Checkers

Software Design Document

Version 1.2

March 04, 2018

Group Members	Chetan K Parakh, Cody Moser, Nicholas Nugent, Saugat Sthapit
Submitted To	Dr. Filippos Vokolos, Ph. D.

Table of Contents

1. Introduction

1.1. Purpose of the Document

This document serves to describe the implementation of Checkers game as described in the Checkers Requirement document. Checkers is designed to be a two player game that can be played over the internet.

1.2. Scope of the Document

This document describes the implementation details of the Checker Application. This application in mainly divided into two components : The client side and the server side. Code in the client side is intended to be run on the client machine (web browser). Code in the server side is intended to be run on the server machine (computer).

1.3. Definitions, Acronyms, Abbreviations

1.3.1 Board

The game is played on the board. Checker board has checker pieces initially stacked on top of the darker color of the board.

1.3.2 Move

Move refers to changing the position of a checker piece in the board.

1.3.3 Checker Piece

Each player starts with 12 piece. One player has lighter colored pieces and the other player has darker pieces.

1.3.4. King [1]

When a player's piece lands in one of the squares at the far end of the board, its move ends there and it becomes a king.

2. System Overview

2.1. Description of Software

Checkers is designed to be a two player online game. Each player will have to compete to capture the opponent's piece. The player to have more than the opponent's piece remain in the game wins. Player will be able to make a valid move to their piece within the time limit during their turn. Player can also choose to view the game rule during the match.

2.2. Technologies Used

Checkers will use mouse for user input. Latest version of chrome, edge and safari (as on first day of 2018) will be supported. The players will be connected through a node.js server hosted on Amazon Web Service. Github will be used for version control and collaboration. Mocha will be used for testing of the application. For the smooth functioning of the application and to get the application status and crash report, Sentry will be used. It will also be used for monitoring and fixing/ tracking errors in real time.

3. System Architecture

3.1. Architectural Design Components

Connection State Manager

The connection state manager is a small component which is responsible for keeping track of the state of players connections. The component will keep track of how many players have connected to the server and are looking to play a game. The connection state manager will only allow 2 players to look for a game at once and will propagate the information about players connections to the

server's game state. Additionally the connection state manager will propagate information about players who have disconnected as well.

Game State

The game state is a component which is shared across the client and server. The game state holds information about the current state of the game, no game, waiting, in progress, or over, as well information about the pieces and their locations on the board and which players turn it is. The game state must be updated each time a player makes a move and the new game state should be distributed to the clients.

Client

The client component is responsible for handling most of the users interaction with the game and the user interface. The client component receives or queries the server for information about the status of connections, the game, and the board. The client component will update the user interface to display the start screen, waiting screen, game screen, and game over screen. The client component also handles accepting moves from players, validating moves, and sending moves to the server.

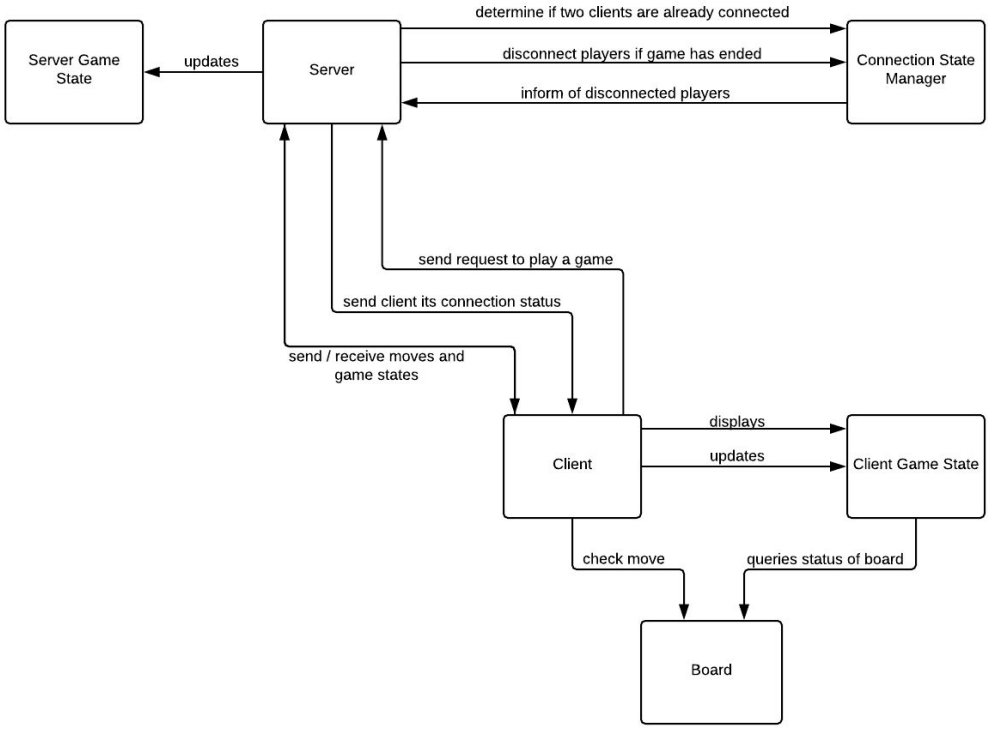
Server

The server component handles integrating the communication between the clients and the other components in the game. The server will manage the connection state and propagate changes of the state to the clients, such as when players connect, disconnect, or are ready to start a game. The server will also accept moves from clients and update the game state and board and propagate or allow these states to be queried by clients. The server must provide endpoints which facilitate communication between the clients, the server, and all components.

Board

The board component is responsible for keeping track of the current state of the checkerboard. This includes how many pieces are on the board, what pieces are on the board, and where pieces on the board are located. The board state will be updated after successful moves by clients and propagated to other clients through the server.

3.2 High Level Architecture Design



3.3 Design Rationale

Why a web application over a downloadable one?

We decided that a web application would allow users to connect and play through a web browser easily. As it is a simple game of checkers, it would not make sense to have the user download an entire application to their computer in order to play. Moreover, in case of any changes made to checkers game, the user does not have to download and install the new update of our application. Refreshing the web page will suffice.

Why not allow a user to play a game locally or against a "computer"?

The requirements for this assignment were to allow two people to play a game of checkers against each other remotely. Those other playing options are unnecessary, but would hopefully be easily added with our design.

Why javascript instead of any other server-side language?

Node.js provides easy way to connect and communicate with web browser. Moreover, the light weight requirements of checkers game leaves us with space to use pre existing node modules to improve and add more features to our game when required.

4. Component Design

4.1 Overview

In this section, more details on each component are given. For each of the component, UML and a brief description is given.

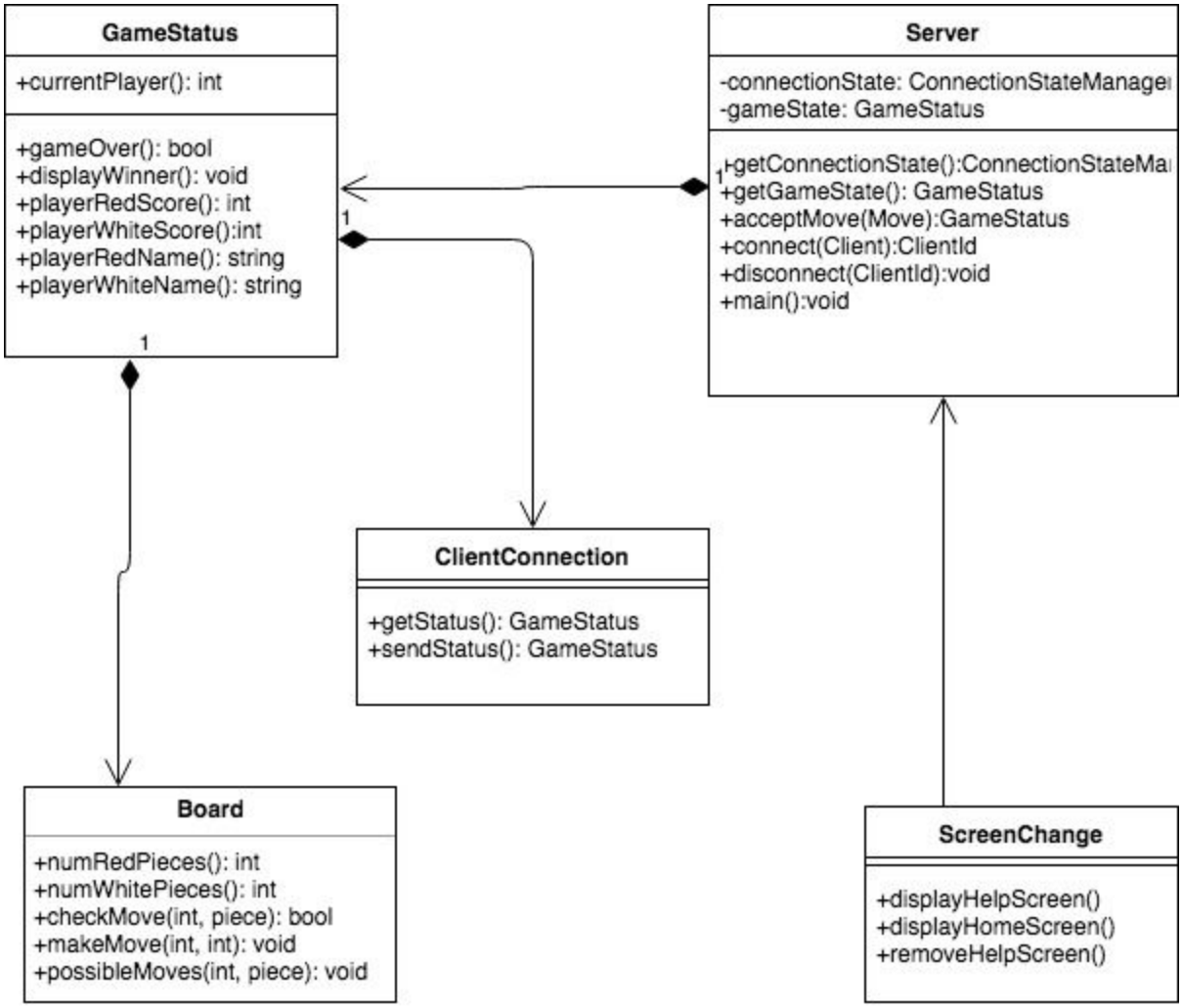


Figure 4.1

4.2 Game

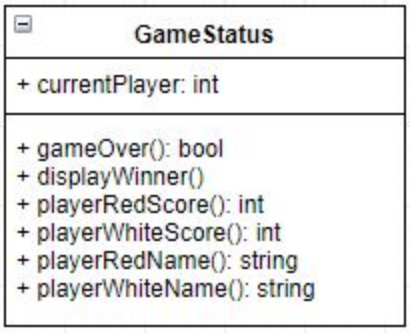


Figure 4.2

4.2.1. Functions

Name	Output	Description
gameOver()	bool	Return if any other move can be made on the board
displayWinner()	-	Displays name of the winner on the screen after game over.
playerRedScore()	int	Returns number of white pieces missing from the board (i.e score of red)
playerWhiteScore()	int	Returns number of red pieces missing from the board (i.e score of red)
playerRedName()	String	Name of player with red checker piece
playerWhiteName()	String	Name of player with white checker piece

4.3 Board

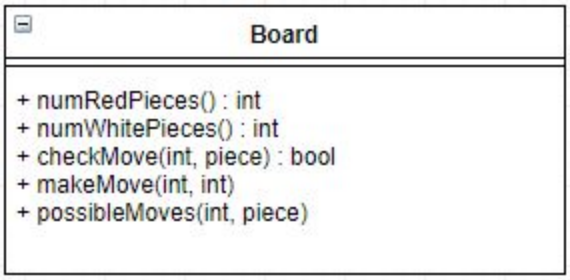


Figure 4.3

4.3.1 Functions

Name	Output	Description
numRedPieces()	int	Returns the number of red checker pieces in the board
numWhitePieces()	int	Returns the number of white checker pieces in the board
checkMove()	bool	Returns if a move is valid or not
makeMove()	-	Changes the state of board by moving a checker piece if checkMove() returns true
possibleMoves()	-	Highlights all the box in which a checker piece can be moved.

4.4 Server

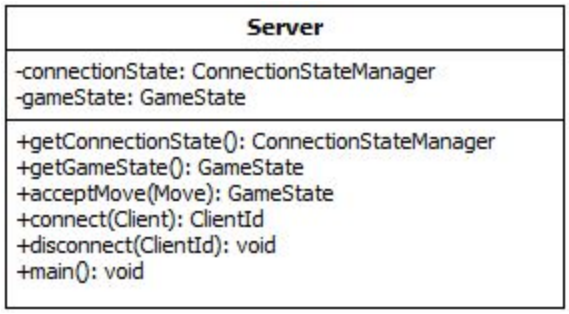


Figure 4.4

4.4.1. Attributes

Name	Type	Description
connectionState	ConnectionStateManager	Holds the current status of player connections
gameState	GameState	Holds the current status of the game

4.4.2. Methods

Function Name	Inputs	Outputs	Description
getConnectionState	void	ConnectionStateManager	Endpoint to query current connection state
getGameState	void	GameState	Endpoint to query current game state
acceptMove	Move	GameState	Endpoint to accept moves from a client

			and return an updated gameState
connect	Client	ClientId	Endpoint to allow clients to connect to game server and receive a unique identifier
disconnect	ClientId	void	Endpoint for disconnecting clients to update server
main	void	void	Main entry point for server to setup default states for components

4.5. Client Connection

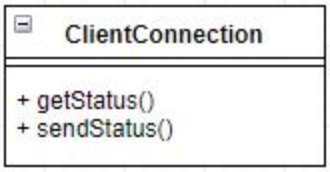


Figure 4.5

4.5.1. Functions

Name	Output	Description
getStatus()	-	Get current game status from the server and makes changes to board and/or game
sendStats()	-	Sends current game status to the server.

4.6 Screen Change

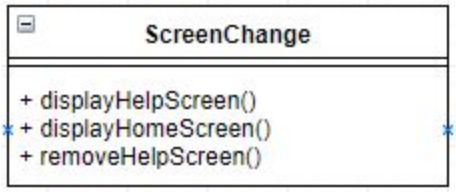


Figure 4.6

4.6.1 Functions

Name	Output	Description
+ displayHelpScreen()	-	Displays help screen
+ displayHomeScreen()	-	Exits the game and displays the home page.
+ removeHelpScreen()	-	Removes help instruction from the screen

5. User Interface Design

This section includes the overview of the user interface design component. Till now, there are no changes in the design. We will be using Socket.io for real-time events which would run on top of NodeJs server. Basic web tools and frameworks like Bootstrap, CSS, Html will be used for page designs. More detail can be found on the requirements document v0.01.

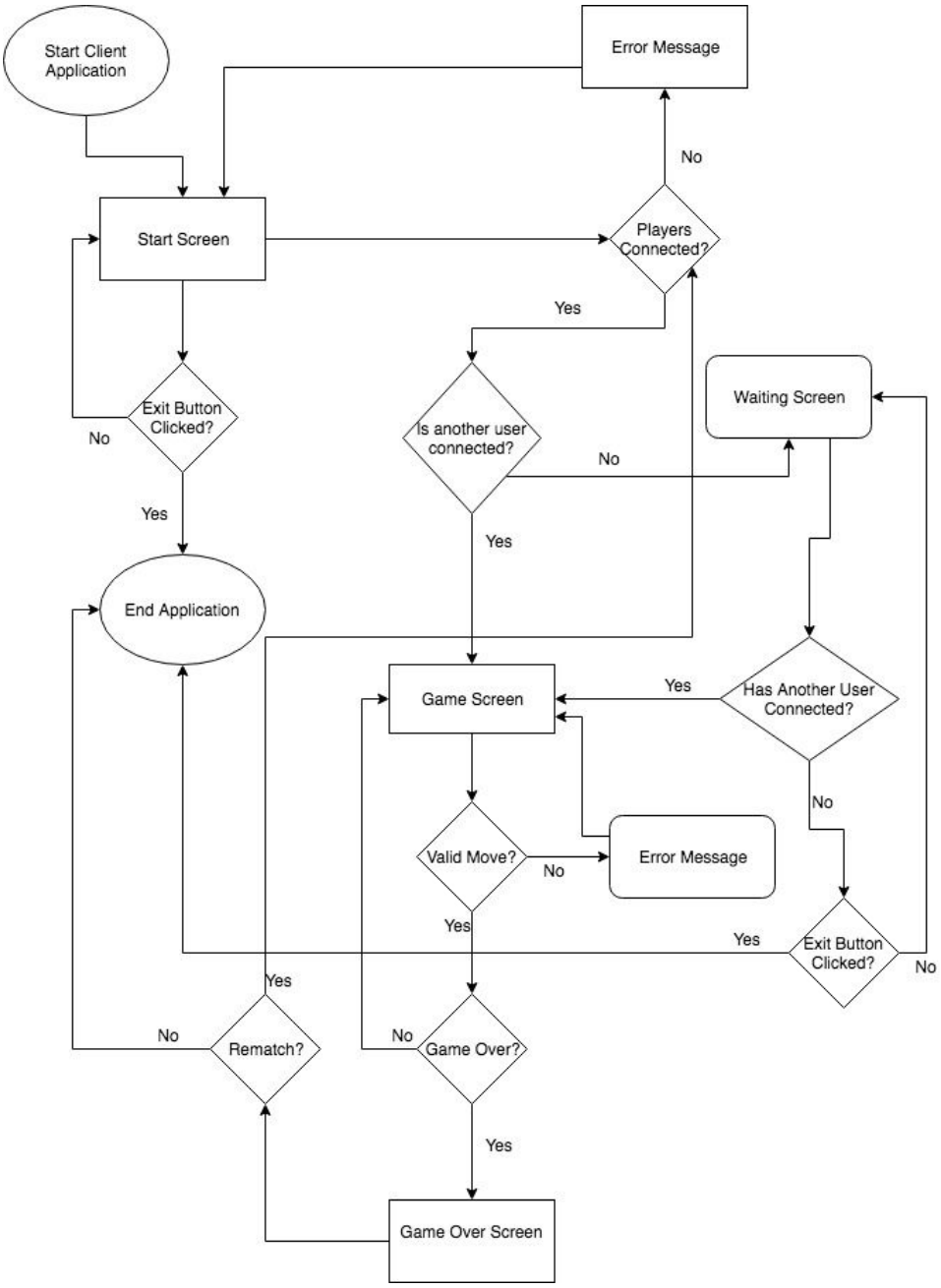


Figure 5.0

6. References

[1]. http://www.darkfish.com/checkers/rules.html