with trigger this use case. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Use Case Main Course Steps** | | | |
| **Number** | **Description** | **Adds/Alt UC Name/Number** | **Bus Rule(s)#** |
| 01 | User decides how long of a chess game to play. |  |  |
| 02 | Server places User in a queue for that kind of chess game. |  |  |
| 03 | User will be idle until Server successfully find another User in the queue within 200 points of the User’s Elo rating, or if the user decides to exit early. |  |  |
| 04 | If User is put into a match with another User, they will play a game of chess with that time control. Otherwise if the User left early, User is brought back to the lobby. |  |  |
| 05 | Server randomly decides which User goes first if a match is initiated. |  |  |

|  |  |  |
| --- | --- | --- |
| **Exception Conditions** | | |
| **Exception Situations** | **Action(s) on Exception** | **Adds/Alt Use Case #** |
| User loses connection during idle time | User is removed from queue, logged out, and brought back to log in screen. |  |
| User loses connection in the lobby | User is logged out and returned to log in screen. |  |

|  |  |
| --- | --- |
| **Post-Conditions** | |
| **#** | **Description** |
| 1 | If successfully put in a match, Users will be put together in a match of chess with the selected time control. |
| 2 | If the User exits early, that User will be returned to the lobby screen. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Assumptions** | | | | | |
| **#** | **Assumption** | **Date**  **Raised** | **Raised**  **By** | **Date**  **Verified** | **Verified By** |
| 1 | That users can speak and read English | 12/13/14 | Aaron Costner |  |  |
| 2 | That there will be more than 1 player in each time control pool/queue | 12/13/14 | Stewart Taylor |  |  |

|  |  |
| --- | --- |
| **General Information** | |
| Use Case ID Number : UC003  Subject Area : Playing Chess  Description : Sending chess moves via text | Responsible Analyst : Todd Breedlove |

|  |  |
| --- | --- |
| **Requirements/Feature Trace** | |
| REQ# | Requirements Name and / or Short Description |
| 12 | Allow user to play chess against another player on application. |
|  |  |
|  |  |
|  |  |

|  |  |  |
| --- | --- | --- |
| **Actors** | | |
| **Actor Name** | **P/S** | **Brief Description** |
| White Player | P | This User sends messages as the white player. |
| Black Player | P | This User sends messages as the black player. |
| Server | S | Server gets the information from a User who makes a move and sends it to the other User. |

|  |  |
| --- | --- |
| **Pre-Conditions** | |
| # | Description |
| 1 | Both Users need to have been successfully put into a match with one another for a selected time control. |

|  |
| --- |
| **Start Stimulus** |
| That both Users have been put into a match with one another. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Use Case Main Course Steps** | | | |
| **Number** | **Description** | **Adds/Alt UC Name/Number** | **Bus Rule(s)#** |
| 01 | Users are notified of their color, who they are matched with, and that a match is being set up. |  |  |
| 02 | White Player types in a move and sends it to the Server. |  |  |
| 03 | Server will transmit White Player’s move and send it to the Black Player. |  |  |
| 04 | Black Player types in a move and sends it to the Server. |  |  |
| 05 | Server will transmit Black Player’s move and send it to the White Player. |  |  |
| 06 | Steps 02 through 05 will repeat until a Player leaves the match, or an end state is arrived at. |  |  |

|  |  |  |
| --- | --- | --- |
| **Exception Conditions** | | |
| **Exception Situations** | **Action(s) on Exception** | **Adds/Alt Use Case #** |
| Player enters an illegal move | User is notified with an error sound and the move transmitted will not be made. |  |
| Player loses connection during the game | The Player who disconnected will forfeit the match, the Server will update the database, and the connected Player will return to the lobby, whereas the disconnected Player will be returned to the log in screen. |  |

|  |  |
| --- | --- |
| **Post-Conditions** | |
| **#** | **Description** |
| 1 | An end state is reached, and that state is recorded for each Player from the Server to the database. |
| 2 |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Assumptions** | | | | | |
| **#** | **Assumption** | **Date**  **Raised** | **Raised**  **By** | **Date**  **Verified** | **Verified By** |
| 1 | That a player knows how to play chess | 12/13/14 | Stewart Taylor |  |  |

|  |  |
| --- | --- |
| **General Information** | |
| Use Case ID Number : UC004  Subject Area : Connecting to Server  Description : Players connecting to a server to play chess | Responsible Analyst : Todd Breedlove |

|  |  |
| --- | --- |
| **Requirements/Feature Trace** | |
| REQ# | Requirements Name and / or Short Description |
| 12 | Allow user to play chess against another player on application. |

|  |  |  |
| --- | --- | --- |
| **Actors** | | |
| **Actor Name** | **P/S** | **Brief Description** |
| User | P | User connects to Raspberry Pi. |
| Raspberry Pi | S | Runs on port 2500 and handles incoming connections. |

|  |  |
| --- | --- |
| **Pre-Conditions** | |
| # | Description |
| 1 | User must have entered a valid username and password. |

|  |
| --- |
| **Start Stimulus** |
| User clicks log in button |

|  |  |  |  |
| --- | --- | --- | --- |
| **Use Case Main Course Steps** | | | |
| **Number** | **Description** | **Adds/Alt UC Name/Number** | **Bus Rule(s)#** |
| 01 | Server silently listens in background for incoming connections. |  |  |
| 02 | User sends connection request to server |  |  |
| 03 | Server creates new thread to compensate for new request. |  |  |
| 04 | Server sends acknowledge that user is connected to server. |  |  |

|  |  |  |
| --- | --- | --- |
| **Exception Conditions** | | |
| **Exception Situations** | **Action(s) on Exception** | **Adds/Alt Use Case #** |
| Server times out | User notified that server timed out and prompted to retry connecting. |  |
| Maximum number of pending connections exceeded | Server notifies user(s) that server is full and prompted to retry at another time. |  |
|  |  |  |

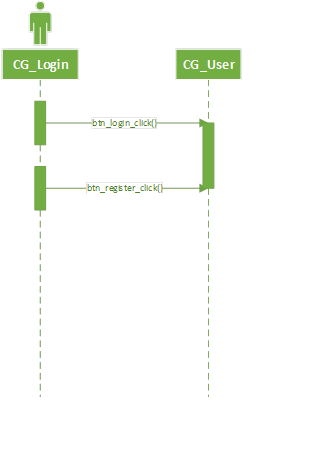
|  |  |
| --- | --- |
| **Post-Conditions** | |
| **#** | **Description** |
| 1 | A message stating the user is connected to the Chessgames Mobile Server |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Assumptions** | | | | | |
| **#** | **Assumption** | **Date**  **Raised** | **Raised**  **By** | **Date**  **Verified** | **Verified By** |
| 1 | Raspberry Pi is running 24/7. | 12/13/14 | Aaron Costner |  |  |

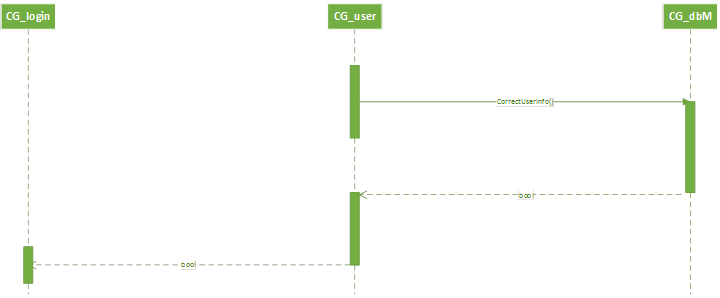
# Sequence Diagrams

Below are three sequence diagrams.

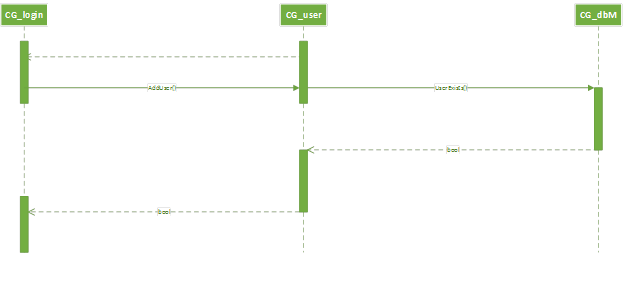
## User selects either login or register button



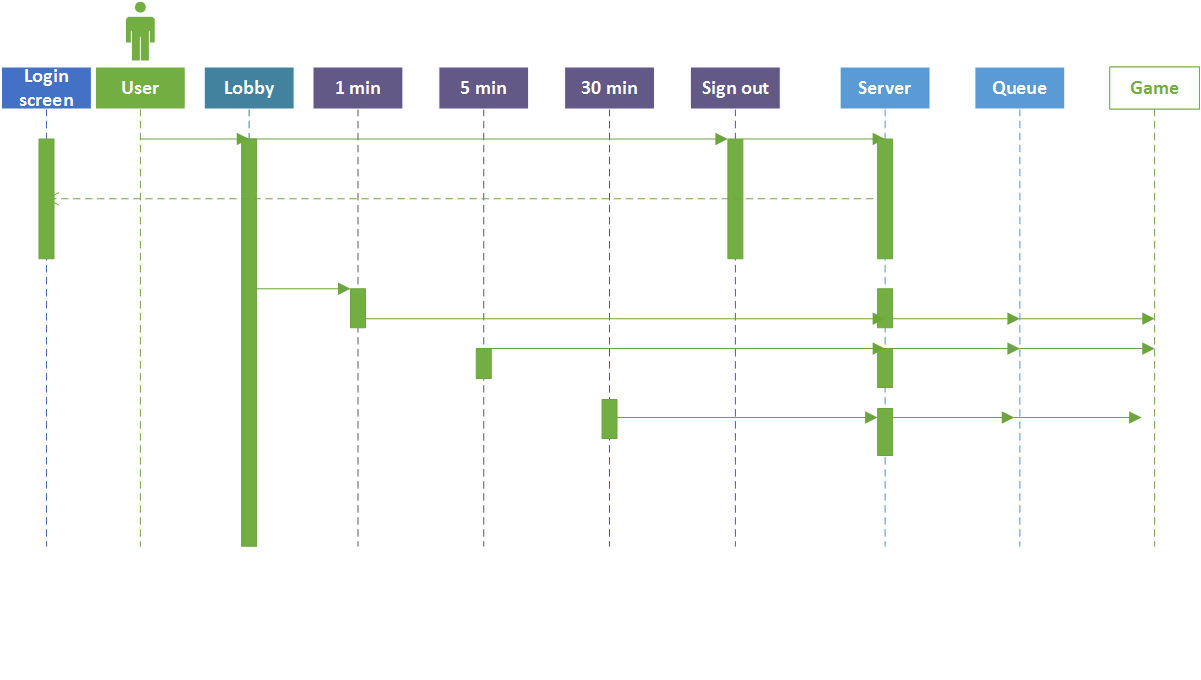
## Login the user if the user info is correct



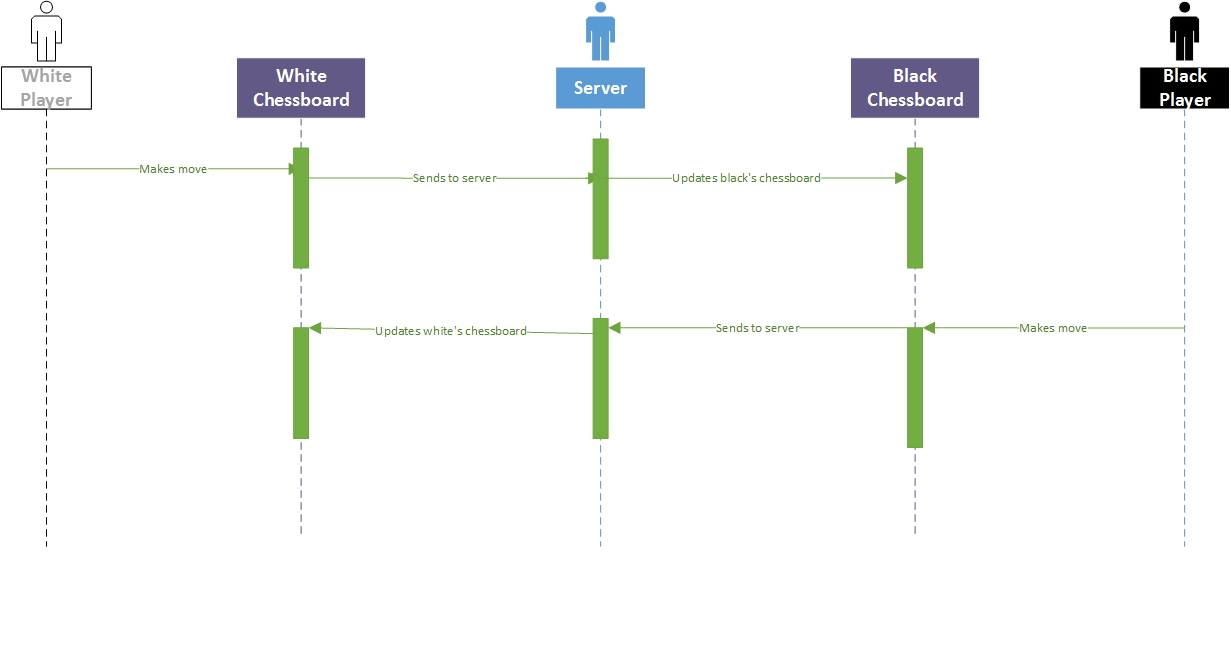
## Register the user if the user does not exist and the username, password and email pass validation



## User selects chess game pool



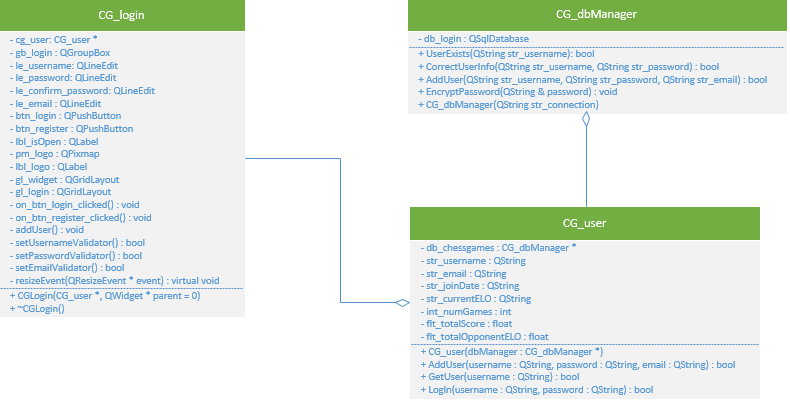
## User transmits move to other user



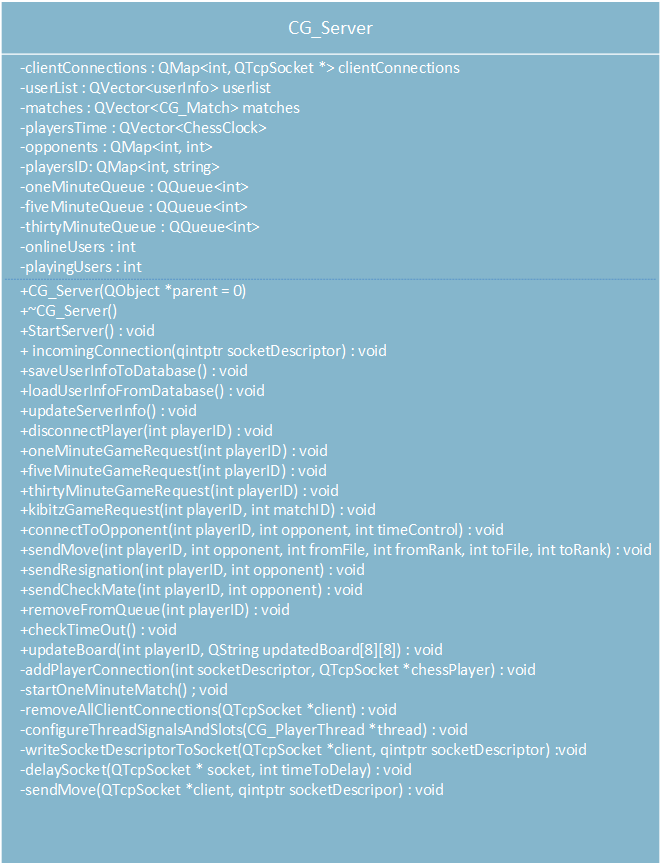
## User connects to dedicated server



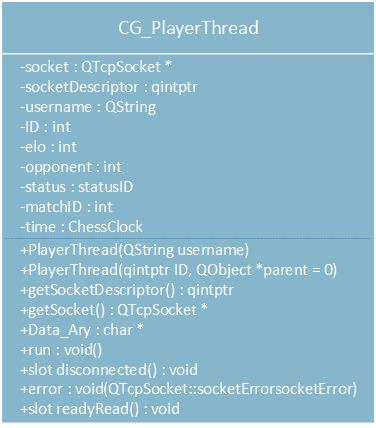
# Class Diagrams



## CG\_Server



## CG\_PlayerThread



## CG\_Match



# C++ Code

## Main.cpp

#include "CG\_login.h"

#include "CG\_user.h"

#include "CG\_dbManager.h"

#include "CG\_board.h"

#include <QApplication>

#include <QQmlApplicationEngine>

#include <QQmlContext>

#include <QSettings>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Author: ChessGames

\* Date Created: 11/07/2014

\* Date Last Modified: 3/15/2015

\* File Name: src/main.cpp

\*

\* Overview:

\* • We worked on and mostly completed our database design with an Entity

\* Relationship Diagram, SQL script for creating the database which we

\* later converted into SQLite and sample inserts.

\*

\* • Connected user to login system and deployed a “Hello World!”

\* application to and Android device.

\*

\* • Set up the game lobby and the server. This had a foundation for

\* different game pools (1 minute, 5 minute, or 30 minute games).

\*

\* • Converted over to QML for our presentation layer.

\*

\* • Created a game window with a board being displayed and the basic

\* layout of the board itself, with piece movements and sounds.

\*

\* • Created a loading screen to transition into game.

\*

\* • Validated piece movements, added piece dragging / zooming, cleaned

\* up layout.

\*

\* • Implemented chess clocks.

\*

\* • Created a history class to review moves at the end of a game.

\*

\* • Created a settings page currently to change country flags in the

\* game window.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main(int argc, char \*argv[])

{

QApplication app(argc, argv);

QQmlApplicationEngine engine;

//Set up the database manager (Persistence Layer)

CG\_dbManager db\_manager("chessgames.db");

//Set up the chessgames user (Business Layer)

CG\_user cg\_user(&db\_manager);

//Set up the board

CG\_board cg\_board;

QLabel validator\_feedback;

CG\_validator cg\_validator(validator\_feedback);

engine.rootContext()->setContextProperty("User", &cg\_user);

engine.rootContext()->setContextProperty("Validator", &cg\_validator);

engine.rootContext()->setContextProperty("BoardLogic", &cg\_board);

engine.load(QUrl(QStringLiteral("qrc:/main.qml")));

return app.exec();

}

## CG\_user.h

#ifndef CG\_USER\_H

#define CG\_USER\_H

#include <QString>

#include <QObject>

#include "CG\_dbManager.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Class: CG\_user

\*

\* Constructors:

\* CG\_user(CG\_dbManager \* dbManager)

\* Sets up the user to reference the main db manager.

\*

\* Methods:

\* bool AddUser(QString username, QString password, QString email)

\* Returns whether or not the user was successfully added to the

\* database.

\* bool GetUser(QString username)

\* Returns whether or not a user with the given username exists.

\* bool LogIn(QString username, QString password)

\* Returns whether or not the user info given is correct.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class CG\_user : public QObject

{

Q\_OBJECT

public:

CG\_user(CG\_dbManager \* dbManager);

public slots:

bool addUser(QString username, QString password, QString email);

bool getUser(QString username);

bool logIn(QString username, QString password);

QString getUsername() { return str\_username; }

QString getCurrentELO() { return str\_currentELO; }

private:

CG\_dbManager \* db\_chessgames;

QString str\_username;

QString str\_email;

QString str\_joinDate;

QString str\_currentELO;

int int\_numGames;

float flt\_totalScore;

float flt\_totalOpponentELO;

};

#endif

## CG\_user.cpp

#include "CG\_user.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Constructor. Passes the reference of the

\* main db manager to the class.

\*

\* Entry: The db manager has been set up.

\*

\* Exit: A reference to the main db manager has been passed

\* to the class.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CG\_user::CG\_user(CG\_dbManager \* dbManager)

{

db\_chessgames = dbManager;

str\_username = "Radjabov";

str\_currentELO = "2710";

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: To check whether the user entered a valid

\* username and password.

\*

\* Entry: User has clicked the login button

\*

\* Exit: Returns whether or not the username and password

\* are correct.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool CG\_user::logIn(QString username, QString password)

{

bool logged\_in = false;

if (db\_chessgames->correctUserInfo(username, password))

{

logged\_in = true;

str\_username = username;

str\_currentELO = db\_chessgames->getCurrentELO(username);

}

return logged\_in;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: To check if user exists in database.

\*

\* Entry: User has clicked the register button

\*

\* Exit: Returns whether or not the user exists in the

\* database.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool CG\_user::getUser(QString username)

{

bool user\_exists = false;

if(db\_chessgames->userExists(username))

user\_exists = true;

return user\_exists;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: To add a user to the database.

\*

\* Entry: Username and password has been validated at login

\* screen.

\*

\* Exit: Returns whether or not the user has been

\* successfully added to the database.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool CG\_user::addUser(QString username, QString password, QString email)

{

bool added\_user = false;

if (db\_chessgames->addUser(username, password, email))

added\_user = true;

return added\_user;

}

## CG\_dbManager.h

#ifndef CG\_DBMANAGER\_H

#define CG\_DBMANAGER\_H

#include <QString>

#include <QVariant>

#include <QObject>

#include <QtSql/QSqlDatabase>

#include <QtSql/QSqlQuery>

#include <QFile>

#include <QCryptographicHash> // Needed for encrypting in SHA256

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Class: CG\_dbManager

\*

\* Constructors:

\* CG\_dbManager(QString str\_connection)

\* Opens the SQLite database for connection based upon the db path passed.

\*

\* Methods:

\* bool UserExists(QString str\_username)

\* Returns whether or not a user with the passed username exists in

\* the database.

\* bool CorrectUserInfo(QString str\_username, QString str\_password)

\* Returns whether or not the username and password info are correct

\* and in the database.

\* bool AddUser(QString str\_username, QString str\_password, QString str\_email)

\* Returns true if the user is successfully added into the database.

\* QString getCurrentELO(QString str\_username)

\* Returns the current ELO of a user.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class CG\_dbManager : public QObject

{

Q\_OBJECT

public:

CG\_dbManager(QString str\_connection);

public slots:

bool userExists(QString str\_username);

bool correctUserInfo(QString str\_username, QString str\_password);

bool addUser(QString str\_username, QString str\_password, QString str\_email);

void encryptPassword(QString & password);

QString getCurrentELO(QString str\_username);

private:

QSqlDatabase db\_login;

};

#endif

## CG\_dbManager.cpp

#include "CG\_dbManager.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Constructor. Opens the database based upon the

\* db path specified.

\*

\* Entry: User has opened the app.

\*

\* Exit: The database is set to SQLite and the path to the

\* db is specified.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CG\_dbManager::CG\_dbManager(QString str\_connection)

{

QFile dfile("assets:/" + str\_connection);

if (dfile.exists())

{

dfile.copy("./" + str\_connection);

QFile::setPermissions("./" + str\_connection, QFile::WriteOwner | QFile::ReadOwner);

}

//Connect to database

db\_login = QSqlDatabase::addDatabase("QSQLITE");

db\_login.setDatabaseName("./" + str\_connection);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: To check whether or not the user entered the

\* correct username and password and it exists in

\* the database.

\*

\* Entry: User has clicked the login button

\*

\* Exit: Returns whether or not the login info is correct.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool CG\_dbManager::correctUserInfo(QString str\_username, QString str\_password)

{

int count = 0;

if(db\_login.open())

{

//Calls encryption password

encryptPassword(str\_password);

QSqlQuery qry( db\_login );

qry.prepare( "SELECT \* FROM CG\_user WHERE Username= ? AND Passwd= ? COLLATE NOCASE" );

qry.addBindValue(str\_username);

qry.addBindValue(str\_password);

if(qry.exec())

for (; qry.next(); count++);

db\_login.close();

}

//Returns if the database finds a username and password match

return count > 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: To check whether or not the user exists in the

\* database.

\*

\* Entry: User has clicked the register button

\*

\* Exit: Returns whether or not the user exists.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool CG\_dbManager::userExists(QString str\_username)

{

int count = 0;

if(db\_login.open())

{

QSqlQuery qry( db\_login );

qry.prepare( "SELECT \* FROM CG\_user WHERE Username= ? COLLATE NOCASE" );

qry.addBindValue(str\_username);

if(qry.exec())

for (; qry.next(); count++);

db\_login.close();

}

return count > 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: To add a user into the database with the passed

\* parameters: username, password and email.

\*

\* Entry: User has clicked the register button

\*

\* Exit: Returns whether or not the user was successfully

\* added into the database.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool CG\_dbManager::addUser(QString str\_username, QString str\_password, QString str\_email)

{

bool added\_user = false;

if(db\_login.open())

{

//Calls encryption password

encryptPassword( str\_password );

QSqlQuery qry( db\_login );

qry.prepare( "INSERT INTO CG\_user (Username, Passwd, Email) VALUES(?, ?, ?)" );

qry.addBindValue(str\_username);

qry.addBindValue(str\_password);

qry.addBindValue(str\_email);

if(qry.exec())

added\_user = true;

db\_login.close();

}

return added\_user;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: To retrieve the current ELO rating of a user.

\*

\* Entry: User is logged in.

\*

\* Exit: Returns the current ELO of a user.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

QString CG\_dbManager::getCurrentELO(QString str\_username)

{

QString result;

if(db\_login.open())

{

QSqlQuery qry( db\_login );

qry.prepare( "SELECT CurrentELO FROM CG\_user WHERE Username= ? COLLATE NOCASE" );

qry.addBindValue(str\_username);

if(qry.exec())

for (int count = 0; qry.next(); count++)

result = qry.value(0).toString();

db\_login.close();

}

return result;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: To encrypt a password using the SHA256 hashing

\* function.

\*

\* Entry: User is attempting to login or register.

\*

\* Exit: Alters the password string based upon the SHA256

\* encryption.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void CG\_dbManager::encryptPassword(QString & password)

{

// Takes the text in the le\_password and converts it to Utf8, so it can be placed in a type of 'const char \*' next

QByteArray passwordInBytes = password.toUtf8();

// casting the data in passwordInBytes (currently in the form of 'Utf8' to a type of 'const char \*'

const char \* convertedPasswordToVerify = passwordInBytes.constData();

/\* Instantiating an object that will create the hashing key for the password. Takes an argument specifying

\* the encryption type you would like be executed on the string \*/

QCryptographicHash sha256PasswordEncryptionGenerator( QCryptographicHash::Sha256 );

// Adding the data to password encryption generator

sha256PasswordEncryptionGenerator.addData( convertedPasswordToVerify );

// Converting the password to the hash key using the result method and placing it in password.

password = (QString)sha256PasswordEncryptionGenerator.result();

}

## CG\_board.h

#ifndef CG\_BOARD\_H

#define CG\_BOARD\_H

#include <QObject>

#include <QString>

#include "CG\_history.h"

#include "CG\_square.h"

class CG\_board : public QObject

{

Q\_OBJECT

public:

//Default constructor

CG\_board();

//Copy COnstructor

CG\_board(const CG\_board & copy);

public slots:

bool move(int f\_source, int r\_source, int f\_dest, int r\_dest);

void callHistoryBackward();

void callHistoryForward();

QString getSquare(int source\_file, int source\_rank);

void resetBoard();

bool CheckForClearPath(int f\_source, int r\_source, int f\_dest, int r\_dest);

void CheckRookMovement(int f\_source, int r\_source, int f\_dest, int r\_dest, bool & valid);

void CheckBishopMovement(int f\_source, int r\_source, int f\_dest, int r\_dest, bool & valid);

bool CheckKingInCheck( int f\_source, int r\_source, int f\_dest, int r\_dest );

private:

CG\_square m\_board[8][8];

CG\_history m\_history;

bool m\_whiteToMove;

};

#endif // CG\_BOARD\_H

## CG\_board.cpp

#include "CG\_board.h"

#include <QDebug>

//Default constructor

CG\_board::CG\_board() : m\_board(), m\_whiteToMove(true)

{

//Set up the CG\_squares to be the correct file and rank.

for ( int rank = 1; rank <= 8; rank++)

{

for ( int file = 1; file <= 8; file++)

{

m\_board[file-1][rank-1].setSquare((File)file, rank);

}

}

//Set up all of the white pieces on the board.

m\_board[0][0].setPiece(BLACK, "Rook");

m\_board[0][1].setPiece(BLACK, "Knight");

m\_board[0][2].setPiece(BLACK, "Bishop");

m\_board[0][3].setPiece(BLACK, "Queen");

m\_board[0][4].setPiece(BLACK, "King");

m\_board[0][5].setPiece(BLACK, "Bishop");

m\_board[0][6].setPiece(BLACK, "Knight");

m\_board[0][7].setPiece(BLACK, "Rook");

m\_board[1][0].setPiece(BLACK, "Pawn");

m\_board[1][1].setPiece(BLACK, "Pawn");

m\_board[1][2].setPiece(BLACK, "Pawn");

m\_board[1][3].setPiece(BLACK, "Pawn");

m\_board[1][4].setPiece(BLACK, "Pawn");

m\_board[1][5].setPiece(BLACK, "Pawn");

m\_board[1][6].setPiece(BLACK, "Pawn");

m\_board[1][7].setPiece(BLACK, "Pawn");

//Set up all of the black pieces on the board.

m\_board[7][0].setPiece(WHITE, "Rook");

m\_board[7][1].setPiece(WHITE, "Knight");

m\_board[7][2].setPiece(WHITE, "Bishop");

m\_board[7][3].setPiece(WHITE, "Queen");

m\_board[7][4].setPiece(WHITE, "King");

m\_board[7][5].setPiece(WHITE, "Bishop");

m\_board[7][6].setPiece(WHITE, "Knight");

m\_board[7][7].setPiece(WHITE, "Rook");

m\_board[6][0].setPiece(WHITE, "Pawn");

m\_board[6][1].setPiece(WHITE, "Pawn");

m\_board[6][2].setPiece(WHITE, "Pawn");

m\_board[6][3].setPiece(WHITE, "Pawn");

m\_board[6][4].setPiece(WHITE, "Pawn");

m\_board[6][5].setPiece(WHITE, "Pawn");

m\_board[6][6].setPiece(WHITE, "Pawn");

m\_board[6][7].setPiece(WHITE, "Pawn");

CG\_square tempFrom;

CG\_square tempTo;

m\_history.setNext(m\_board, tempFrom, tempTo);

}

CG\_board::CG\_board(const CG\_board & copy ) : m\_board(), m\_whiteToMove(copy.m\_whiteToMove), m\_history(m\_board)

{

for ( int rank = 1; rank <= 8; rank++)

{

for ( int file = 1; file <= 8; file++)

{

m\_board[file-1][rank-1].setSquare((File)file, rank);

if ( copy.m\_board[file-1][rank-1].getPiece() != nullptr )

{

m\_board[file-1][rank-1].setPiece(CG\_Color(copy.m\_board[file-1][rank-1].getPiece()->getPieceColor()),

copy.m\_board[file-1][rank-1].getPiece()->getPieceName());

}

else

{

m\_board[file-1][rank-1].setPiece();

}

}

}

}

bool CG\_board::move(int f\_source, int r\_source, int f\_dest, int r\_dest)

{

bool move\_made = false;

CG\_Color currentPlayer = m\_whiteToMove ? WHITE : BLACK;

//If the square selected has a piece in it.

if(m\_board[f\_source][r\_source].getPiece() != nullptr)

{

//If the piece selected is of the color of the player moving,

//continue onward. Otherwise, this is an invalid move.

//White pieces cannot be moved unless it is the white player moving them,

//and vice versa for black.

if((m\_board[f\_source][r\_source].getPiece()->getPieceColor() == currentPlayer))

{

//If the piece being moved is a King and the destination doesn't have

//a piece in it, and the king is actually moving only by rank.

if (m\_board[f\_source][r\_source].getPiece()->getPieceName() == "King"

&& m\_board[f\_dest][r\_dest].getPiece() == nullptr

&& abs(r\_source - r\_dest) == 2 && abs(f\_source - f\_dest) == 0

&& static\_cast<CG\_king \*>(m\_board[f\_source][r\_source].getPiece())->getHasMoved() == false)

{

//This figures out which rank to use. If the king is moving right,

//the rank should be the rook on the right, and vice versa.

int rook\_rank = 7 == (r\_dest + 1) ? 7 : 0;

//This determines what side to put the rook on after castling.

//If the King castled right, the Rook should be on the King's left,

//and vice versa.

int rook\_side = rook\_rank == 0 ? 1 : -1;

//We need to make sure that the Rook is in the proper place, that it is of the same color,

//and that it hasn't moved.

if(m\_board[f\_source][r\_source].getPiece()->*move*((File) f\_source, (Rank) r\_source, (File) f\_dest, (Rank) r\_dest)

&& (m\_board[f\_source][rook\_rank].getPiece()->getPieceColor() == currentPlayer &&

m\_board[f\_source][rook\_rank].getPiece()->getPieceName() == "Rook" &&

static\_cast<CG\_rook \*>(m\_board[f\_source][rook\_rank].getPiece())->getHasMoved() == false))

{

//Check to see if the path between the King and the Rook is clear.

if(CheckForClearPath( f\_source, r\_source, f\_source, rook\_rank))

{

//Check to see if the King were to move, would the King be in check.

if ( !CheckKingInCheck(f\_source, r\_source, f\_dest, r\_dest))

{

m\_board[f\_dest][r\_dest].setPiece(m\_board[f\_source][r\_source].getPiece());

m\_board[f\_dest][r\_dest].getPiece()->*UpdatePiece*();

m\_board[f\_source][r\_source].setPiece();

m\_board[f\_dest][r\_dest+rook\_side].setPiece(m\_board[f\_source][rook\_rank].getPiece());

m\_board[f\_dest][r\_dest+rook\_side].getPiece()->*UpdatePiece*();

m\_board[f\_dest][rook\_rank].setPiece();

move\_made = true;

m\_whiteToMove = !m\_whiteToMove;

}

}

}

}

//If the destination doesn't have a piece

//or the destination has a piece of a different color

else if(m\_board[f\_dest][r\_dest].getPiece() == nullptr ||

m\_board[f\_source][r\_source].getPiece()->getPieceColor() !=

m\_board[f\_dest][r\_dest].getPiece()->getPieceColor())

{

//If the piece can actually make that movement pattern

if(m\_board[f\_source][r\_source].getPiece()->*move*((File) f\_source, (Rank) r\_source, (File) f\_dest, (Rank) r\_dest))

{

if(CheckForClearPath( f\_source, r\_source, f\_dest, r\_dest))

{

//Make a check to see if the moving player's king would be

//in check after that move would be made. If so, then that

//movement cannot be made.

if ( !CheckKingInCheck( f\_source, r\_source, f\_dest, r\_dest ))

{

//Move the piece from its source to the destination.

m\_board[f\_dest][r\_dest].setPiece(m\_board[f\_source][r\_source].getPiece());

//Update the piece to show that it has moved.

m\_board[f\_dest][r\_dest].getPiece()->*UpdatePiece*();

//Remove the piece from its original location.

m\_board[f\_source][r\_source].setPiece();

move\_made = true;

m\_whiteToMove = !m\_whiteToMove;

//Add move to history list

m\_history.setNext(m\_board, m\_board[f\_source][r\_source], m\_board[f\_dest][r\_dest]);

}

}

}

}

}

}

return move\_made;

//Make the destination CG\_square hold the CG\_piece from the source CG\_square,

//and make the source CG\_square hold a null pointer to a CG\_piece.

//Thus, this empties the source CG\_square of its CG\_piece.

//m\_board[f\_dest][r\_dest].setPiece(m\_board[f\_source][r\_source].getPiece());

//m\_board[f\_source][r\_source].setPiece(0);

}

QString CG\_board::getSquare(int source\_file, int source\_rank)

{

//Makes a temporary QString of the CG\_square's CG\_piece name.

QString ret\_str = m\_board[source\_file][source\_rank].getPieceName();

char temp = '\0';

//If the CG\_piece pointer in CG\_square isn't a null pointer,

//Add an integer to the end representing the color of the piece.

if (m\_board[source\_file][source\_rank].getPiece() != nullptr)

{

temp = m\_board[source\_file][source\_rank].getPiece()->getPieceColor() + 48;

ret\_str += QString(temp);

}

//Return the QString representing the CG\_piece in the CG\_square.

return ret\_str;

}

bool CG\_board::CheckForClearPath(int f\_to, int r\_to, int f\_from, int r\_from)

{

bool path\_clear = true;

QString pieceName = "";

if ( m\_board[f\_to][r\_to].getPiece() != nullptr )

{

pieceName = m\_board[f\_to][r\_to].getPiece()->getPieceName();

}

//If the movement matches that of a rook's movement patterns,

//meaning it is moving vertically or horizontally.

if ( f\_to == f\_from || r\_to == r\_from )

{

CheckRookMovement(f\_to, r\_to, f\_from, r\_from, path\_clear);

}

//If the movement matches that of a bishop's movement patterns,

//meaning it is moving diagonally.

else if ( abs(f\_from - f\_to) == abs(r\_to - r\_from) )

{

CheckBishopMovement(f\_to, r\_to, f\_from, r\_from, path\_clear);

}

return path\_clear;

}

void CG\_board::resetBoard()

{

//Set up all of the white pieces on the board.

m\_board[0][0].setPiece(BLACK, "Rook");

m\_board[0][1].setPiece(BLACK, "Knight");

m\_board[0][2].setPiece(BLACK, "Bishop");

m\_board[0][3].setPiece(BLACK, "Queen");

m\_board[0][4].setPiece(BLACK, "King");

m\_board[0][5].setPiece(BLACK, "Bishop");

m\_board[0][6].setPiece(BLACK, "Knight");

m\_board[0][7].setPiece(BLACK, "Rook");

m\_board[1][0].setPiece(BLACK, "Pawn");

m\_board[1][1].setPiece(BLACK, "Pawn");

m\_board[1][2].setPiece(BLACK, "Pawn");

m\_board[1][3].setPiece(BLACK, "Pawn");

m\_board[1][4].setPiece(BLACK, "Pawn");

m\_board[1][5].setPiece(BLACK, "Pawn");

m\_board[1][6].setPiece(BLACK, "Pawn");

m\_board[1][7].setPiece(BLACK, "Pawn");

for (int row = 2; row < 7; row++)

for (int col = 0; col < 8; col++)

m\_board[row][col].setPiece(nullptr);

//Set up all of the black pieces on the board.

m\_board[7][0].setPiece(WHITE, "Rook");

m\_board[7][1].setPiece(WHITE, "Knight");

m\_board[7][2].setPiece(WHITE, "Bishop");

m\_board[7][3].setPiece(WHITE, "Queen");

m\_board[7][4].setPiece(WHITE, "King");

m\_board[7][5].setPiece(WHITE, "Bishop");

m\_board[7][6].setPiece(WHITE, "Knight");

m\_board[7][7].setPiece(WHITE, "Rook");

m\_board[6][0].setPiece(WHITE, "Pawn");

m\_board[6][1].setPiece(WHITE, "Pawn");

m\_board[6][2].setPiece(WHITE, "Pawn");

m\_board[6][3].setPiece(WHITE, "Pawn");

m\_board[6][4].setPiece(WHITE, "Pawn");

m\_board[6][5].setPiece(WHITE, "Pawn");

m\_board[6][6].setPiece(WHITE, "Pawn");

m\_board[6][7].setPiece(WHITE, "Pawn");

//Reset board history

m\_history.resetHistory();

//Reset white to first

m\_whiteToMove = true;

}

void CG\_board::CheckRookMovement(int f\_source, int r\_source, int f\_dest, int r\_dest, bool & valid)

{

//This checks a vertical movement on the board for any pieces

//inbetween the source and destination squares.

if ( f\_source == f\_dest )

{

//This loop should check every square inbetween the source and

//destination squares to see if a piece exists in that square.

for(int i = r\_dest < r\_source ? r\_dest + 1 : r\_source + 1;

(i < (r\_dest < r\_source ? r\_source : r\_dest)) && valid;

i++)

{

//If a square in the path contains a piece, then the

//movement cannot be a valid movement.

if(m\_board[f\_dest][i].getPiece() != nullptr)

{

valid = false;

}

}

}

else //Same thing as vertical, except for horizontal movement.

{

//This loop should check every square inbetween the source and

//destination squares to see if a piece exists in that square.

for(int i = f\_dest < f\_source ? f\_dest + 1 : f\_source + 1;

(i < (f\_dest < f\_source ? f\_source : f\_dest)) && valid;

i++)

{

//If a square in the path contains a piece, then the

//movement cannot be a valid movement.

if(m\_board[i][r\_dest].getPiece() != nullptr)

{

valid = false;

}

}

}

}

void CG\_board::CheckBishopMovement(int f\_source, int r\_source, int f\_dest, int r\_dest, bool &valid)

{

//These variables help decide how to adjust the file and rank

//when checking the squares diagonally. If the file source

//is larger than the file destination, it will set the movement

//pattern to move down by 1 (-1) each time, or vice versa.

int f\_movement = f\_source > f\_dest ? -1 : 1;

int r\_movement = r\_source > r\_dest ? -1 : 1;

//This is going to check each square inbetween the source and destination

//diagonally to see if a piece exists in any of those squares.

//If a piece is found, then the movement isn't valid.

for( int i = f\_source + f\_movement,

j = r\_source + r\_movement;

i != f\_dest && j != r\_dest;

i += f\_movement, j += r\_movement)

{

//If a square in the path has a piece in it, then the

//movement cannot be a valid movement.

if(m\_board[i][j].getPiece() != nullptr)

{

valid = false;

}

}

}

bool CG\_board::CheckKingInCheck(int f\_source, int r\_source, int f\_dest, int r\_dest)

{

bool king\_inCheck = false;

CG\_board temp\_board(\*this);

int king\_rank = 1;

int king\_file = 1;

bool king\_found = false;

CG\_Color playerMoving = m\_whiteToMove ? WHITE : BLACK;

//This finds the king of the player currently moving pieces.

for( int rank = 1; rank <= 8 && !king\_found; rank++ )

{

for( int file = 1; file <= 8 && !king\_found; file++ )

{

//If the king of the player currently moving pieces exists in this location,

//mark that location for future use in the method.

if( temp\_board.m\_board[file-1][rank-1].getPiece() != nullptr &&

(CG\_Color(temp\_board.m\_board[file-1][rank-1].getPiece()->getPieceColor()) == playerMoving)

&& temp\_board.m\_board[file-1][rank-1].getPiece()->getPieceName() == "King"

)

{

king\_rank = rank;

king\_file = file;

king\_found = true;

}

}

}

//If the king is the piece moving, we'll update our king\_rank & king\_file

//to show that for validation purposes.

if ( king\_rank == r\_source + 1 && king\_file == f\_source + 1 )

{

king\_rank = r\_dest + 1;

king\_file = f\_dest + 1;

}

//Move the piece on the temporary board to test the movement out.

temp\_board.m\_board[f\_dest][r\_dest].setPiece(temp\_board.m\_board[f\_source][r\_source].getPiece());

temp\_board.m\_board[f\_source][r\_source].setPiece();

//Check to see if any of the opposing player's pieces is capable of taking the

//king of the player currently moving pieces.

for ( int rank = 1; rank <= 8 && !king\_inCheck; rank++ )

{

for ( int file = 1; file <= 8 && !king\_inCheck; file++ )

{

if ( temp\_board.m\_board[file-1][rank-1].getPiece() != nullptr

&& temp\_board.m\_board[file-1][rank-1].getPiece()->getPieceColor() != playerMoving )

{

if ( temp\_board.m\_board[file-1][rank-1].getPiece()->*move*(File(king\_file-1),

Rank(king\_rank-1), File(file-1), Rank(rank-1)))

{

if (temp\_board.CheckForClearPath(file-1, rank-1, king\_file-1, king\_rank-1))

{

king\_inCheck = true;

}

}

}

}

}

return king\_inCheck;

}

void CG\_board::callHistoryBackward()

{

m\_history.moveBack();

//Set each individual square in board to temp

for(int rank = 0; rank < NUMBER\_OF\_RANKS; rank++)

{

for (int file = 0; file < NUMBER\_OF\_RANKS; file++)

{

m\_board[rank][file].setRank(m\_history.getCurrentBoard()[rank][file].getRank());

m\_board[rank][file].setFile(m\_history.getCurrentBoard()[rank][file].getFile());

m\_board[rank][file].setPiece(m\_history.getCurrentBoard()[rank][file].getPiece());

}

}

}

void CG\_board::callHistoryForward()

{

m\_history.moveForward();

//Set each individual square in board to temp

for(int rank = 0; rank < NUMBER\_OF\_RANKS; rank++)

{

for (int file = 0; file < NUMBER\_OF\_RANKS; file++)

{

m\_board[rank][file].setRank(m\_history.getCurrentBoard()[rank][file].getRank());

m\_board[rank][file].setFile(m\_history.getCurrentBoard()[rank][file].getFile());

m\_board[rank][file].setPiece(m\_history.getCurrentBoard()[rank][file].getPiece());

}

}

}

## CG\_history.h

#ifndef CG\_HISTORY\_H

#define CG\_HISTORY\_H

#include <QDebug>

#include <vector>

#include "CG\_square.h"

const int NUMBER\_OF\_RANKS = 8;

class CG\_history

{

public:

CG\_history();

CG\_history(CG\_square board[8][8]);

void setNext(CG\_square board[8][8], CG\_square from, CG\_square to);

void moveBack();

void moveForward();

CG\_square \*\* getCurrentBoard();

void resetHistory();

private:

std::vector<CG\_square \*\*> m\_boardList;

CG\_square m\_from;

CG\_square m\_to;

int m\_currentBoardIndex;

};

#endif // CG\_HISTORY\_H

## CG\_history.cpp

#include "CG\_history.h"

CG\_history::CG\_history(CG\_square board[8][8])

{

CG\_square \*\* temp = new CG\_square \* [NUMBER\_OF\_RANKS];

//Set each individual square in board to temp

for(int rank = 0; rank < NUMBER\_OF\_RANKS; rank++)

{

temp[rank] = new CG\_square[NUMBER\_OF\_RANKS];

for (int file = 0; file < NUMBER\_OF\_RANKS; file++)

{

temp[rank][file].setRank(board[rank][file].getRank());

temp[rank][file].setFile(board[rank][file].getFile());

temp[rank][file].setPiece(board[rank][file].getPiece());

}

}

for(int i = 0; i < NUMBER\_OF\_RANKS; i++)

delete temp[i];

delete [] temp;

m\_boardList.push\_back(temp);

m\_currentBoardIndex = -1;

}

CG\_history::CG\_history()

{

m\_currentBoardIndex = -1;

}

void CG\_history::setNext(CG\_square board[8][8], CG\_square from, CG\_square to)

{

m\_currentBoardIndex++;

CG\_square \*\* temp = new CG\_square \* [NUMBER\_OF\_RANKS];

for(int rank = 0; rank < NUMBER\_OF\_RANKS; rank++)

{

temp[rank] = new CG\_square[NUMBER\_OF\_RANKS];

for (int file = 0; file < NUMBER\_OF\_RANKS; file++)

{

temp[rank][file].setRank(board[rank][file].getRank());

temp[rank][file].setFile(board[rank][file].getFile());

temp[rank][file].setPiece(board[rank][file].getPiece());

}

}

m\_boardList.push\_back(temp);

// for(int i = 0; i < NUMBER\_OF\_RANKS; i++)

// delete [] temp[i];

// delete [] temp;

// m\_from = from;

// m\_to = to;

}

void CG\_history::moveBack()

{

if(m\_currentBoardIndex > 0)

m\_currentBoardIndex--;

}

void CG\_history::moveForward()

{

if(m\_currentBoardIndex < m\_boardList.size() - 1)

m\_currentBoardIndex++;

}

CG\_square \*\* CG\_history::getCurrentBoard()

{

return m\_boardList[m\_currentBoardIndex];

}

void CG\_history::resetHistory()

{

m\_boardList.clear();

m\_currentBoardIndex = -1;

}

## CG\_square.h

#ifndef CG\_SQUARE\_H

#define CG\_SQUARE\_H

#include <QObject>

#include <QString>

#include "CG\_rook.h"

#include "CG\_bishop.h"

#include "CG\_queen.h"

#include "CG\_king.h"

#include "CG\_knight.h"

#include "CG\_pawn.h"

#ifndef nullptr

#define nullptr 0

#endif

class CG\_square

{

public:

//Default constructor

CG\_square();

//Mutators

File getFile() const;

Rank getRank() const;

void setFile(File file);

void setRank(Rank rank);

void setSquare(File file, Rank rank);

void Aaron();

QString getPieceName() const;

CG\_piece \* getPiece() const;

void setPiece(CG\_piece \* piece = nullptr);

void setPiece(CG\_Color color, QString name);

CG\_square &operator=(const CG\_square & rhs);

private:

File m\_file;

Rank m\_rank;

CG\_piece \* m\_piece;

};

#endif // CG\_SQUARE\_H

## CG\_square.cpp

#include "CG\_square.h"

//Default Constructor

CG\_square::CG\_square()

: m\_file(a), m\_rank(1), m\_piece(nullptr)

{

}

//Sets the CG\_square's File.

void CG\_square::setFile(File file)

{

m\_file = file;

}

//Sets the CG\_square's Rank.

void CG\_square::setRank(Rank rank)

{

m\_rank = rank;

}

//Sets the CG\_square's File and Rank.

void CG\_square::setSquare(File file, Rank rank)

{

this->setFile(file);

this->setRank(rank);

}

//Returns the name of the CG\_piece in the CG\_square.

//If the pointer is null, it returns an empty string.

QString CG\_square::getPieceName() const

{

QString temp\_name = "";

if(m\_piece != 0)

{

temp\_name = m\_piece->getPieceName();

}

return temp\_name;

}

//Returns the pointer to the CG\_piece in CG\_square.

CG\_piece \* CG\_square::getPiece() const

{

return m\_piece;

}

//This sets the CG\_piece pointer to that of another pointer.

void CG\_square::setPiece(CG\_piece \* piece)

{

m\_piece = piece;

}

//This sets the CG\_piece pointer based on a CG\_Color and QString.

//Of the two setPiece's, this one's purpose is to actually allocate

//the data for that particular piece.

void CG\_square::setPiece(CG\_Color color, QString name)

{

if ( m\_piece != nullptr )

{

delete m\_piece;

}

//Depending on the piece name, this makes the pointer

//become a particular piece with the specified color.

if ( name == "Pawn" )

{

m\_piece = new CG\_pawn(color);

}

else if ( name == "Rook" )

{

m\_piece = new CG\_rook(color);

}

else if ( name == "Knight" )

{

m\_piece = new CG\_knight(color);

}

else if ( name == "Bishop" )

{

m\_piece = new CG\_bishop(color);

}

else if ( name == "Queen" )

{

m\_piece = new CG\_queen(color);

}

else

{

m\_piece = new CG\_king(color);

}

}

File CG\_square::getFile() const

{

return m\_file;

}

Rank CG\_square::getRank() const

{

return m\_rank;

}

CG\_square &CG\_square::operator=(const CG\_square & rhs)

{

//Check for self-assignment

if(this != &rhs)

{

CG\_piece \*\* temp = nullptr;

//Delete old data

delete [] m\_piece;

//Make deep copy of new data

m\_file = rhs.m\_file;

m\_rank = rhs.m\_rank;

temp = new CG\_piece \*[sizeof rhs];

m\_piece = \*temp;

}

//Return self-invoking object

return \*this;

}

## CG\_piece.h

#ifndef CG\_PIECE\_H

#define CG\_PIECE\_H

#include <QObject>

#include <QString>

//Definitions

typedef enum CG\_Color { WHITE = 0, BLACK } CG\_Color;

typedef int Rank;

enum File { a=0, b, c, d, e, f, g, h };

class CG\_piece : public QObject

{

Q\_OBJECT

public:

//Constructors

CG\_piece(CG\_Color pieceColor, QString pieceName);

//CG\_piece & operator=(CG\_piece & rhs );

//Virtual methods

virtual bool *move*(File f\_to, Rank r\_to, File f\_from, Rank r\_from);

virtual void *UpdatePiece*();

//Mutators

QString getPieceName() const;

int getPieceColor() const;

private:

CG\_Color m\_pieceColor;

QString m\_pieceName;

protected:

};

#endif // CG\_PIECE\_H

## CG\_piece.cpp

#include "CG\_piece.h"

//Default constructor

CG\_piece::CG\_piece(CG\_Color pieceColor, QString pieceName)

: m\_pieceColor(pieceColor), m\_pieceName(pieceName)

{

}

bool CG\_piece::*move*(File f\_to, Rank r\_to, File f\_from, Rank r\_from)

{

bool canMakeMove = false;

return canMakeMove;

}

void CG\_piece::*UpdatePiece*()

{

}

//Member functions

QString CG\_piece::getPieceName() const

{

return m\_pieceName;

}

int CG\_piece::getPieceColor() const

{

return (int) m\_pieceColor;

}

## CG\_pawn.h

#ifndef CG\_PAWN\_H

#define CG\_PAWN\_H

#include "CG\_piece.h"

//Declare pawn class and inherit from piece class

class CG\_pawn : public CG\_piece

{

public:

//Constructors

CG\_pawn(CG\_Color color = WHITE);

//Member Functions

bool *move*(File f\_to, Rank r\_to, File f\_from, Rank r\_from);

void *UpdatePiece*();

bool canPromote();

bool enPassant();

//CG\_piece promote(CG\_piece promoteTo);

private:

//Data Members

int direction;

bool m\_hasMoved;

};

#endif // CG\_PAWN\_H

## CG\_pawn.cpp

#include "CG\_pawn.h"

//Default constructor

CG\_pawn::CG\_pawn(CG\_Color color)

: CG\_piece(color, "Pawn"), m\_hasMoved(false)

{

if(color == WHITE)

{

direction = 1;

}

else

{

direction = -1;

}

}

//Member functions

bool CG\_pawn::*move*(File f\_to, Rank r\_to, File f\_from, Rank r\_from)

{

bool canMakeMove = false;

//The Pawn's movements are by far the most complex of the

//chess piece collection. For now, this only cares if the

//Pawn moves within its File, but eventually movements for

//capturing, en passant, and the 2 space first move will be

//accounted for.

if( !m\_hasMoved && abs(f\_to - f\_from) == 2 &&

abs(r\_to - r\_from) == 0 )

{

canMakeMove = true;

}

else if ( abs(f\_to - f\_from) == 1 &&

abs(r\_to - r\_from) == 0 &&

direction == f\_to - f\_from )

{

canMakeMove = true;

}

else if ( abs(f\_to - f\_from) == 1 &&

abs(r\_to - r\_from) == 1 &&

direction == f\_to - f\_from )

{

canMakeMove = true;

}

return canMakeMove;

}

void CG\_pawn::*UpdatePiece*()

{

//This updates the pawn to show that it has moved.

m\_hasMoved = true;

}

## CG\_rook.h

#ifndef CG\_ROOK\_H

#define CG\_ROOK\_H

#include <QObject>

#include "CG\_piece.h"

class CG\_rook : public CG\_piece

{

Q\_OBJECT

public:

CG\_rook(CG\_Color color = WHITE);

bool *move*(File f\_to, Rank r\_to, File f\_from, Rank r\_from);

void *UpdatePiece*();

bool getHasMoved();

private:

bool m\_hasMoved;

};

#endif // CG\_ROOK\_H

## CG\_rook.cpp

#include "CG\_rook.h"

//Default constructor

CG\_rook::CG\_rook(CG\_Color color)

: CG\_piece(color, "Rook"), m\_hasMoved(false)

{

}

//Member functions

bool CG\_rook::*move*(File f\_to, Rank r\_to, File f\_from, Rank r\_from)

{

bool canMakeMove = false;

//Check to see if Rook is moving only by File or only by Rank.

//aka, only vertically or horizontally.

if ( f\_from == f\_to || r\_from == r\_to )

{

canMakeMove = true;

}

return canMakeMove;

}

void CG\_rook::*UpdatePiece*()

{

//If a rook has moved, it needs to update the rook accordingly.

m\_hasMoved = true;

}

bool CG\_rook::getHasMoved()

{

return m\_hasMoved;

}

## CG\_bishop.h

#ifndef CG\_BISHOP\_H

#define CG\_BISHOP\_H

#include <QObject>

#include "CG\_piece.h"

class CG\_bishop : public CG\_piece

{

//Macro for all QObjects

Q\_OBJECT

public:

//Constructors

CG\_bishop(CG\_Color color = WHITE);

//Member functions

bool *move*(File f\_to, Rank r\_to, File f\_from, Rank r\_from);

private:

};

#endif // CG\_BISHOP\_H

## CG\_bishop.cpp

#include "CG\_bishop.h"

//Default constructor

CG\_bishop::CG\_bishop(CG\_Color color)

: CG\_piece(color, "Bishop")

{

}

//Member functions

bool CG\_bishop::*move*(File f\_to, Rank r\_to, File f\_from, Rank r\_from)

{

//Assume bishop cannot move until proven otherwise

bool canMakeMove = false;

//Check to see if Bishop only moves diagonally.

//The way it does this is through the absolute difference in its

//Rank and File. If both differences are equal, then the piece

//moved diagonally.

if ( abs(f\_from - f\_to) == abs(r\_to - r\_from) )

canMakeMove = true;

//Return whether bishop can move or not.

return canMakeMove;

}

## CG\_knight.h

#ifndef CG\_KNIGHT\_H

#define CG\_KNIGHT\_H

#include "CG\_piece.h"

//Declare knight class and inherit from piece class

class CG\_knight : public CG\_piece

{

public:

//Constructors

CG\_knight(CG\_Color color = WHITE);

//Member functions

bool *move*(File f\_to, Rank r\_to, File f\_from, Rank r\_from);

private:

//Data members

};

#endif // CG\_KNIGHT\_H

## CG\_knight.cpp

#include "CG\_knight.h"

//Default constructor

CG\_knight::CG\_knight(CG\_Color color)

: CG\_piece(color, "Knight")

{

}

//Member functions

bool CG\_knight::*move*(File f\_to, Rank r\_to, File f\_from, Rank r\_from)

{

//Assume knight cannot move until proven otherwise

bool canMakeMove = false;

//A Knight moves in an 'L' pattern, which is essentially either

//moving 1 File away and 2 Ranks away from its original position,

//or moving 2 Files away and 1 Rank away from its original position.

if ( (abs(f\_to - f\_from) == 2 && abs(r\_to - r\_from) == 1) ||

(abs(f\_to - f\_from) == 1 && abs(r\_to - r\_from) == 2) )

{

canMakeMove = true;

}

//Return whether knight can move or not.

return canMakeMove;

}

## CG\_queen.h

#ifndef CG\_QUEEN\_H

#define CG\_QUEEN\_H

#include <QObject>

#include "CG\_piece.h"

class CG\_queen : public CG\_piece

{

Q\_OBJECT

public:

//Constructors

CG\_queen(CG\_Color color = WHITE);

//Member functions

bool *move*(File f\_to, Rank r\_to, File f\_from, Rank r\_from);

private:

};

#endif // CG\_QUEEN\_H

## CG\_queen.cpp

#include "CG\_queen.h"

//Default constructor

CG\_queen::CG\_queen(CG\_Color color)

: CG\_piece(color, "Queen")

{

}

//Member functions

bool CG\_queen::*move*(File f\_to, Rank r\_to, File f\_from, Rank r\_from)

{

bool canMakeMove = false;

//Check to see if Queen is moving only by File or only by Rank.

//aka, only vertically or horizontally.

if ( f\_from == f\_to || r\_from == r\_to )

{

canMakeMove = true;

}

//Check to see if Queen only moves diagonally.

if ( abs(f\_from - f\_to) == abs(r\_to - r\_from) )

{

canMakeMove = true;

}

return canMakeMove;

}

## CG\_king.h

#ifndef CG\_KING\_H

#define CG\_KING\_H

#include "CG\_piece.h"

//Declare king class and inherit from piece

class CG\_king : public CG\_piece

{

public:

//Constructors

CG\_king(CG\_Color color = WHITE);

//Member functions

bool *move*(File f\_to, Rank r\_to, File f\_from, Rank r\_from);

void *UpdatePiece*();

bool getHasMoved();

private:

bool m\_hasMoved;

//Data members

};

#endif // CG\_KING\_H

## CG\_king.cpp

#include "CG\_king.h"

//Default constructor

CG\_king::CG\_king(CG\_Color color)

:CG\_piece(color, "King"), m\_hasMoved(false)

{

}

//Member functions

bool CG\_king::*move*(File f\_to, Rank r\_to, File f\_from, Rank r\_from)

{

bool canMakeMove = false;

//This checks to see if the King is attempting to castle.

//This requires that a square 2 ranks away is selected without changing

//the file, and that the king has not moved yet.

if ( (abs(r\_to - r\_from) == 2 && abs(f\_to - f\_from) == 0 ) && !m\_hasMoved)

{

canMakeMove = true;

}

//This checks to see if the king is moving 1 square in any direction.

else if ( (abs(f\_to - f\_from) == 1 && abs(r\_to - r\_from) == 0) ||

(abs(f\_to - f\_from) == 0 && abs(r\_to - r\_from) == 1) ||

(abs(f\_to - f\_from) == 1 && abs(r\_to - r\_from) == 1))

{

canMakeMove = true;

}

return canMakeMove;

}

void CG\_king::*UpdatePiece*()

{

//If a king has moved, it needs to update the piece accordingly.

m\_hasMoved = true;

}

bool CG\_king::getHasMoved()

{

return m\_hasMoved;

}

# QML Code

## Main.qml

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* File Name: main.qml

\*

\* Overview:

\* This is the Main QML file for our presentation layer. This layer

\* allows us to switch among the different screens for:

\* - Login

\* - Lobby

\* - Game

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

import QtQuick 2.3

import QtQuick.Controls 1.2

import QtQuick.Controls.Styles 1.1

import QtQuick.Window 2.0

import QtGraphicalEffects 1.0

ApplicationWindow

{

id: *background*

visible: true

width: 400

height: 600

title: *qsTr*("Chessgames")

color: "#448ed3"

FontLoader

{

id: *customFont*

source: "fonts/CHOCD TRIAL\_\_\_.otf"

}

CG\_login

{

id: *login*

anchors.fill: *parent*

onLoggedIn: { *login*.visible = false

*lobby*.visible = true }

}

CG\_lobby

{

id: *lobby*

anchors.fill: *parent*

onGoBack: { *lobby*.visible = false

*login*.visible = true }

onStartOneMinuteGame: { *lobby*.visible = false

*loading*.visible = true

*loading*.timer.start()

*game*.gameTimeInMinutes = 1 }

onStartFiveMinuteGame: { *lobby*.visible = false

*loading*.visible = true

*loading*.timer.start()

*game*.gameTimeInMinutes = 5 }

onStartThirtyMinuteGame: { *lobby*.visible = false

*loading*.visible = true

*loading*.timer.start()

*game*.gameTimeInMinutes = 30 }

onSettings:

{

*lobby*.visible = false

*settings*.visible = true

}

visible: false

}

CG\_settings

{

id: *settings*

anchors.fill: *parent*

onGoBack:

{

*settings*.visible = false

*lobby*.visible = true

}

onSettingsChanged:

{

*game*.playerFlagFrame = *settings*.countryFlag

*settings*.visible = false

*lobby*.visible = true

}

visible: false

}

CG\_loading

{

id: *loading*

anchors.fill: *parent*

onMatched: { *loading*.visible = false

*game*.visible = true }

visible: false

}

CG\_game

{

id: *game*

anchors.fill: *parent*

onFinished: { *game*.visible = false

*lobby*.visible = true }

visible: false

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Get the background width of the app for all

\* that need access to it.

\*

\* Entry: User has opened the application.

\*

\* Exit: Returns the width of the application.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function *getBackgroundWidth*()

{

return *background*.width

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Get the background height of the app for all

\* that need access to it.

\*

\* Entry: User has opened the application.

\*

\* Exit: Returns the height of the application.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function *getBackgroundHeight*()

{

return *background*.height

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Get the smallest orientation for scalability

\* purposes, whether it's the portrait or landscape

\* orientation.

\*

\* Entry: User has opened the application.

\*

\* Exit: Returns the smallest orientation of the

\* application.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function *getSmallestOrientation*()

{

return *background*.width < *background*.height ? *background*.width : *background*.height

}

function *isLandscape*()

{

return *getSmallestOrientation*() == *getBackgroundHeight*() ? true : false

}

function *isPortrait*()

{

return !*isLandscape*()

}

}

## CG\_definitions.js

var *FONT\_SIZE* = 0.03;

var *TOP\_COLOR\_FOR\_BUTTON* = "#000000";

var *BOTTOM\_COLOR\_FOR\_BUTTON* = "#66CCFF";

var *BUTTON\_COLOR\_ON\_CLICK* = "yellow";

## CG\_login.qml

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* File Name: CG\_login.qml

\*

\* Overview:

\* This QML file is the presentation layer of our login screen.

\*

\* A user can either login to an existing account or register a new one.

\* The registration info is completely validated with validators for:

\* - Valid username length

\* - Valid password length, and contains 1 uppercase, 1 lowercase and

\* 1 number

\* - Valid email address

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

import QtQuick 2.3

import QtQuick.Controls 1.2

import QtQuick.Controls.Styles 1.2

import QtQuick.Window 2.0

import QtGraphicalEffects 1.0

import QtMultimedia 5.0

import "CG\_definitions.js" as *Definitions*

import "CG\_login.js" as *Login*

Item

{

id: *root*

width: 400

height: 600

signal loggedIn

Audio

{

id: *iPod*

source: "Sounds/successfulLogin.mp3"

autoLoad: true

loops: 79

}

Column

{

anchors.horizontalCenter: *parent*.horizontalCenter

anchors.verticalCenter: *parent*.verticalCenter

spacing: 6

Image

{

source: "images/cg\_logo\_hires\_app.png"

anchors.horizontalCenter: *parent*.horizontalCenter

width: *Login*.getLogoSize()

height: *Login*.getLogoSize()

}

Label

{

id: *txt\_isOpen*

anchors.horizontalCenter: *parent*.horizontalCenter

text: *Validator*.getFeedback()

// Adjusts the font size for scalability

font.pixelSize: *getSmallestOrientation*() \* 0.03

}

TextField

{

id: *tf\_username*

placeholderText: "Chessgames username"

style: *cgTextFieldStyle*

// For testing purposes I inputted StarWars automatically

text: "StarWars"

// Adjusts the font size for scalability

font.pixelSize: *getSmallestOrientation*() \* 0.03

width: *Login*.getControlWidth()

height: *Login*.getControlHeight()

// When the text for the username is changed validate it

onTextChanged:

{

*root*.setUsernameValidator()

}

}

TextField

{

id: *tf\_password*

placeholderText: "Chessgames password"

style: *cgTextFieldStyle*

echoMode: TextInput.Password

// For testing purposes I inputted StarWars automatically

text: "StarWars1"

// Adjusts the font size for scalability

font.pixelSize: *getSmallestOrientation*() \* 0.03

width: *Login*.getControlWidth()

height: *Login*.getControlHeight()

// When the text for the password is changed validate it

onTextChanged:

{

*root*.setPasswordValidator()

}

}

TextField

{

id: *tf\_confirmPassword*

placeholderText: "Confirm password"

style: *cgTextFieldStyle*

echoMode: TextInput.Password

visible: false

// Adjusts the font size for scalability

font.pixelSize: *getSmallestOrientation*() \* 0.03

width: *Login*.getControlWidth()

height: *Login*.getControlHeight()

// When the text for password confirmation is changed validate it

onTextChanged: *root*.setConfirmPasswordValidator()

}

TextField

{

id: *tf\_emailAddress*

placeholderText: "Email Address"

style: *cgTextFieldStyle*

visible: false

// Adjusts the font size for scalability

font.pixelSize: *getSmallestOrientation*() \* 0.03

width: *Login*.getControlWidth()

height: *Login*.getControlHeight()

// When the text for email is changed validate it

onTextChanged: *root*.setEmailValidator()

}

Button

{

id: *btn\_login*

text: "Login"

style: *cgButtonStyle*

onClicked:

{

if (!*User*.logIn(*tf\_username*.text, *tf\_password*.text))

*txt\_isOpen*.text = "Username or password is incorrect."

else

{

// Reset these values for next login.

*txt\_isOpen*.text = ""

*tf\_username*.text = ""

*tf\_password*.text = ""

*tf\_confirmPassword*.text = ""

*tf\_emailAddress*.text = ""

*root*.loggedIn()

*iPod*.play()

}

*tf\_confirmPassword*.visible = false

*tf\_emailAddress*.visible = false

// If the user is logging in reset the text color

*tf\_username*.textColor = "#000000"

*tf\_password*.textColor = "#000000"

}

width: *Login*.getControlWidth()

height: *Login*.getControlHeight()

}

Button

{

id: *btn\_register*

text: "Register"

style: *cgButtonStyle*

onClicked:

{

*txt\_isOpen*.text = ""

// If the user does not exist allow the user to create the user.

if (!*User*.getUser(*tf\_username*.text))

{

if (*tf\_confirmPassword*.visible == true)

{

// Check the validators before adding a new user.

if (*root*.setValidators())

{

if (*User*.addUser(*tf\_username*.text, *tf\_password*.text, *tf\_emailAddress*.text))

*txt\_isOpen*.text = "Successfully created user."

}

}

else

{

/\* The first time the register button is clicked it should

reveal the password confirmation and email address fields.\*/

*tf\_confirmPassword*.visible = true

*tf\_emailAddress*.visible = true

*root*.setValidators()

}

}

else

*txt\_isOpen*.text = "User already exists."

}

width: *Login*.getControlWidth()

height: *Login*.getControlHeight()

}

Component

{

id: *cgTextFieldStyle*

TextFieldStyle

{

background: Rectangle {

color: "#FFFFFF"

border.color: "#448ed3"

smooth: true

radius: 100

}

}

}

Component

{

id: *cgButtonStyle*

ButtonStyle

{

background: Rectangle

{

gradient: Gradient

{

GradientStop { position: 0.0; color: control.pressed ? *Definitions*.BUTTON\_COLOR\_ON\_CLICK : *Definitions*.TOP\_COLOR\_FOR\_BUTTON }//"#b6ee65" : "#76ae25" }

GradientStop { position: 0.5; color: control.pressed ? *Definitions*.BUTTON\_COLOR\_ON\_CLICK : *Definitions*.BOTTOM\_COLOR\_FOR\_BUTTON /\*"#fbdb65"\*/ }//"#76ae25" : "#b6ee65" }

}

border.color: "#1c375b" //"#448ed3"

border.width: 4

smooth: true

radius: 100

}

label: Text

{

font.pixelSize: *getSmallestOrientation*() \* 0.1

renderType: Text.NativeRendering

verticalAlignment: Text.AlignVCenter

horizontalAlignment: Text.AlignHCenter

font.family: *customFont*.name

color: "#000000" //"#CCFFFF"

text: control.text

}

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Ensures the control fields pass all the

\* validators.

\*

\* Entry: User has clicked register.

\*

\* Exit: Returns whether the fields are valid or not.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function *setValidators*()

{

return *setUsernameValidator*() && *setPasswordValidator*() && *setConfirmPasswordValidator*() && *setEmailValidator*()

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Ensures the username is valid when registering.

\*

\* Entry: User has clicked register.

\*

\* Exit: Displays whether the username is valid or not.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function *setUsernameValidator*()

{

var *valid\_username* = true

if (*tf\_confirmPassword*.visible == true)

{

if (*Validator*.checkValidUsername(*tf\_username*.text))

{

*tf\_username*.textColor = "#00AA00"

*tf\_username*.font.bold = false

}

else

{

*valid\_username* = false

*tf\_username*.textColor = "#FF0000"

*tf\_username*.font.bold = true

}

}

return *valid\_username*

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Ensures the password is valid when registering.

\*

\* Entry: User has clicked register.

\*

\* Exit: Displays whether the password is valid or not.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function *setPasswordValidator*()

{

var *valid\_password* = true

if (*tf\_confirmPassword*.visible == true)

{

if (*Validator*.checkValidPassword(*tf\_password*.text))

{

*tf\_password*.textColor = "#00AA00"

*tf\_password*.font.bold = false

}

else

{

*valid\_password* = false

*tf\_password*.textColor = "#FF0000"

*tf\_password*.font.bold = true

}

}

return *valid\_password*

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Ensures the password is the same when registering.

\*

\* Entry: User has clicked register.

\*

\* Exit: Displays whether the confirm password is the same.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function *setConfirmPasswordValidator*()

{

var *valid\_confirm\_password* = true

if (*tf\_confirmPassword*.visible == true)

{

if (*tf\_password*.text == *tf\_confirmPassword*.text)

{

*tf\_confirmPassword*.textColor = "#00AA00"

*tf\_confirmPassword*.font.bold = false

}

else

{

*valid\_confirm\_password* = false

*tf\_confirmPassword*.textColor = "#FF0000"

*tf\_confirmPassword*.font.bold = true

}

}

return *valid\_confirm\_password*

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Ensures the email is valid when registering.

\*

\* Entry: User has clicked register.

\*

\* Exit: Displays whether the email is valid or not.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function *setEmailValidator*()

{

var *valid\_email* = true

if (*tf\_confirmPassword*.visible == true)

{

if (*Validator*.checkValidEmailAddress(*tf\_emailAddress*.text))

{

*tf\_emailAddress*.textColor = "#00AA00"

*tf\_emailAddress*.font.bold = false

}

else

{

*valid\_email* = false

*tf\_emailAddress*.textColor = "#FF0000"

*tf\_emailAddress*.font.bold = true

}

}

return *valid\_email*

}

}

## CG\_login.js

// The logo width and height is 1/2 of the height when in portrait

// and 1/3 of the height when in landscape.

function *getLogoSize*()

{

return *background*.width > *background*.height ? *background*.height \* 0.33 : *background*.height / 2

}

// The individual control height is 1/15 of the height when in portrait

// and (2/3) \* (1/8) = apprx 8.33% in landscape.

function *getControlHeight*()

{

return *background*.width > *background*.height ? *background*.height \* 0.0833 : *background*.height / 15

}

// The individual control width is 90% of the smallest orientation.

function *getControlWidth*()

{

return *getSmallestOrientation*() \* 0.9

}

## CG\_lobby.qml

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* File Name: CG\_lobby.qml

\*

\* Overview:

\* This QML file is the presentation layer of our lobby screen.

\*

\* A user can either select a game time for a

\* - 1 minute game

\* - 5 minute game

\* - 30 minute game

\* Or select the Logout or Settings option.

\*

\* A local clock is displayed next to the user info, which includes a

\* - Username

\* - ELO Rating

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

import QtQuick 2.3

import QtQuick.Controls 1.2

import QtQuick.Controls.Styles 1.1

import QtQuick.Window 2.0

import QtWebKit 1.0

import QtGraphicalEffects 1.0

import QtQuick.Layouts 1.0

import QtMultimedia 5.0

import "CG\_definitions.js" as *Definitions*

Item

{

id: *root*

width: 400

height: 600

signal goBack

signal startOneMinuteGame

signal startFiveMinuteGame

signal startThirtyMinuteGame

signal settings

// The individual control height is 1/8 of the total height.

function *getControlHeight*()

{

return *background*.height / 8

}

// The individual control width is 100% of the smallest orientation.

function *getControlWidth*()

{

return *getSmallestOrientation*() \* 0.9

}

ColumnLayout

{

anchors.horizontalCenter: *parent*.horizontalCenter

anchors.verticalCenter: *parent*.verticalCenter

anchors.fill: *parent*

RowLayout

{

Layout.alignment: *Qt*.AlignTop && *Qt*.AlignHCenter

Layout.preferredWidth: *getControlWidth*()

Layout.preferredHeight: *getBackgroundHeight*() \* 0.33

/\*Clock

{

city: "Klamath Falls"

shift: -8

}\*/

WebView

{

id: *webview*

url: "http://www.chessgames.com/puzzle.html"

width: (*getBackgroundHeight*() \* 0.33) \* 0.66

height: *getBackgroundHeight*() \* 0.33

onNavigationRequested:

{

// detect URL scheme prefix, most likely an external link

var *schemaRE* = /^\w+:/;

if (*schemaRE*.test(request.url))

request.action = WebView.AcceptRequest;

else

{

request.action = WebView.IgnoreRequest;

// delegate request.url here

}

}

}

ColumnLayout

{

Layout.alignment: *Qt*.AlignRight

Text

{

id: *lbl\_username*

text: *root*.visible == true ? *User*.getUsername() : ""

color: "white"

font.family: *customFont*.name

font.bold: true

style: Text.Raised

styleColor: "black"

// Adjusts font size for scalability

font.pixelSize: *getControlWidth*() \* 0.1

}

Text

{

id: *lbl\_eloRating*

text: *root*.visible == true ? ("ELO: " + *User*.getCurrentELO()) : ""

color: "white"

font.family: "Helvetica"

font.bold: true

style: Text.Raised

styleColor: "black"

// Adjusts font size for scalability

font.pixelSize: *getControlWidth*() \* 0.09

}

}

}

Button

{

id: *btn\_oneMinuteGame*

text: "One Minute"

style: *cgButtonStyle*

Layout.alignment: *Qt*.AlignCenter

Layout.preferredWidth: *getControlWidth*()

Layout.preferredHeight: *getControlHeight*()

onClicked:

{

*root*.startOneMinuteGame()

}

}

Button

{

id: *btn\_fiveMinuteGame*

text: "Five Minute"

style: *cgButtonStyle*

Layout.alignment: *Qt*.AlignCenter

Layout.preferredWidth: *getControlWidth*()

Layout.preferredHeight: *getControlHeight*()

onClicked:

{

*root*.startFiveMinuteGame()

}

}

Button

{

id: *btn\_thirtyMinuteGame*

text: "Thirty Minute"

style: *cgButtonStyle*

Layout.alignment: *Qt*.AlignCenter

Layout.preferredWidth: *getControlWidth*()

Layout.preferredHeight: *getControlHeight*()

onClicked:

{

*root*.startThirtyMinuteGame()

}

}

RowLayout

{

Layout.alignment: *Qt*.AlignBottom && *Qt*.AlignHCenter

Button

{

id: *btn\_logout*

text: "Logout"

style: *cgButtonStyle*

Layout.preferredWidth: *getControlWidth*() / 2

Layout.preferredHeight: *getControlHeight*()

onClicked: *root*.goBack()

}

Button

{

id: *btn\_settings*

text: "Settings"

style: *cgButtonStyle*

Layout.preferredWidth: *getControlWidth*() / 2

Layout.preferredHeight: *getControlHeight*()

onClicked: *root*.settings()

}

}

}

Component

{

id: *cgButtonStyle*

ButtonStyle

{

background: Rectangle

{

gradient: Gradient

{

GradientStop { position: 0.0; color: control.pressed ? *Definitions*.BUTTON\_COLOR\_ON\_CLICK : *Definitions*.TOP\_COLOR\_FOR\_BUTTON }

GradientStop { position: 0.5; color: control.pressed ? *Definitions*.BUTTON\_COLOR\_ON\_CLICK : *Definitions*.BOTTOM\_COLOR\_FOR\_BUTTON }

}

border.color: "#448ed3"

smooth: true

radius: 100

}

label: Text {

font.pixelSize: *getSmallestOrientation*() \* 0.1

renderType: Text.NativeRendering

verticalAlignment: Text.AlignVCenter

horizontalAlignment: Text.AlignHCenter

font.family: *customFont*.name

color: "black"

text: control.text

}

}

}

}

## CG\_settings.qml

import QtQuick 2.3

import QtQuick.Controls 1.2

import QtQuick.Controls.Styles 1.1

import QtQuick.Window 2.0

import QtGraphicalEffects 1.0

import QtQuick.Layouts 1.0

import QtMultimedia 5.0

import "CG\_definitions.js" as *Definitions*

Item

{

id: *root*

width: 400

height: 600

signal goBack

//property alias countryFlag: cb\_country.currentIndex

property int countryFlag: 0

signal settingsChanged

function *getSmallestOrientation*()

{

return *background*.width < *background*.height ? *background*.width : *background*.height

}

// The individual control height is 1/8 of the total height.

function *getControlHeight*()

{

return *background*.height / 8

}

// The individual control width is 100% of the smallest orientation.

function *getControlWidth*()

{

return *getSmallestOrientation*() \* 0.9

}

ColumnLayout

{

anchors.horizontalCenter: *parent*.horizontalCenter

Text

{

id: *lbl\_settings*

text: "Settings"

anchors.horizontalCenter: *parent*.horizontalCenter

color: "white"

font.family: *customFont*.name

font.bold: true

style: Text.Raised

styleColor: "black"

font.pixelSize: *getControlWidth*() \* 0.15

}

RowLayout

{

id: *row\_country*

anchors.horizontalCenter: *parent*.horizontalCenter

Text

{

id: *lbl\_country*

text: "Country:"

width: 20

color: "white"

font.family: *customFont*.name

font.bold: true

style: Text.Raised

styleColor: "black"

//Layout.alignment: Qt.AlignCenter

font.pixelSize: *getControlWidth*() \* 0.1

}

ComboBox

{

id: *cb\_country*

editable: true

currentIndex: -1

width: 20

validator: RegExpValidator { regExp: /^[A-Za-z]+$/ }

Layout.alignment: *Qt*.AlignCenter

model: ListModel

{

id: *lm\_country*

ListElement { text: "" }

ListElement { text: "United States" }

ListElement { text: "Austria" }

ListElement { text: "Azerbaijan" }

ListElement { text: "Belarus" }

ListElement { text: "Bulgaria" }

ListElement { text: "Croatia" }

ListElement { text: "Czech Republic" }

ListElement { text: "Denmark" }

ListElement { text: "France" }

ListElement { text: "Germany" }

ListElement { text: "Greece" }

ListElement { text: "Italy" }

ListElement { text: "Kazakstan" }

ListElement { text: "Luxembourg" }

ListElement { text: "Poland" }

ListElement { text: "Portugal" }

ListElement { text: "Romania" }

ListElement { text: "Russia" }

ListElement { text: "Spain" }

ListElement { text: "Sweden" }

ListElement { text: "Switzerland" }

ListElement { text: "Turkey" }

ListElement { text: "Ukraine" }

ListElement { text: "United Kingdom" }

}

}

}

RowLayout

{

anchors.right: *row\_country*.right

Text

{

id: *lbl\_sound*

text: "Sound:"

width: 20

color: "white"

font.family: *customFont*.name

font.bold: true

style: Text.Raised

styleColor: "black"

//Layout.alignment: Qt.AlignCenter

//anchors.right: lbl\_country.right

font.pixelSize: *getControlWidth*() \* 0.1

}

ComboBox

{

id: *cb\_sound*

currentIndex: -1

width: 20

Layout.alignment: *Qt*.AlignCenter

model: ListModel

{

id: *lm\_sound*

ListElement { text: "On"; color: "yellow" /\*american flag\*/ }

ListElement { text: "Off"; color: "Green" /\*English flag\*/ }

}

}

}

}

RowLayout

{

Layout.alignment: *Qt*.AlignBottom && *Qt*.AlignCenter

anchors.horizontalCenter: *parent*.horizontalCenter

anchors.bottom: *parent*.bottom

Button

{

id: *btn\_back*

text: "Back"

style: *cgButtonStyle*

Layout.preferredWidth: *getControlWidth*() / 2

Layout.preferredHeight: *getControlHeight*()

onClicked: *root*.goBack()

}

Button

{

id: *btn\_Save*

text: "Save"

style: *cgButtonStyle*

Layout.preferredWidth: *getControlWidth*() / 2

Layout.preferredHeight: *getControlHeight*()

onClicked:

{

*root*.countryFlag = *cb\_country*.currentIndex - 1

*root*.settingsChanged()

}

}

}

Component

{

id: *cgButtonStyle*

ButtonStyle

{

background: Rectangle

{

gradient: Gradient

{

GradientStop { position: 0.0; color: control.pressed ? *Definitions*.BUTTON\_COLOR\_ON\_CLICK : *Definitions*.TOP\_COLOR\_FOR\_BUTTON }

GradientStop { position: 0.5; color: control.pressed ? *Definitions*.BUTTON\_COLOR\_ON\_CLICK : *Definitions*.BOTTOM\_COLOR\_FOR\_BUTTON }

}

border.color: "#448ed3"

smooth: true

radius: 100

}

label: Text

{

font.pixelSize: *getSmallestOrientation*() \* 0.1

renderType: Text.NativeRendering

verticalAlignment: Text.AlignVCenter

horizontalAlignment: Text.AlignHCenter

font.family: *customFont*.name

color: "black"

text: control.text

}

}

}

}

## CG\_loading.qml

import QtQuick 2.3

import QtQuick.Controls 1.2

import QtQuick.Controls.Styles 1.2

import QtQuick.Window 2.2

import QtQuick.Dialogs 1.2

import QtQuick.Layouts 1.0

import QtMultimedia 5.0

import "CG\_definitions.js" as *Definitions*

Item

{

id: *root*

width: 400

height: 600

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Retrieves the size of the circle, to aid in the

\* dynamically resizing process.

\*

\* Entry: NA

\*

\* Exit: Returns the size of the circle relative to the size

\* of the window.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function *sizeOfCircle*()

{

// 1/7 leaves spacing for the six circles along with the proper

// amount of spacing needed for each circle.

return *getSmallestOrientation*() \* (1/7)

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Retrieves the time it takes for the circles to

\* complete a revolution on the screen.

\*

\* Entry: NA

\*

\* Exit: Returns the duration of a revolution in milliseconds.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function *getDuration*()

{

return 500

}

anchors.horizontalCenter: *parent*.horizontalCenter

signal matched

property alias timer: *timer*

Timer

{

id: *timer*

interval: 3000

running: false

onTriggered:

{

*root*.matched()

*iPod*.play()

}

}

Audio

{

id: *iPod*

source: "Sounds/gameStarted.mp3"

autoLoad: true

}

Row

{

anchors.horizontalCenter: *parent*.horizontalCenter

Repeater

{

model: 5

Rectangle

{

id: *rect*

height: *sizeOfCircle*()

width: *sizeOfCircle*()

radius: *width* \* .5

smooth: true

z: index % 2 == 0 ? -1 : 2

gradient: Gradient

{

GradientStop { position: 0.0; color: *Definitions*.TOP\_COLOR\_FOR\_BUTTON }

GradientStop { position: 0.5; color: *Definitions*.BOTTOM\_COLOR\_FOR\_BUTTON }

}

SequentialAnimation

{

running: true

loops: Animation.Infinite

NumberAnimation

{

target: *rect*

property: "y"

duration: *getDuration*()

to:

{

if (index % 2 == 0)

*getBackgroundHeight*() \* .75

else

*getBackgroundHeight*() \* .25

}

}

NumberAnimation

{

target: *rect*

property: "y"

duration: *getDuration*()

to: (*getBackgroundHeight*() / 2) - (*sizeOfCircle*() / 2)

}

NumberAnimation

{

target: *rect*

property: "y"

duration: *getDuration*()

to:

{

if (index % 2 == 0)

*getBackgroundHeight*() \* .25

else

*getBackgroundHeight*() \* .75

}

}

NumberAnimation

{

target: *rect*

property: "y"

duration: *getDuration*()

to: (*getBackgroundHeight*() / 2) - (*sizeOfCircle*() / 2)

}

}

}

}

}

Text

{

text: "Finding Opponent..."

font.family: *customFont*.name

anchors.horizontalCenter: *parent*.horizontalCenter

y: (*getBackgroundHeight*() / 2) - (*getSmallestOrientation*() \* 0.02)

color: "white"

font.bold: true

style: Text.Raised

styleColor: "white"

z: 1

// Adjusts font size for scalability

font.pixelSize: *getSmallestOrientation*() \* 0.08

}

}

## CG\_game.qml

import QtQuick 2.3

import QtQuick.Controls 1.2

import QtQuick.Controls.Styles 1.1

import QtQuick.Window 2.0

import QtGraphicalEffects 1.0

import QtQuick.Layouts 1.0

import QtMultimedia 5.0

import QtQuick.Dialogs 1.2

import "CG\_board.js" as *Board*

import "CG\_definitions.js" as *Definitions*

import "CG\_game.js" as *Game*

Item

{

id: *root*

width: 400

height: 600

signal finished

property int gameTimeInMinutes: 1

property bool gameOver: false

property alias playerFlagFrame: *cg\_player*.flagFrame

CG\_banner

{

id: *cg\_opponent*

width: *Game*.getBannerWidth()

height: *Game*.getBannerHeight()

playerInfo: "Trudodyr\n2715"

ledActive: *cg\_board*.whiteBlackMove == 1 ? true : false

flagFrame: 21

clockRunning:

{

//If it's black's turn and the game is active.

if (*cg\_board*.whiteBlackMove == 1 && *root*.visible == true)

true

else

false

}

anchors.top: *parent*.top

anchors.left: *parent*.left

}

CG\_banner

{

id: *cg\_player*

width: *Game*.getBannerWidth()

height: *Game*.getBannerHeight()

playerInfo: *root*.visible == true ? (*User*.getUsername() + "\n" + *User*.getCurrentELO()) : ""

ledActive: *cg\_board*.whiteBlackMove == 0 ? true : false

clockRunning:

{

if (*cg\_board*.whiteBlackMove == 0 && *root*.visible == true)

true

else

false

}

anchors.bottom: *isLandscape*() ? *parent*.bottom : undefined

anchors.top: *isPortrait*() ? *cg\_board*.bottom : undefined

anchors.left: *parent*.left

}

// Board / Clocks

CG\_board

{

id: *cg\_board*

width: *Game*.getBoardSize()

height: *Game*.getBoardSize()

//anchors.left: isLandscape() === false ? parent.left : rowPlayerInfo.right

anchors.top: *isPortrait*() ? *cg\_opponent*.bottom : *parent*.top

anchors.horizontalCenter: *parent*.horizontalCenter

}

// Draw / Resign buttons and notation

Button

{

id: *btn\_draw*

style: *cgButtonStyle*

width: *isPortrait*() ? *Game*.getBannerHeight() : *Game*.getBannerHeight() / 3

height: *isPortrait*() ? *Game*.getBannerHeight() : *Game*.getBannerHeight() / 3

x:

{

if (*isLandscape*())

*getBackgroundWidth*() - *width*

else

*width*

}

y:

{

if (*isLandscape*())

0

else

*getBackgroundHeight*() - *height*

}

Image

{

source: *gameOver* == false ? "images/cg\_draw.png" : "images/cg\_leftArrow.png"

width: *parent*.width \* 0.5

height: *parent*.height \* 0.5

anchors.horizontalCenter: *parent*.horizontalCenter

anchors.verticalCenter: *parent*.verticalCenter

}

onClicked:

{

//Flash client screen green with sound effect

//Flash opponent's screen green with sound effect

//Flash for ten seconds or until opponent makes move

//Go backward throughout the game.

if (*cg\_board*.currentMoveNumber - 1 >= 0)

{

*cg\_board*.currentMoveNumber = *Board*.backward(*cg\_board*.listOfMoves, *cg\_board*.currentMoveNumber)

*BoardLogic*.callHistoryBackward()

*Board*.refreshBoard(*cg\_board*.pieces)

*iPod2*.play()

}

else

*resignDialog*.open()

//Make the starting and ending selections invisible when

//traversing the history.

*cg\_board*.startingMove.visible = false

*cg\_board*.endingMove.visible = false

}

}

Text

{

id: *lbl\_notation*

text: *Board*.current(*cg\_board*.listOfMoves, *cg\_board*.currentMoveNumber)

color: "white"

font.family: "Helvetica"

font.bold: true

style: Text.Raised

styleColor: "black"

// Adjusts font size for scalability

font.pixelSize: *getSmallestOrientation*() \* 0.08

x:

{

if (*isLandscape*())

*getBackgroundWidth*() - *lbl\_notation*.width

else

(*getBackgroundWidth*() / 2) - (*lbl\_notation*.width / 2)

}

y:

{

if (*isLandscape*())

(*getBackgroundHeight*() / 2) - (*lbl\_notation*.height / 2)

else

*getBackgroundHeight*() - (*btn\_resign*.height / 2) - (*lbl\_notation*.height / 2)

}

}

Audio

{

id: *iPod*

source: "Sounds/resign.mp3"

autoLoad: true

}

Audio

{

id: *iPod2*

source: "Sounds/wrongMove.mp3"

autoLoad: true

volume: 1.0

}

Button

{

id: *btn\_resign*

style: *cgRedButtonStyle*

width: *isPortrait*() ? *Game*.getBannerHeight() : *Game*.getBannerHeight() / 3

height: *isPortrait*() ? *Game*.getBannerHeight() : *Game*.getBannerHeight() / 3

x:

{

if (*isLandscape*())

*getBackgroundWidth*() - *width*

else

*getBackgroundWidth*() - (*width* \* 2)

}

y: *getBackgroundHeight*() - *height*

Image

{

source: *gameOver* == false ? "images/cg\_resign.png" : "images/cg\_rightArrow.png"

width: *parent*.width \* 0.5

height: *parent*.height \* 0.5

anchors.horizontalCenter: *parent*.horizontalCenter

anchors.verticalCenter: *parent*.verticalCenter

}

onClicked:

{

if(*gameOver* == false)

{

//If someone resigned, the game is over

*gameOver* = true

//Post results

*cg\_player*.result = "0"

*cg\_player*.resultVisible = true

//Post results

*cg\_opponent*.result = "1"

*cg\_opponent*.resultVisible = true

//Sound effect to notify user of resignation

*iPod*.play()

//Notify user game is over and how many ELO points were gained/lost

*resignDialog*.open()

//Stop the clocks from running

*cg\_opponent*.clockRunning = false

*cg\_player*.clockRunning = false

}

else

{

*cg\_board*.currentMoveNumber = *Board*.forward(*cg\_board*.listOfMoves, *cg\_board*.currentMoveNumber)

*BoardLogic*.callHistoryForward()

*Board*.refreshBoard(*cg\_board*.pieces)

*iPod2*.play()

}

}

}

MessageDialog

{

id: *resignDialog*

title: "Game Over"

text: "You resigned against:\n\n" + *cg\_opponent*.playerInfo + "\n\nDo you wish to review the game?"

standardButtons: StandardButton.Yes | StandardButton.No

onYes: *resignDialog*.close()

onNo:

{

//Reset the presentation and business layer

*BoardLogic*.resetBoard()

*Board*.refreshBoard(*cg\_board*.pieces)

*gameOver* = false

*resignDialog*.close()

*root*.finished()

}

}

Component

{

id: *cgButtonStyle*

ButtonStyle

{

background: Rectangle {

gradient: Gradient {

GradientStop { position: 0.0; color: control.pressed ? "#b6ee65" : "#76ae25" }

GradientStop { position: 0.5; color: control.pressed ? "#76ae25" : "#b6ee65" }

}

border.color: "#448ed3"

smooth: true

radius: *width* \* 0.5

}

label: Text {

font.pixelSize: *getSmallestOrientation*() \* 0.04

renderType: Text.NativeRendering

verticalAlignment: Text.AlignVCenter

horizontalAlignment: Text.AlignHCenter

font.family: "Chocolate Dealer"

color: "black"

text: control.text

}

}

}

Component

{

id: *cgButtonStyle2*

ButtonStyle

{

background: Rectangle

{

gradient: Gradient

{

GradientStop { position: 0.0; color: control.pressed ? *Definitions*.BUTTON\_COLOR\_ON\_CLICK : *Definitions*.TOP\_COLOR\_FOR\_BUTTON }//"#b6ee65" : "#76ae25" }

GradientStop { position: 0.5; color: control.pressed ? *Definitions*.BUTTON\_COLOR\_ON\_CLICK : *Definitions*.BOTTOM\_COLOR\_FOR\_BUTTON /\*"#fbdb65"\*/ }//"#76ae25" : "#b6ee65" }

}

border.color: "#1c375b" //"#448ed3"

border.width: 4

smooth: true

radius: *width* \* 0.5

}

label: Text

{

font.pixelSize: 30

renderType: Text.NativeRendering

verticalAlignment: Text.AlignVCenter

horizontalAlignment: Text.AlignHCenter

font.family: *customFont*.name

color: "#000000" //"#CCFFFF"

text: control.text

}

}

}

Component

{

id: *cgRedButtonStyle*

ButtonStyle

{

background: Rectangle {

gradient: Gradient {

GradientStop { position: 0.0; color: control.pressed ? "#AA1111" : "#FF5555" }

GradientStop { position: 0.5; color: control.pressed ? "#FF5555" : "#AA1111" }

}

border.color: "#448ed3"

smooth: true

radius: *width* \* 0.5

}

label: Text {

font.pixelSize: *getSmallestOrientation*() \* 0.04

renderType: Text.NativeRendering

verticalAlignment: Text.AlignVCenter

horizontalAlignment: Text.AlignHCenter

font.family: "Chocolate Dealer"

color: "black"

text: control.text

}

}

}

}

## CG\_game.js

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Returns the appropriate size the chessboard

\* should be.

\*

\* Entry: User has entered a game.

\*

\* Exit: The chessboard is set to the appropriate size.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function *getBoardSize*()

{

if (!*isLandscape*())

return *getBackgroundWidth*()

else

return *getBackgroundHeight*()

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Returns the height of each banner in the game

\* window. (Each player has a banner for info / Clock)

\*

\* Entry: User has entered a game.

\*

\* Exit: Each banner is set to the appropriate height.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function *getBannerHeight*()

{

if (!*isLandscape*())

return (*getBackgroundHeight*() - *getBackgroundWidth*()) / 3

else

return *getBackgroundHeight*() / 2

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Returns the width of each banner in the game

\* window. (Each player has a banner for info / Clock)

\*

\* Entry: User has entered a game.

\*

\* Exit: Each banner is set to the appropriate width.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function *getBannerWidth*()

{

if (!*isLandscape*())

return *getBackgroundWidth*()

else

return (*getBackgroundWidth*() - *getBackgroundHeight*()) / 2

}

function *setCountryFlag*(index)

{

if(*index* === 21)

playersCountryFlag = "images/cg\_flag\_turkey.png"

else

playersCountryFlag = "images/cg\_flag\_unitedStates.png"

}

## CG\_board.qml

import QtQuick 2.3

import QtQuick.Controls 1.2

import QtQuick.Controls.Styles 1.1

import QtQuick.Window 2.0

import QtGraphicalEffects 1.0

import QtQuick.Layouts 1.0

import QtMultimedia 5.0

import "CG\_board.js" as *Board*

Item

{

id: *root*

property bool starting\_position: false

property int starting\_index: -1

property int ending\_index: -1

property int whiteBlackMove: 0

property int starting\_piece: -1

property int plyNumber: 0

property int moveNumber: 0

property int whiteMoveNumber: 0

property int blackMoveNumber: 0

property string currentMove: ""

property int i: 0

property int currentMoveNumber: 0

property var listOfMoves: [{move: ""}]

property alias pieces: *repeaterPieces*

property alias startingMove: *starting*

property alias endingMove: *ending*

Image

{

id: *img\_boardTexture*

source: "images/cg\_board.png"

width: *root*.width

height: *root*.height

}

Audio

{

id: *iPod2*

source: "Sounds/move.mp3"

autoLoad: true

}

Audio

{

id: *iPod*

source: "Sounds/wrongMove.mp3"

autoLoad: true

}

Repeater

{

id: *board*

model: 64

Rectangle

{

id: *rect*

color: *Board*.getWhiteOrBlack(index)

opacity: 0.45

width: *Board*.getSquareSize()

height: *Board*.getSquareSize()

x: *Board*.getX(index)

y: *Board*.getY(index)

}

}

MouseArea

{

anchors.fill: *parent*

onMouseXChanged:

{

*current\_piece*.x = *mouse*.x - (*current\_piece*.width / 2)

if (*starting\_position* && *starting*.x !== ((*mouse*.x - (*mouse*.x % *Board*.getSquareSize())) + *Board*.getBoardOffset()))

{

*repeaterPieces*.itemAt(*starting\_index*).currentFrame = 12

*current\_piece*.visible = true

}

}

onMouseYChanged:

{

*current\_piece*.y = *mouse*.y - (*current\_piece*.height / 2)

if (*starting\_position* && *starting*.y !== ((*mouse*.y - (*mouse*.y % *Board*.getSquareSize())) + *Board*.getBoardOffset()))

{

*repeaterPieces*.itemAt(*starting\_index*).currentFrame = 12

*current\_piece*.visible = true

}

}

onPressed:

{

if (*starting\_position* == false)

{

*starting*.visible = false

*ending*.visible = false

*starting\_position* = true

//Our board has a texture so the board itself is offset

//we need to account for it in our indexes.

*starting\_index* = *Board*.getIndex(*mouse*.x - *Board*.getBoardOffset(), *mouse*.y - *Board*.getBoardOffset())

*starting*.x = *Board*.getX(*starting\_index*)

*starting*.y = *Board*.getY(*starting\_index*)

// This allows us to undo a movement if not validated.

*starting\_piece* = *repeaterPieces*.itemAt(*starting\_index*).currentFrame

*ending*.visible = false

*current\_piece*.frame = *starting\_piece*

}

}

onReleased:

{

if (*starting\_position* == true)

{

*starting*.visible = true

*ending*.visible = true

//Our board has a texture so the board itself is offset

//we need to account for it in our indexes.

*ending\_index* = *Board*.getIndex(*mouse*.x - *Board*.getBoardOffset(), *mouse*.y - *Board*.getBoardOffset())

//Ensure the piece has actually moved.

if (*starting\_index* != *ending\_index*)

{

*ending*.x = *Board*.getX(*ending\_index*)

*ending*.y = *Board*.getY(*ending\_index*)

//Ensure that the player is actually picking up a piece.

if (*BoardLogic*.getSquare(*Board*.getRow(*starting\_index*), *Board*.getColumn(*starting\_index*)) !== "")

{

if (*gameOver* || !*BoardLogic*.move(*Board*.getRow(*starting\_index*), *Board*.getColumn(*starting\_index*), *Board*.getRow(*ending\_index*), *Board*.getColumn(*ending\_index*)))

{

//Play wrong move sound

*iPod2*.play()

}

else

{

//Make sound on successful movement

*iPod*.play()

// Toggle the current player's LED if the movement was made.

*whiteBlackMove* = *whiteBlackMove* == 1 ? 0 : 1

//Increment ply number

*plyNumber*++

//Check whose turn it is

*plyNumber* % 2 != 0 ? *whiteMoveNumber*++ : *blackMoveNumber*++

*plyNumber* % 2 != 0 ? *moveNumber* = *whiteMoveNumber* : *moveNumber* = *blackMoveNumber*

//Convert y to file

var *file* = *Board*.getColumn(*ending\_index*)

//Convert x to rank

var *rank* = *Board*.getRow(*ending\_index*)

//Update current move

*currentMove* = *moveNumber* + *Board*.pieceToString(*current\_piece*.frame) + *Board*.yToFile(*file*) + *Board*.xToRank(*rank*)

//Store current move in list

*listOfMoves*.push({move: *currentMove*})

*currentMoveNumber*++

}

// Update the board's pieces after a movement

*Board*.refreshBoard(*repeaterPieces*)

}

*starting\_position* = false

}

// Set the draggable piece to invisible while not being dragged.

*current\_piece*.visible = false

}

}

}

// Display Pieces

Repeater

{

id: *repeaterPieces*

model: 64

Rectangle

{

width: *Board*.getSquareSize()

height: *Board*.getSquareSize()

//Get the piece from the business layer.

property int currentFrame: *Board*.setPiece(*BoardLogic*.getSquare(*Board*.getRow(index), *Board*.getColumn(index)))

x: *Board*.getX(index)

y: *Board*.getY(index)

color: "transparent"

CG\_piece

{

id: *piece*

frame: currentFrame

source: "images/cg\_pieces.png"

running: false

frameCount: 12

anchors.fill: *parent*

}

}

}

// Show the start of a move by highlighting the square

Rectangle

{

id: *starting*

color: "transparent"

border.color: "#FF5555"

border.width: *getSmallestOrientation*() \* 0.01

opacity: 0.8

width: *Board*.getSquareSize()

height: *Board*.getSquareSize()

visible: false

x: *Board*.getX(50)

y: *Board*.getY(50)

}

// Show the end of a move by highlighting the square

Rectangle

{

id: *ending*

color: "transparent"

border.color: "#5555FF"

border.width: *getSmallestOrientation*() \* 0.01

opacity: 0.8

width: *Board*.getSquareSize()

height: *Board*.getSquareSize()

visible: false

x: *Board*.getX(51)

y: *Board*.getY(51)

}

CG\_piece

{

id: *current\_piece*

visible: false

width: *Board*.getSquareSize() \* 2

height: *Board*.getSquareSize() \* 2

source: "images/cg\_pieces.png"

running: false

frameCount: 12

x: *starting*.x

y: *starting*.y

}

//Hide the movement selections when the screen size is changed.

onWidthChanged:

{

*starting*.visible = false

*ending*.visible = false

}

//Hide the movement selections when the screen size is changed.

onHeightChanged:

{

*starting*.visible = false

*ending*.visible = false

}

}

## CG\_board.js

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Returns the size of each square on the board.

\*

\* Entry: User has entered a game.

\*

\* Exit: Displays the square with the given size of 1/18

\* of the background height.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function *getSquareSize*()

{

return *root*.width / 9

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Returns whether the square is white or black.

\* Used for creating the checker pattern.

\*

\* Entry: User has entered a game.

\*

\* Exit: Displays the square with the correct color.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function *getWhiteOrBlack*(current\_index)

{

var *row* = parseInt(*current\_index* / 8)

if (*row* % 2 == 0)

{

if (*current\_index* % 2)

return "#000000"

else

return "#FFFFFF"

}

else

{

if (*current\_index* % 2)

return "#FFFFFF"

else

return "#000000"

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Returns the x coordinate of the passed index.

\*

\* Entry: User has entered a game.

\*

\* Exit: The x coordinate is determined for the passed

\* index.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function *getX*(current\_index)

{

return (*current\_index* % 8) \* *getSquareSize*() + *getBoardOffset*()

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Returns the y coordinate of the passed index.

\*

\* Entry: User has entered a game.

\*

\* Exit: The y coordinate is determined for the passed

\* index.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function *getY*(current\_index)

{

var *row* = parseInt(*current\_index* / 8)

return *row* \* *getSquareSize*() + *getBoardOffset*()

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Refreshes the pieces on the board.

\*

\* Entry: User has entered a game.

\*

\* Exit: The board's pieces are updated.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function *refreshBoard*(repeaterPieces)

{

for (var *currentPiece* = 0; *currentPiece* < *repeaterPieces*.count; ++*currentPiece*)

*repeaterPieces*.itemAt(*currentPiece*).currentFrame = *setPiece*(*BoardLogic*.getSquare(*getRow*(*currentPiece*), *getColumn*(*currentPiece*)))

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Returns the offset of the board itself.

\* This is the offset from the texture behind

\* to the board pattern.

\*

\* Entry: User has entered a game.

\*

\* Exit: The board has the offset from the top left corner

\* of the CG\_board qml object that exists within

\* CG\_game.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function *getBoardOffset*()

{

return (*img\_boardTexture*.width - (*getSquareSize*() \* 8)) / 2

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Returns the index of a pair of coordinates

\*

\* Entry: User has entered a game.

\*

\* Exit: The index is determined based on the passed

\* coordinates.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function *getIndex*(x, y)

{

return ((*Math*.floor(*y* / *getSquareSize*()) \* 8) + *Math*.floor(*x* / *getSquareSize*())) % 64

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Returns the row of the passed index.

\*

\* Entry: User has entered a game.

\*

\* Exit: The row is determined for the passed index.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function *getRow*(current\_index)

{

return parseInt(*current\_index* / 8)

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Returns the column of the passed index.

\*

\* Entry: User has entered a game.

\*

\* Exit: The column is determined for the passed index.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function *getColumn*(current\_index)

{

return (*current\_index* % 8)

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Returns the correct frame for the pieces

\* tilesheet based on an integer value.

\*

\* Entry: User has moved a piece.

\*

\* Exit: The piece is set to the correct frame of the

\* tilesheet.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function *setPiece*(piece)

{

var *frame* = 0

if (*piece* === "King1")

*frame* = 0

else if (*piece* === "Queen1")

*frame* = 1

else if (*piece* === "Knight1")

*frame* = 2

else if (*piece* === "Bishop1")

*frame* = 3

else if (*piece* === "Rook1")

*frame* = 4

else if (*piece* === "Pawn1")

*frame* = 5

else if (*piece* === "King0")

*frame* = 6

else if (*piece* === "Queen0")

*frame* = 7

else if (*piece* === "Knight0")

*frame* = 8

else if (*piece* === "Bishop0")

*frame* = 9

else if (*piece* === "Rook0")

*frame* = 10

else if (*piece* === "Pawn0")

*frame* = 11

else

*frame* = 12

return *frame*

}

function *getMoveNumber*()

{

return *moveNumber*

}

function *pieceToString*(frame)

{

switch(*frame*)

{

case 0:

return "...K"

case 1:

return "...Q"

case 2:

return "...N"

case 3:

return "...B"

case 4:

return "...R"

case 5:

return "..."

case 6:

return ". K"

case 7:

return ". Q"

case 8:

return ". N"

case 9:

return ". B"

case 10:

return ". R"

case 11:

return ". "

default:

return ""

}

}

function *yToFile*(index)

{

switch(*index*)

{

case 0:

return "a"

case 1:

return "b"

case 2:

return "c"

case 3:

return "d"

case 4:

return "e"

case 5:

return "f"

case 6:

return "g"

case 7:

return "h"

default:

return ""

}

}

function *xToRank*(index)

{

switch(*index*)

{

case 0:

return 8

case 1:

return 7

case 2:

return 6

case 3:

return 5

case 4:

return 4

case 5:

return 3

case 6:

return 2

case 7:

return 1

default:

return ""

}

}

function *forward*(moves, current)

{

//Move current index backwards

return ++*current*

}

function *backward*(moves, current)

{

//Move current index backwards

return --*current*

}

function *current*(moves, current)

{

return *moves*[*current*].move

}

## CG\_piece.qml

import QtQuick 2.0

Item

{

id: *root*

clip: true

property alias running: *timer*.running

property alias source:*image*.source

property alias pieceWidth: *image*.width

property alias pieceHeight: *image*.height

property int frame:0

property int frameCount: 0

// Used for keeping the proportion of the

// width to 1/12 of the width of the tilesheet

// for each piece.

onWidthChanged:

{

*image*.width = *root*.width \* 12

}

// Used for adjusting the height of the tilesheet

// to the full size of the piece height

onHeightChanged:

{

*image*.height = *root*.height

}

Image

{

id: *image*

// Moves the image to the correct position

// to display the current frame.

x: -*root*.width \* *root*.frame

}

// Used for animation through the tilesheet if

// we ever want to use this in the future.

Timer

{

id:*timer*

interval: 200

running: false

repeat: true

onTriggered:

{

*nextFrame*();

}

}

// Used to increment through frames

function *nextFrame*()

{

*root*.frame = ++*root*.frame % *root*.frameCount

}

}

## CG\_banner.qml

import QtQuick 2.3

import QtQuick.Controls 1.2

import QtQuick.Controls.Styles 1.1

import QtQuick.Window 2.0

import QtGraphicalEffects 1.0

import QtQuick.Layouts 1.0

Item

{

id: *root*

property alias playerInfo: *lbl\_player*.text

property bool ledActive: false

property alias clockRunning: *cg\_clock*.running

property alias result: *lbl\_result*.text

property alias resultVisible: *lbl\_result*.visible

property alias flagFrame: *img\_flag*.frame

Row

{

id: *rowPlayerInfo*

//Country flag is displayed

CG\_image

{

id: *img\_flag*

width: *getCountryFlagSize*()

height: *getCountryFlagSize*()

source: "images/countries.png"

frame: 0

//This is the number of country flags available

//on the tilesheet.

frameCount: 24

}

//Player Info (Username, ELO) is displayed

Text

{

id: *lbl\_player*

color: "white"

font.family: "Helvetica"

font.bold: true

style: Text.Raised

styleColor: "black"

anchors.verticalCenter: *img\_flag*.verticalCenter

// Adjusts font size for scalability

font.pixelSize: *getPlayerInfoFontSize*()

}

}

Row

{

id: *rowLEDAndClock*

x: *isLandscape*() ? 0 : *root*.width - *width*

y: *isLandscape*() ? *rowPlayerInfo*.height : (*height* / 2)

Rectangle

{

id: *rect\_LED*

width: *getSmallestOrientation*() \* 0.04

height: *getSmallestOrientation*() \* 0.04

radius: (*getSmallestOrientation*() \* 0.04) \* 0.5

//If the led should be active then light it

//with a green color, or if it's not set it

//to white.

color: *ledActive* ? "#00FF00" : "#FFFFFF"

anchors.verticalCenter: *parent*.verticalCenter

}

CG\_clock

{

id: *cg\_clock*

width: *getSmallestOrientation*() \* 0.2

height: (*getSmallestOrientation*() \* 0.2) \* (172/475)

//Convert the game time to seconds and pass it.

clockTime: *gameTimeInMinutes* \* 60

running: false

anchors.verticalCenter: *parent*.verticalCenter

}

}

//Result of the game (0, 1) is displayed

Text

{

id: *lbl\_result*

visible: false

color: "yellow"

opacity: 0.5

font.family: "Helvetica"

font.bold: true

font.pixelSize: *getResultFontSize*()

anchors.centerIn: *isPortrait*() ? *parent* : undefined

}

onWidthChanged:

{

if (*isLandscape*())

{

*lbl\_result*.y = *rowPlayerInfo*.height + *rowLEDAndClock*.height

*lbl\_result*.x = 0

}

}

onHeightChanged:

{

if (*isLandscape*())

{

*lbl\_result*.y = *rowPlayerInfo*.height + *rowLEDAndClock*.height

*lbl\_result*.x = 0

}

}

//If the device or window is in "portrait orientation"

//then return the full height of the banner so that the

//image which has equal width and height can fill the

//size. Or, if in landscape it'll return 1/4 of the

//size.

function *getCountryFlagSize*()

{

return *isPortrait*() ? *root*.height : *root*.height / 4

}

//This returns 4% of the smallest orientation

//to be used as a font pixel size.

//Default size is 400x600 for the application

//so 400 < 600 => 400 \* 0.04 = 16px font size.

function *getPlayerInfoFontSize*()

{

return *getSmallestOrientation*() \* 0.04

}

//This returns 10% of the smallest orientation

//to be used as a font pixel size for the

//result of the game.

function *getResultFontSize*()

{

return *isPortrait*() ? *root*.height : *root*.height - *rowPlayerInfo*.height - *rowLEDAndClock*.height

}

}

## CG\_clock.qml

import QtQuick 2.0

Item

{

id: *root*

property int clockTime: 60 //In seconds

property alias running: *timer*.running

onWidthChanged:

{

*tensOfMinutes*.width = *root*.width / 5

*onesOfMinutes*.width = *root*.width / 5

*tensOfSeconds*.width = *root*.width / 5

*onesOfSeconds*.width = *root*.width / 5

*colon*.width = *root*.width / 5

}

onHeightChanged:

{

*tensOfMinutes*.height = *root*.height

*onesOfMinutes*.height = *root*.height

*tensOfSeconds*.height = *root*.height

*onesOfSeconds*.height = *root*.height

*colon*.height = *root*.height

}

Row

{

CG\_image

{

id: *tensOfMinutes*

width: 95

height: 172

source: "images/cg\_seven\_segment.png"

frame: parseInt((*clockTime* / 60) / 10)

frameCount: 11

}

CG\_image

{

id: *onesOfMinutes*

width: 89

height: 172

source: "images/cg\_seven\_segment.png"

frame: ((*clockTime* / 60) % 10)

frameCount: 11

}

CG\_image

{

id: *colon*

width: 95

height: 172

source: "images/cg\_seven\_segment.png"

frame: 10

frameCount: 11

}

CG\_image

{

id: *tensOfSeconds*

width: 95

height: 172

source: "images/cg\_seven\_segment.png"

frame: parseInt((*clockTime* % 60) / 10)

frameCount: 11

}

CG\_image

{

id: *onesOfSeconds*

width: 95

height: 172

source: "images/cg\_seven\_segment.png"

frame: ((*clockTime* % 60) % 10)

frameCount: 11

}

}

Timer

{

id: *timer*

interval: 1000

running: false

repeat: true

onTriggered: *clockTime* > 0 ? --*clockTime* : undefined

}

}

## CG\_image.qml

import QtQuick 2.0

Item

{

id: *root*

clip: true

property alias source:*image*.source

property int frame:0

property int frameCount: 0

// Used for keeping the proportion of the

// width to 1 / frameCount of the width of

// the tilesheet for each piece.

onWidthChanged:

{

*image*.width = *root*.width \* *frameCount*

}

// Used for adjusting the height of the tilesheet

// to the full size of the piece height

onHeightChanged:

{

*image*.height = *root*.height

}

Image

{

id: *image*

// Moves the image to the correct position

// to display the current frame.

x: -*root*.width \* *root*.frame

}

}

# Screenshots

Below are screenshots of the Chessgames application.

## Server

This is a multithreaded server that handles incoming connection requests from clients.



## Start state

This is what the application will primarily look like when a user opens up the Chessgames application.



## Username Validation

This is an example of what our login page could look like when a user attempts to register an account with our Chessgames application, and their username does not pass at least one of the validations for a valid username.



## Password Validation

This is an example of what our login page could look like when a user attempts to register with our Chessgames application, and their inputted password does not the specifications of our password validator.



## Email Validation

This is an example of what our login page will look like when a user attempts to register with our Chessgames application, and their email address does not the specifications of our email validator.

****

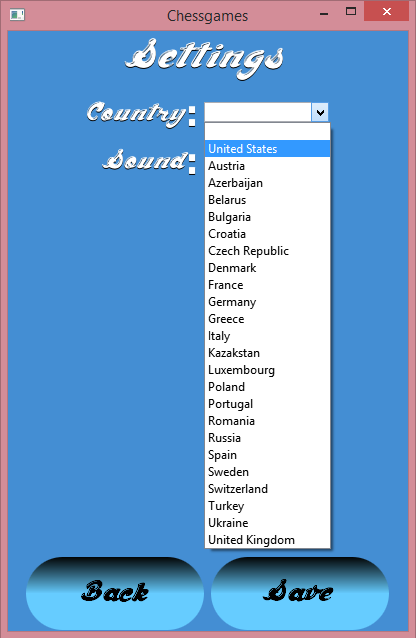
## Lobby

This is our game lobby where our users can select either a 1 minute, 5 minute, or 30 minute game pool.



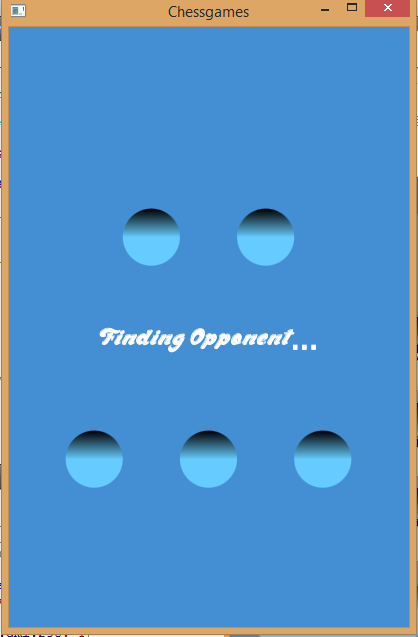
## Settings

This is our settings page where uses can select their country which will update their country flag in game.



## Loading

This is our loading screen when a game is being loaded.



## White player makes move

The following screenshot depicts the white player making a move. The square with the red rectangle border is the source square. The square with the blue rectangle border is the destination square. The notation updates to the last valid move (7. d4) as well as the green LED toggles to signify it is black’s turn.



## Black player makes move

The following screenshot depicts the black player making a move. The source and destination squares denotes the same border colors as the previous example. The notation updates the same way as the previous example, with the exception of an added ellipsis. The ellipsis denotes that black made the last move (7…g4). The green LED toggles, as in the previous example, to signify it is white’s turn.



## QML Scalability

This demonstrates our control and label scalability in our app after making the change to QML for our presentation layer.



# Conclusion

From the design documents, code, user interface and overall functionality of our application we have come a long way. We still have to write a client to interface with the server we implemented early in the term, but overall we are getting closer and closer to having a really fully fleshed out chess application running on Android, iOS, Windows, Mac, Linux. We finished the backlog items we had intended on completing by this point, and have went above/beyond the requirements and even received positive feedback from our product owner and stakeholder. Our goal is to continue to maintain this momentum and finish the product while exceeding expectations and having a stable code base for Chessgames.com to pass on to a future team.