Software Design

for

CS451-002 Group 3 Project

Revision 1.2

|  |  |
| --- | --- |
| **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 |
| Samara Painter | 2/17/2019 | Finalized client class descriptions with specifications from Korede | 1.2 |

Table of Contents

[1. Introduction 4](#_Toc1327841)

[1.1 Purpose of Document 4](#_Toc1327842)

[1.2 Scope of Document 4](#_Toc1327843)

[1.3 Definitions, Acronyms, and Abbreviations 4](#_Toc1327844)

[2. System Overview 5](#_Toc1327845)

[2.1 Description of Software 5](#_Toc1327846)

[2.2 Technologies Used 5](#_Toc1327847)

[3. System Architecture 6](#_Toc1327848)

[3.1 Architectural Design Components 6](#_Toc1327849)

[3.2 Design Rationale 6](#_Toc1327850)

[4. Component Design 7](#_Toc1327851)

[4.1 Overview 7](#_Toc1327852)

[4.2 Server/Client Communications 7](#_Toc1327853)

[4.2.1 Server Attributes 7](#_Toc1327854)

[4.2.2 Server Methods 7](#_Toc1327855)

[4.3 Client Design 7](#_Toc1327856)

[4.3.1 Client Attributes 9](#_Toc1327857)

[4.3.2 Client Methods 9](#_Toc1327858)

[4.3.3 GUI Attributes 10](#_Toc1327859)

[4.3.4 GUI Methods 10](#_Toc1327860)

[4.3.5 Board Attributes 10](#_Toc1327861)

[4.3.6 Board Methods 11](#_Toc1327862)

[4.3.7 Space Attributes 12](#_Toc1327863)

[4.3.8 Space Methods 12](#_Toc1327864)

[4.3.9 Piece Attributes 13](#_Toc1327865)

[4.3.10 Piece Methods 13](#_Toc1327866)

[4.3.11 Move Attributes 14](#_Toc1327867)

[4.3.12 Move Methods 14](#_Toc1327868)

[5. User Interface Design 15](#_Toc1327869)

[5.1 Overview of User Interface 15](#_Toc1327870)

[5.2 Screen Objects and Actions 15](#_Toc1327871)

[5.3 Client Menu Flow 15](#_Toc1327872)

# 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.

DOMNode – An HTML element that can be displayed by a browser

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.

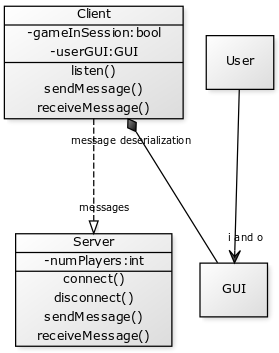
# 4. Component Design

## 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.



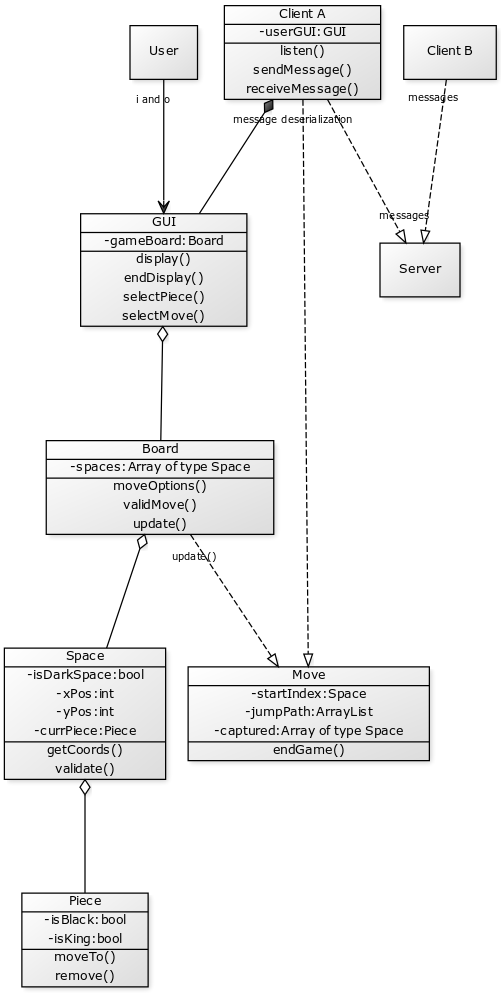
|  |
| --- |
| **Server** |
| // insert Server variables here |
| // insert Server methods here |

### 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) : Void  sendMessage() : Void  receiveMessage() : Void |

### 4.3.1 Client Attributes

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| userGUI | GUI | A GUI object responsible for displaying the game to the user |

### 4.3.2 Client Methods

|  |  |
| --- | --- |
| Void joinGame(String gameId) |  |
| Input: | String containing the ID of the game to join |
| Output: | Void |
| Description | Connects user to the server hosting the game specified by gameID if it is available to connect to |

|  |  |
| --- | --- |
| Void sendMessage() |  |
| Input: | Void |
| Output: | Void |
| Description | Sends a JSON message to the server with an update of the game state |

|  |  |
| --- | --- |
| Void receiveMessage() |  |
| Input: | Void |
| Output: | Void |
| Description | Receives and parses a JSON message from the server with an update of the game state |

|  |
| --- |
| **GUI** |
| domNode: DOMNode |
| display(boardState: Space[]): DOMNode selectPiece(coord: String) : Void  selectMove(coord: String) : Void |

### 4.3.3 GUI Attributes

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| domNode | DOMNode | Represents the DOM Node in the browser that the entire UI is attached to |

### 4.3.4 GUI Methods

|  |  |
| --- | --- |
| DOMNode display(Space[] boardState) |  |
| Input: | A representation of the current board state, including all spaces and positions of all game pieces on those spaces |
| Output: | A DOMNode that can be rendered in the browser |
| Description | Creates a display of the checkers game board that can be rendered in the user’s browser |

|  |  |
| --- | --- |
| Void selectPiece(String coord) |  |
| Input: | A string representation of the space the piece selected by the user is on |
| Output: | Void |
| Description | Forwards the user’s piece selection to the Board class |

|  |  |
| --- | --- |
| Void selectMove(String coord) |  |
| Input: | A string representation of the space the piece selected by the user will move to |
| Output: | Void |
| Description | Forwards the user’s move selection to the Board class |

|  |
| --- |
| **Board** |
| spaces: Space[] |
| moveOptions(space: Space): String[]  validMove(from: Space, to: Space): Boolean update(newBoard: Space[]): Void  render(): DOMNode |

### 4.3.5 Board Attributes

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| spaces | Space[] | All the spaces on the game board in their most current state |

### 4.3.6 Board Methods

|  |  |
| --- | --- |
| String[] moveOptions(Space space) |  |
| Input: | The space containing the piece to be moved |
| Output: | A coordinate representation of all the spaces the piece can make a valid move to |
| Description | Highlights all the spaces that are valid moves for the given piece, returns an error if there is no piece on the given space |

|  |  |
| --- | --- |
| Booleanean validMove(Space from, Space to) |  |
| Input: | A space a piece is moving from and the space the piece wants to move to |
| Output: | True if the piece can legally move to that space, false if it cannot |
| Description | Determines whether a move from one space to another is valid for the piece on the *from* space |

|  |  |
| --- | --- |
| Void update(Space[] newBoard) |  |
| Input: | A representation of all the spaces on the board in an updated state not currently reflected by the user’s current board |
| Output: | Void |
| Description | Updates spaces to reflect the game board in its most current state |

|  |  |
| --- | --- |
| DOMNode render() |  |
| Input: | Void |
| Output: | A DOMNode that can be rendered in the browser |
| Description | Creates a display that represents the board object, specifically, a <table> element |

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

### 4.3.7 Space Attributes

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| xPos | String | A letter representation of the row of the space on the game board |
| yPos | String | A letter representation of the column of the space on the game board |
| currPiece | Piece | The piece that’s currently on the space, or null if the piece is empty |
| isDarkSpace | Boolean | Determines the color of the space, needed for rendering the display of the board |
| isHighlighted | Boolean | True if the space should be highlighted as a legal move for the player |

### 4.3.8 Space Methods

|  |  |
| --- | --- |
| String getCoords() |  |
| Input: | Void |
| Output: | A string representation of the space’s coordinates |
| Description | Concatenates the string xPos and yPos of a space |

|  |  |
| --- | --- |
| Boolean validate() |  |
| Input: | Void |
| Output: | True if the space is valid on the board, false otherwise |
| Description | Determines whether or not a space is valid |

|  |  |
| --- | --- |
| DOMNode render() |  |
| Input: | Void |
| Output: | A DOMNode that can be rendered in the browser |
| Description | Creates a display that represents the space object, specifically, a <td> element |

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

### 4.3.9 Piece Attributes

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| isBlack | Boolean | Determines the color of the game piece – true if black, false if red |
| isKing | Boolean | Determines whether or not a given piece has been kinged over the course of the game |

### 4.3.10 Piece Methods

|  |  |
| --- | --- |
| Move moveTo(Space space) |  |
| Input: | The space to move the piece to |
| Output: | A move representing the piece’s path from its starting space to its ending space |
| Description | Moves a piece from one space to another |

|  |  |
| --- | --- |
| Void remove() |  |
| Input: | Void |
| Output: | Void |
| Description | Removes the piece from the board |

|  |  |
| --- | --- |
| DOMNode render() |  |
| Input: | Void |
| Output: | A DOMNode that can be rendered in the browser |
| Description | Creates a display that represents the piece object, specifically, a <p> element |

|  |
| --- |
| **Move** |
| startIndex: Space  jumpPath: Space[]  captured: Piece[] |
| endGame() : Void |

### 4.3.11 Move Attributes

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| startIndex | Space | The original position of the piece before the move was completed |
| jumpPath | Space[] | The ordered sequence of intermediate space and the final space that the piece is on during the move |
| captured | Piece[] | Objects representing any pieces captured during this move |

### 4.3.12 Move Methods

|  |  |
| --- | --- |
| Void endGame() |  |
| Input: | Void |
| Output: | Void |
| Description | 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.

