Software Design Document

BubbleChess v1.0

Steve Calabro, Mark Koh, Alex Mann, and Eric Most

2/11/2015

Revision History

Name	Date	Change purpose	Version
Steve Calabro, Mark Koh, Alex Mann, Eric Most	2/11/2015	Initial version	1.0

Table of Contents

1. State the purpose of your project/sub-system

1.1 Purpose

The purpose of this document is to describe the implementation of the BubbleChess software system described in the BubbleChess software Design. BubbleChess is a program designed to bring users together to play a game of chess. The creation of this system will allow players to reach much more diverse competition from their usual opponents. With the ability of the history and user system players will have the ability to learn from past mistakes, keep track of their record as a player, and build a profile as a player across the community. With this consistent information, any average player will get the essential practice and statistics needed to become one of the greats.

1.2 Scope

This document describes the initial design of the BubbleChess software system. While this document provides a baseline design for the system, as requirements change so will the design of the system. This document will not describe the playing and rules of the actual game itself, rather it will outline the structure and design of the system used to represent the game in the virtual environment as well as the GUI objects necessary for creating an enjoyable user experience.

2. High Level Design Overview

2.1 Description of Problem

A common problem that chess players face when looking for an opponent is that they cannot physically meet in person. Additionally, the standard chess game does not keep track of information such as game history and user data. This information can be useful in a player's growth in the game of chess. With BubbleChess these problem no longer exist. Using a simple internet connection users will be able to play against another opponent anywhere in the world. Additionally, they will be able to keep track of game history and statistics.

2.2 Technologies Used

BubbleChess users will interact directly with the Client User Interface, a Java Swing graphical user interface which will interact with the client backend and determine the information needed to send to the Java Web Server. The server will make requests to the Server Processing interface which will get/input information from the SQLite Database through SQL Queries. This information will then be sent back to the webserver and to the respective User through the reverse process.

2.3 System Architecture

Figure 1 depicts the high-level system architecture. The application will consist of many distinct components. These components are listed below:

- User (1 and 2): Consists of two separate players that will be play the game via the application
- Client User Interface: The graphical interface that the user will interact with. It will communicate directly with the client backend
- Client Backend: The backend client code to the client that will handle the game state, pieces, board, and calculate the moves. This will also communicate with the server to send game history, player moves, and login information.
- Java Web Server: The backend server code that will handle the object from the client backend to achieve the login, game history, and player move tasks from one client to another.
- Server Processing: The layer that will communicate with the database directly converting the information from the webserver to valid SQL statements
- Database: The interface for storing and receiving data for the server

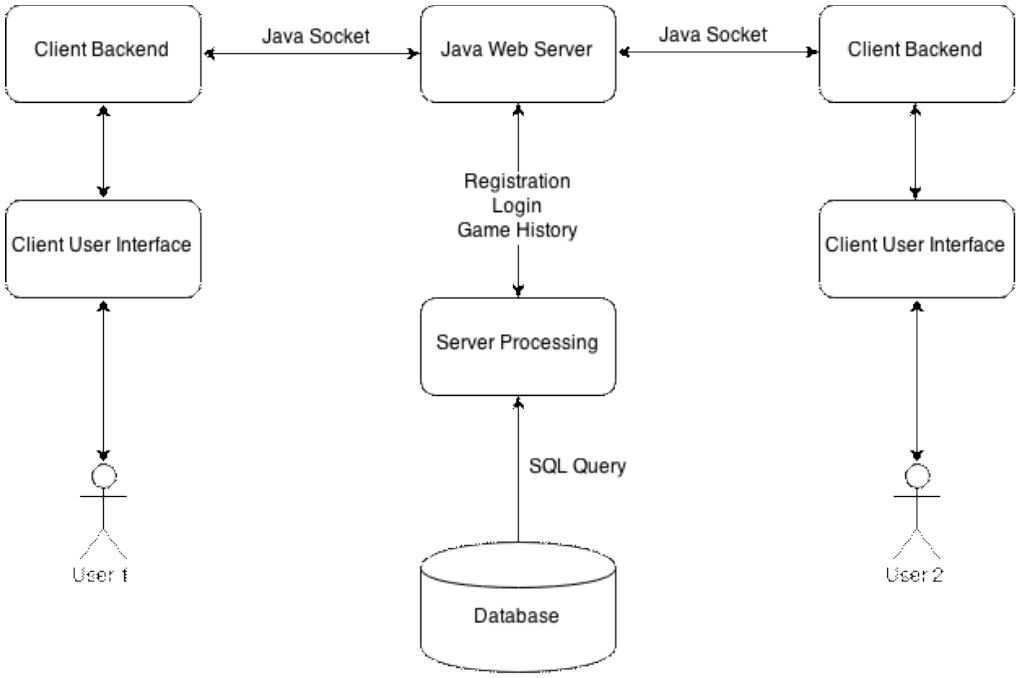

Figure 1: System Architecture Diagram

2.4 System Operation

The following figures and their respective captions describe the primary interactions that take place between the major components of the BubbleChess system during various scenarios.

2.4.1 Register

When the user opens the application they will have the option to create a new account. In this case they will supply a username and password to the client GUI, which will send a createAccount request to the client backed. The client backend will relay this information to the server which will check if the credentials supplied already exist. If the username and password are acceptable the server will create the new account in the user data database and the user will be able to login.

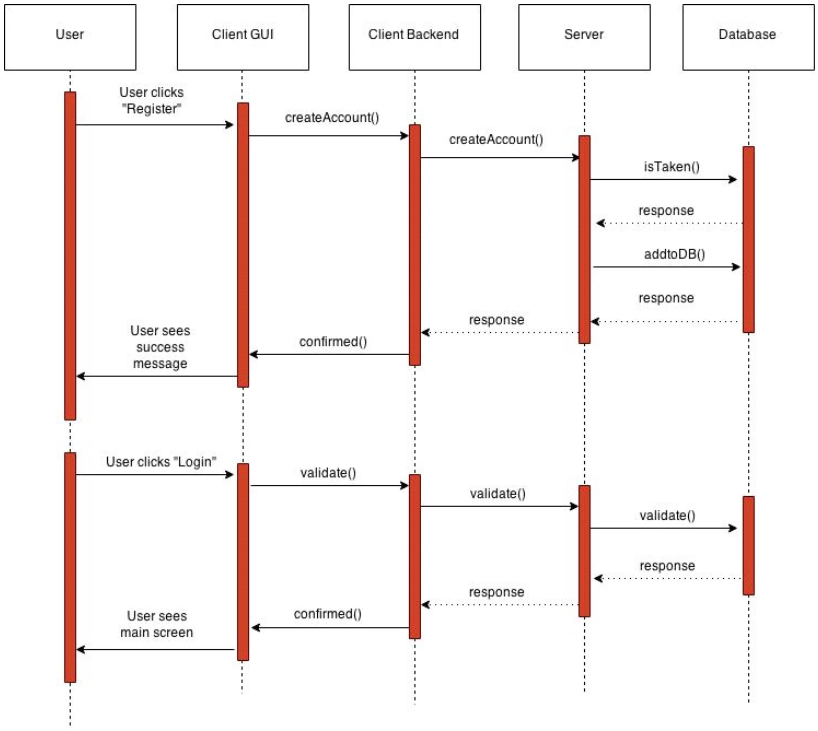

Figure 2: This figure depicts the interactions between components during account registration

2.4.2 Login

When the user opens the application they will be able to provide their username and password to login to their account. The client GUI will send this information to the client backend which will send a validate request to the server. The server will validate the user's password against the account details in the user database. If they passwords match the server will send a success signal to the client, and they main screen will be displayed to the user. If authentication fails, the user will receive a failure message and be able to attempt login again.

These interactions can be seen above in Figure 2.

2.4.3 Game Creation

The user has the option to either join an existing game or create a new one. If the user chooses to create a new game then the client backend will instruct the server to create a new game and log the game information in the game database. A new game will be displayed to the user.

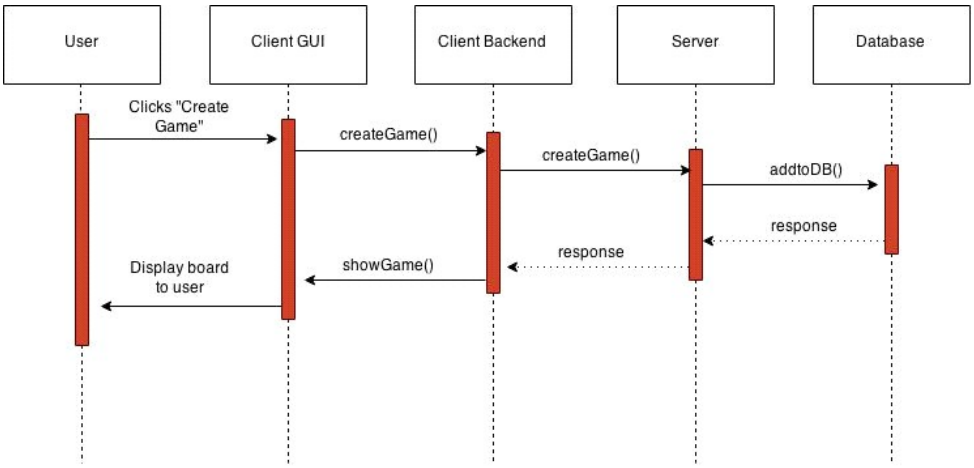

Figure 3: This figure depicts the interactions between components during game creation

2.4.4 Game Play

When the user clicks on a piece, the client GUI will communicate to the client backend with a getMoves request. The client backend will access its game data, calculate the list of possible moves, and send a response to the client GUI. The GUI will highlight possible destination squares. When the user clicks a destination square, the move will be validated on the client backend. If the move is legal it will be sent to the server and added to the game database. When the server receives the move it will update the other user's client and send confirmation to both parties.

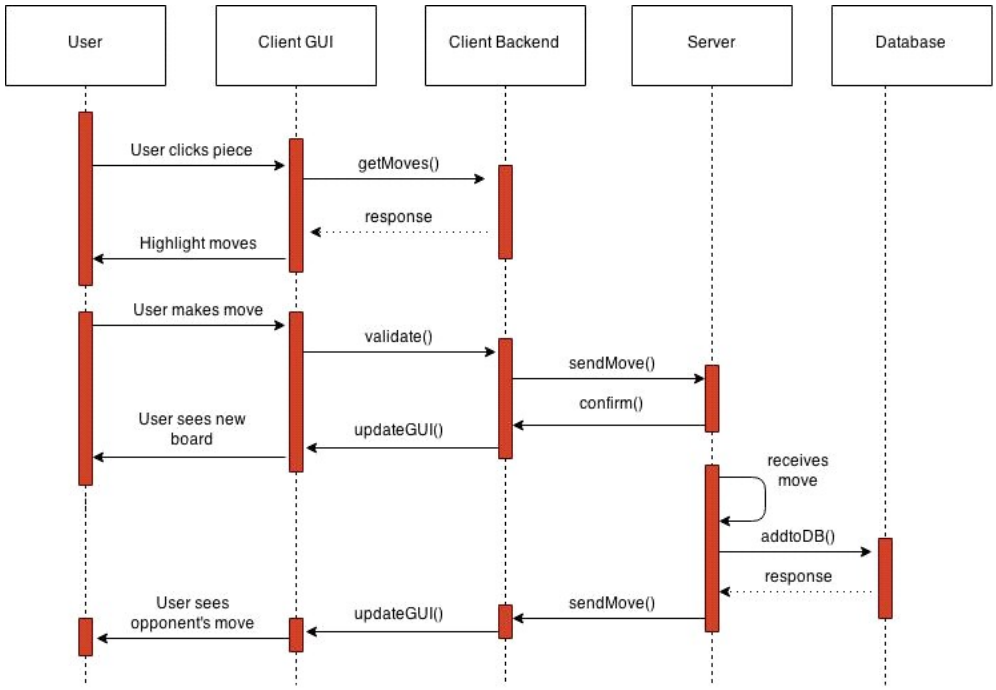

Figure 5: This figure depicts the interactions between components during gameplay

3. Low Level Design Overview

This section is where your objects and object relationships are defined. For each object (or set of objects) define the following:

Usage

Describe in a paragraph how the object is used and what function it serves. If an object will interface with an external object or system, it is a good idea to show the interface for the object. Most importantly, you must again describe your thought process for defining the object as you did. List the benefits and risks. If an object provides an encapsulation, describe in a sentence why the encapsulation adds value. Use your descriptions to give meaning to the diagrams. They don't have to be verbose, just enough to get the point across.

Configuration

If your object needs any special configuration or initialization, this is a good place to describe it. If not, this section can be left out.

Model

Figure 6 shows the depiction of the interaction between the GUI and game classes on the client side. Mainly interacting between the two is the game class which has access and uses the chessboard class where all the the cheese piece classes will be store on their position respectively. This game class will interact with the gameboard gui when a moved is placed and something in the game happens.

Figure 2

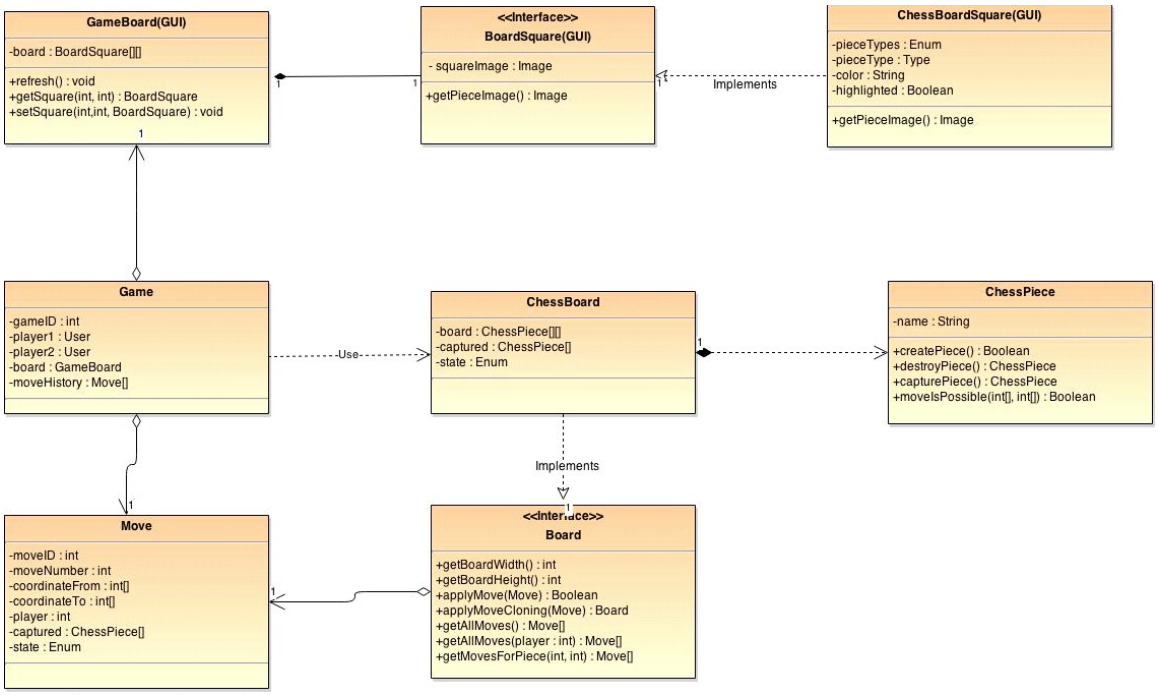

Figure 6: This figure depicts the design for the clientside classes.

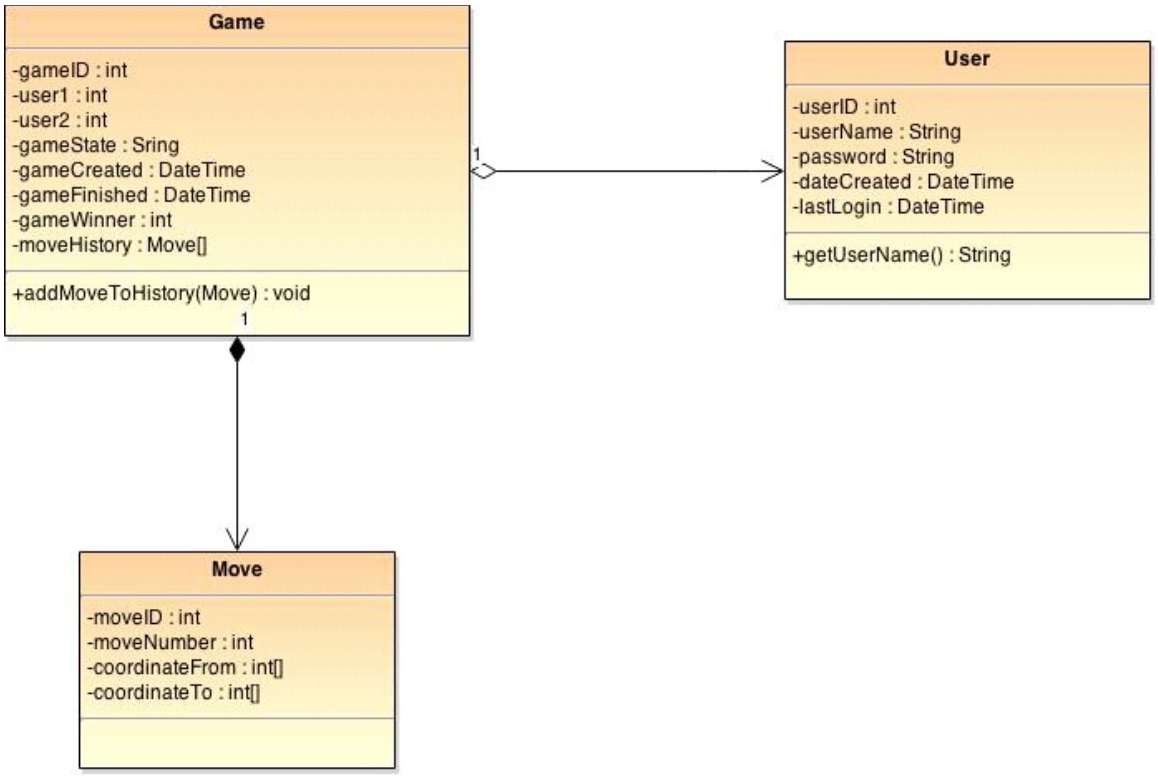

Figure 7: This figure depicts the design for the serverside classes.

The above figure, figure 7, shows the classes that will be used to store things on the server side of the game. This is where the client will be able to get the moves the other player has performed and update their gui representation of the game.

4. Benefits, Assumptions, and Risks

4.1 Benefits

4.1.1 Loose coupling

The design calls for a GUI which has no knowledge of the game being played. The benefit of this design choice is that the client and server sides will have little reliance on each other. It would also be possible to reuse the GUI for other games.

4.1.2 High cohesion

The Game class will encompass two Players, a Board (8x8 matrix of ChessPieces), and individual Pieces. This improves the robustness, reusability, and reliability of the application.

4.1.3 Abstraction

By abstracting the general design of the game on both client and server side, we've enabled major scalability for BubbleChess. For instance, with no migrations needed, the BubbleChess server could easily be used for any other coordinate-based board game with sequential moves. On the client side, by utilizing interfaces we've enabled ourselves to develop other games using similar formats but different rules, board sizes, and playing formats.

4.2 Assumptions

4.2.1 Knowledge of Chess

Our users are generally aware of the rules of chess and how to play chess. These will not be made explicit in the application.

4.2.2 Fault Tolerance

The current software design does not account heavily for fault tolerance. While most of the design is easily scalable to account for system faults, the initial design will assume primarily "sunny-day scenario."

4.3 Risks

4.3.1 Processing Speed

The feature to highlight possible destination squares might slow down the game and take away from the user experience.

4.3.2 Responsibility Delegation

As the software begins to become implemented, there may be a need to refactor the code in such a way to delegate certain responsibilities to more suitable classes.

5. Conclusion

Ultimately, the BubbleChess software system has been designed in a manner that suits the software requirements specification. Not only does the system incorporate many important design concepts necessary for creating a maintainable software system, but the design also allows for scalability. The decisions made in designing the architecture will allow us to easily implement many of the features which we predicted for future releases in the Software Requirements Specification document. The benefits section (4.1) outlines the advantages of many of these decisions.

As the system progresses in development, this document will be updated to reflect the changes to the system design, and it will be used to guide the developers in their implantation of the software system.