Software Design

for

CS451-002 Group 3 Project

Revision 1.1

|  |  |
| --- | --- |
| **Group Members** | Korede Aderele  Natie Bohnel  Samara Painter  Alex Veltchev |
| **Faculty Advisor** | Dr. Filippos Vokolos, Ph. D |

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason for Change** | **Revision** |
| Samara Painter | 1/29/2019 | Initial Document Creation, filled out some sections | 0.8 |
| Natie Bohnel | 2/13/2019 | Added text content to most sections | 0.9 |
| Natie Bohnel | 2/14/2019 | Finalized first draft, added graphics | 1.0 |
| Samara Painter | 2/15/2019 | Added more sections, updated screen flow diagram, minor revisions | 1.1 |

Table of Contents

[1. Introduction 5](#_Toc1115539)

[1.1 Purpose of Document 5](#_Toc1115540)

[1.2 Scope of Document 5](#_Toc1115541)

[1.3 Definitions, Acronyms, and Abbreviations 5](#_Toc1115542)

[2. System Overview 6](#_Toc1115543)

[2.1 Description of Software 6](#_Toc1115544)

[2.2 Technologies Used 6](#_Toc1115545)

[3. System Architecture 7](#_Toc1115546)

[3.1 Architectural Design Components 7](#_Toc1115547)

[3.2 Design Rationale 7](#_Toc1115548)

[4. Component Design 8](#_Toc1115549)

[4.1 Overview 8](#_Toc1115550)

[4.2 Server/Client Communications 8](#_Toc1115551)

[4.2.1 Server Attributes 8](#_Toc1115552)

[4.2.2 Server Methods 8](#_Toc1115553)

[4.3 Client Design 8](#_Toc1115554)

[4.3.1 Client Attributes 10](#_Toc1115555)

[4.3.2 Client Methods 10](#_Toc1115556)

[4.3.3 GUI Attributes 10](#_Toc1115557)

[4.3.4 GUI Methods 10](#_Toc1115558)

[4.3.5 Board Attributes 10](#_Toc1115559)

[4.3.6 Board Methods 10](#_Toc1115560)

[4.3.7 Space Attributes 10](#_Toc1115561)

[4.3.8 Space Methods 10](#_Toc1115562)

[4.3.9 Piece Attributes 11](#_Toc1115563)

[4.3.10 Piece Methods 11](#_Toc1115564)

[4.3.11 Move Attributes 11](#_Toc1115565)

[4.3.12 Move Methods 11](#_Toc1115566)

[5. User Interface Design 12](#_Toc1115567)

[5.1 Overview of User Interface 12](#_Toc1115568)

[5.2 Screen Objects and Actions 12](#_Toc1115569)

[5.3 Client Menu Flow 12](#_Toc1115570)

# 1. Introduction

## 1.1 Purpose of Document

This document describes the implementation of the CS451-002 Group 3 Project software in order to fulfill the requirements specified by the Requirements Specification for the CS451-002 Group 3 Project. This project is a virtual game of checkers played by two people remotely over a network connection.

## 1.2 Scope of Document

This document describes the implementation details of the CS451-002 Group 3 Project software. The software will consist of two separate systems, client and server. Code in the client is intended to be run by the user on their personal computer. Code in the server is intended to be run on an external host that the clients can connect to. This document will not specify the testing of the software, but it will be used to specify the testing of the software in another document.

## 1.3 Definitions, Acronyms, and Abbreviations

Amazon Web Services (AWS) - A suite of web-based products offered by Amazon including cloud-based architecture such as virtual computers and databases.

GUI – Graphical User Interface. The interface users will interact with to play the game.

# 2. System Overview

## 2.1 Description of Software

The CS451-002 Group 3 Project is designed to be a remote game of checkers played over a network by two people. Each player will be able to see all of the possible moves they can make in a turn and select one move to make. The gameplay rules that will be abided by are described at [USACheckers](http://www.usacheckers.com).

## 2.2 Technologies Used

The CS451-002 Group 3 Project will use two remote computers as input devices. These computers will each run a checkers client which will communicate with a remote server. The remote server will be hosted on AWS and will be public to any remote clients attempting to join.

The target platform for the client is any computer which can run a major web browser. The user interface which the client includes will be an applet which will be able to run on any major web browser. Development will be handled in JavaScript (for most of the game logic), node.js (for the server), and HTML (for the interface). Version Control will be handled in GitHub.

# 3. System Architecture

## 3.1 Architectural Design Components

**Networking** – This system allows for a client and server program to communicate with one another. The networking system will function via web sockets and communication between the client and server will use serializable objects.

**Serializable objects** – Game data will be transferred to and from the server using serializable objects. These objects will be able to be interpreted by both the server and client programs.

**Game state** – This system will track the ongoing state of the game on a client.

**Game interface** – This system will allow the user to look at and interact with a virtual checkerboard.

**Game logic** – This system will handle the rules of the game of checkers and communicate with the interface to allow clients to only make valid moves.

**Client** – The client system is responsible for the game itself and will include the game logic, game state, and game interface. A single player will use a single client, so two clients are required in order to play a game of checkers.

**Server** – The server is responsible for connecting two different clients and handles communications during a game. Moves, game ending events (e.g. victory, disconnect), and other miscellaneous game events will be communicated from a client to the server to another client using serializable objects.

## 3.2 Design Rationale

**Why JavaScript and HTML?**

Java graphical interfaces are outdated for both users and developers. By using JavaScript and HTML, the development of the interface should be considerably more straightforward, and users of the system will be able to interact with a GUI which is faster and simpler than one developed in Java (or any C based language).

**Why host a server on AWS?**

By hosting the server on AWS, we ensure that any two clients can connect to the server at any time. AWS resources are generally reliable and independent of any machine used to develop, test, and ultimately use the client. Clients will not depend on any physical server machine in order to connect and play checkers.

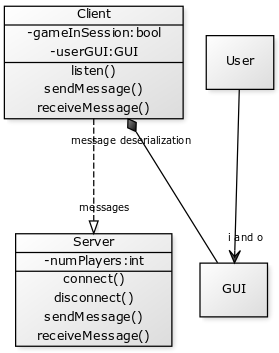
# 4. Component Design

## 4.1 Overview

In this section, more details on the client and server are specified. For both, UML and a brief description are provided.

## 4.2 Server/Client Communications

The server is responsible for sending and receiving messages from two clients and establishing communications between both. Messages will be interpreted by the client before updating the client’s game state.



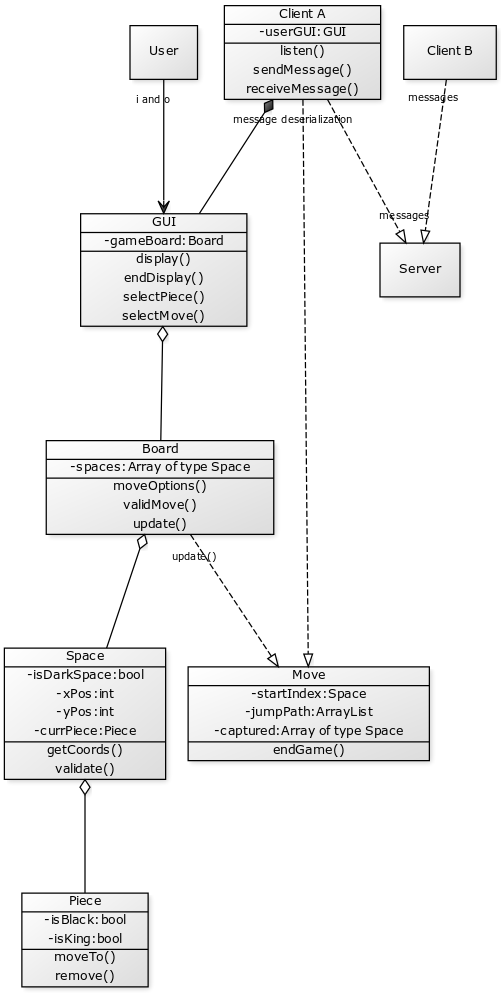
|  |
| --- |
| **Server** |
| // insert Server variables here |
| // insert Server methods here |

### 4.2.1 Server Attributes

### 4.2.2 Server Methods

## 4.3 Client Design

The client is where the game takes place. Through interaction with a GUI, two players will be able to play a game of checkers remotely using two clients. Each move the player makes using the GUI will be sent via a client message and interpreted as a Move in the other client. A Move will update the client’s gameBoard, the data structure responsible for the general board state of the game. Moves can also end the game in each client under specific conditions.



|  |
| --- |
| **Client** |
| userGUI: GUI |
| joinGame(gameId: String), sendMessage(), recieveMessage() |

### 4.3.1 Client Attributes

### The userGUI attribute contains a GUI object responsible for displaying things to the user

### 

### 4.3.2 Client Methods

The joinGame method serves to connect the user to the server which is hosting a specific game. sendMessage and receiveMessage work to manage communication between the clients as mediated by the server, the updates are mainly JSON dumps of the game state.

|  |
| --- |
| **GUI** |
| domNode: DOMNode |
| display(boardState: Array<Space>): DOMNode, selectPiece(coord: String), selectMove(coord: String) |

### 4.3.3 GUI Attributes

domNode represents the DOM Node in the browser that the entire UI is attached to.

### 

### 4.3.4 GUI Methods

The renderBoardState method takes in a boardState Map from a string representing the space index to the space as returned by the Board class and outputs a DOMNode that can be rendered in the browser.

|  |
| --- |
| **Board** |
| spaces: Array< Space> |
| moveOptions(): Array<String>, validMove(from: Space, to: Space): Boolean, update(Array<Space>), render(): DOMNode |

### 4.3.5 Board Attributes

### The space attribute contains an array of all the spaces on the board.

### 4.3.6 Board Methods

The boardState method returns a flat javascript Object/map which maps the board coordinates to the spaces objects. The render method returns a renderable DOM node representing the object, for the Board object this node is a <table> element.

|  |
| --- |
| **Space** |
| xPos: String, yPos: String, currPiece: Piece, isDarkSpace: bool |
| getCoords(): String, validate(): Boolean, render(): DOMNode |

### 4.3.7 Space Attributes

### The xPos and yPos attributes simply contain the row and column (represented as letter) of the space on the board. currPiece specifies the piece that’s currently in the space (if any) and isDarkSpace is simply used to determine the color it is rendered as.

### 4.3.8 Space Methods

getCoords returns the concatenated xPos and yPos strings and validate returns a Boolean indicating whether or not the space is valid on the board. The render method returns a renderable DOM node representing the object, for the Space object this node is a <td> element.

|  |
| --- |
| **Piece** |
| isBlack: Boolean, isKing: Boolean |
| moveTo(Space): Move, remove(), render(): DOMNode |

### 4.3.9 Piece Attributes

isBlack simply determines the color of the piece (black or red) and isKing says whether or not the piece has become a king according to the game rules. . The render method returns a renderable DOM node representing the object, for the Piece object this node is a <p> element.

### 4.3.10 Piece Methods

moveTo repositions the piece in a new space and returns a Move object, while remove method removes a piece from the board. . The render method returns a renderable DOM node representing the object, for the Space object this node is a <td> element.

|  |
| --- |
| **Move** |
| startIndex: Space, jumpPath: Array<Space>, captured: Array<Piece> |
| endGame() |

### 4.3.11 Move Attributes

### startIndex refers to the original space/position of the piece before the move was made while jumpPath refers to the ordered sequence of intermediate and final spaces that the piece is on during the move. Captured contains the objects representing any pieces captured during or as a result of the move.

### 4.3.12 Move Methods

### endGame method terminates the game.

# 5. User Interface Design

## 5.1 Overview of User Interface

The objectives of the User Interface are specified in more detail in the Requirements Specification Document. The interface is meant to be as simple as possible so that even a player with a limited understanding of checkers can enjoy a game with an opponent. Details for the implementation of the interface can be found in the “Technologies Used” section.

## 5.2 Screen Objects and Actions

The server will not be interacted with by any users and will therefore not include any screens for user interaction. Users will input moves in the client using the mouse. The interface which users interact with is meant to be as intuitive as possible – a user will click to select the piece they wish to move, then click the valid space which they wish to move the piece to. Users will also interact with some basic, straightforward menus which will handle the connection and disconnection of the clients.

## 5.3 Client Menu Flow

Below is a graphic of the client’s screen flow. Menus will be as simple as possible, as the focus of the client is the game of checkers. See the Requirements Specification for further details.

