SOFTWARE REQUIREMENTS SPECIFICATION

for

Checkers

Version 2.0.0 approved

Prepared by:
John Zlotek
Matt Horger
Jake Carfagno
Preet Patel

Team: Big Chungus

July 22, 2019

Contents

Revision History

Name	Date	Reason For Changes	Version
1.0.0	11–07–19	Initial Structure	mhorger, jzlotek
1.0.1	17–07–19	More Structure	mhorger, jzlotek
1.1.0	18–07–19	User Interface	jcarfagno
1.1.1	20–07–19	Introduction Information	ppatel
2.0.0	22–07–19	Final Submission	mhorger

1 Introduction

1.1 Purpose of Document

The purpose of the requirement specification is to outline the state-of-the-art web-based Checker game. This game will allow multiple players to interact with each other through a client-server-client manner, it allows them to play and chat. This document covers the scope, objective, basic requirement and goal for this new game. Apart from that this document also covers a topic in- depth like functional, non-functional, user interface design, test cases, program usage, and references. The intention here to make end-user or an engineer understand the design and implantation of this game.

1.2 Project Scope

The purpose of this document is to educate the reader thoroughly about this game, how it works, what's been used, and to lay out the requirements of the program itself.

1.3 Overview of Document

This document contains all general explanation about technology, software work and why they are implemented. Here the document also outlines and describe the specific components of this project. As far as sequence goes, it starts with functional requirement which goes over in two major parts – Server and Client, followed by non-functional requirement which covers networking components and system requirements, then it goes over front end of the project like user interface which goes over framework, menus, and standard components, it also has outlined program usage which includes lobbies, gameplay and wining conditions and finally document ending with references.

1.4 Background

1.4.1 History

There have been many instances in history where the concept of checkers has been found. The earliest was as far as 3000 B.C in ancient Ira q and then in 1400 B.C in ancient Egypt using a 5X5 board [2]. The game we know today was developed in the mid 1500s as an English mathematician developed the rules on what was known as Draughts in 1756, adapting many of the similar mechanics from a Frenchman 400 years earlier. Over time, this game developed to be one of the most popular classic board games of all time.

1.4.2 Game Rules

These rules are adapted from the American Checker Federation [1].

1. Red always gets to play first.
2. A player can forfeit at any time, conceding victory to the other player.

Moves

1. A player may only move their own pieces.
2. Normal Piece

   A normal piece may only move toward the other player's side of the board.

   A normal piece may move diagonally to the left or right to a vacant square in front of it.

   A normal piece may capture on the diagonal if there exists a vacant square one more diagonal position ahead.

   A piece may move again if there exists another piece to capture after making a capture.
3. A King Piece moves the same as a normal piece but can move and capture backward.
4. A King Piece may capture forward or backward.
5. If a normal piece reaches the opposite edge of the board, it becomes a King Piece.

Win Condition

1. When one player has no more pieces to move, the other player is the winner.
2. If a player forfeits the match, the other player is conceded the winner.

1.5 Abstract

We believe that this document will serve any team member currently or in the future enough requirement to build both a client-facing and server application in order to play checkers. By constructing our game engine, we will be able to host any amount of games for an infinite amount of clients to enjoy the classic game of checkers. As for the timeline of this class, CS 451, we hope to implement all of the following requirements and specifications listed out below in detail.

2 Overall Description

2.1 Product Functions

1. Provide a client-facing application to play checkers with another user over the local network or internet.
2. Provide a centralized server that mediates gameplay, game sessions, and client interactions.
3. Provide enough documentation that future team members can pickup the product from scratch and improve on it.

2.2 Assumptions and Dependencies

1. A connection to a local network where the server is hosted or internet connection.
2. A computer with a graphical windowing environment for the client and any *nix or Windows based server.
3. Client who have some knowledge of the rules of checkers and how to play either previously or by looking at our requirements documentation.
4. Client know how to launch a .JAR file from a desktop and interact with simple menu components.

3 Functional Requirements

3.1 Client

R1. Client - Server Interaction

R1.1. Client will automatically check to see if the game server is live given the set URL in the program.

R1.2. Client will not be able to join a lobby if no response is sent from the requested server URL.

R1.3. Client will be able to terminate itself from the server at any given moment.

R1.4. Client will be able to reconnect back to the lobby within 30 seconds, else they will receive a notification saying that the lobby has been closed and they are unable to rejoin due to the timeout period.

R1.5. Client will be able to see all available games to join on the server and be able to select a lobby.

R1.6. Client should be able to distinguish themselves with a valid player name when they join a lobby.

R1.7. Clients will be able to select which color they want to play, and receive the assigned color once the game begins.

3.1.1 Board State

R2. Board

R2.1. The clients will keep a copy of the board state in order to render it

R2.2. The board will update upon receiving an input from either:

Server response

3.2 Server

R3. Server should be able to be run all the time without crashing.

Server will have a heartbeat function that will send an email to the developers if the server is down for some reason.

R4. Server - Client Interaction

R4.1. Server will be able to process client connection information to create a lobby.

R4.2. Server will keep track of clients and game sessions.

R4.3. On user timeout, wait a set amount of time.

If not connected within that time, signal other client game has ended due to disconnected player.

R4.4. Server will validate moves before sending updated move to clients.

On invalid move, signal client that move was invalid and to try again.

R5. Server - Lobby Interaction

R5.1. Server has a fixed range of ports that it can assign a lobby upon creation. This range is to make sure we do not flood our network with ports ranging far away from each other.

R5.2. Server will first check that a port is open before assigning it to a lobby upon creation.

R5.3. Server will close the port assigned to a lobby when the game has been finished.

R5.4. Server will be able to print out a list of lobbies for the clients to see.

R5.5. Server will be able to send notice to any lobbies that the server is shutting down, whether expectedly or unexpectedly, in order to provide better experience for the clients playing.

4 Other Requirements

4.1 System Requirements

S1. Server and Client

JRE Version 12 or higher

S2. Client

Windowing display environment:

Windows

MacOS

Xorg or Wayland

4.2 Network Requirements

N1. Client and Server

N1.1. An active internet connection is required for the client to connect to the server.

N1.2. Client cannot download the server and run it locally, unless the client sets up port forwarding properly on their local network.

N1.3. Client must be connected to Drexel's network in order to play. This requires credentials to sign-in. Client is responsible for their own Drexel credentials.

N1.4. Response time to the server must be less than 120ms. If it is greater, then the game quality may be decreased and we will alert the clients.

N2. Server

N2.1. Server must be hosted on tux.cci.drexel.edu. We are at the mercy of tux.cci.drexel.edu's uptime.

N2.2. Server will be running on one dedicated box, and will not have a load balancer assigned to it, meaning that there will be one direct access point for our clients to connect to.

N2.3. There will be no snapshots saved of the server or gamestates. Server will have little to no storage to save information to load back onto the network.

N2.4. If a client is taking too long to connect, the server will automatically ditch the requested connection.

5 User Interface

5.1 Framework

The project shall use JavaFX to create the user interface. We shall write a wrapper for it to facilitate ease of use.

5.2 Menus

- Main Menu

  Play Game button: connects player to the server to search for a match, moving them to the Game Lobby screen

  Quit Game: closes the game

- Game Lobby

  Show one (or both) players in the lobby

  Ready? button: start game once both players click

  Main Menu button: return to the main menu

- Game

  Checkers match screen, showing a checkers board with both players' pieces

  Player's "side" of the board is always the bottom from their own perspective, mirroring real-life players sitting on opposite sides of a board

  Top of screen shows whose turn it is: Black or Red

  During the player's turn, they click on a piece to select it

  Piece becomes highlighted, as does all valid spaces the player could move to

  Player clicks on a highlighted space to move the piece there or clicks on a new piece to select it

  After a move has been made, the game checks if there is a winner. Move to Winner Display if a winner is found. Otherwise, start next player's turn.

- Winner Display

  Show the player who won the game: Black or Red

  Rematch? No button: return both players to the main menu

  Rematch? Yes button: register this player as wanting a rematch

  If both players click Yes, then return both players to the Game Lobby screen

5.3 Standard Components

- Buttons: The buttons will be used for interaction within menus, allowing for navigation throughout the program.
- Pieces (Black and Red variants): The pieces will be the basic playing pieces within gameplay. Clicking on a piece will select it for making a move during the player's turn.
- King Pieces (Black and Red variants): These pieces will behave identically to the regular pieces (see above) but with additional movement options according to the rules of Checkers.
- Checkers Board: an 8x8 grid of alternating black and red tiles on which all pieces will be displayed in the gameplay. When a piece is selected during a player's turn, all spaces to which the player could move will be highlighted. Clicking on a highlighted space will move the piece to that space, jumping over any opponent pieces if possible.

6 Program Usage

6.1 Lobbies

L1. Lobby Creation

L1.1. Clients should be able to create a lobby (start a new game) or join a lobby (join another player to start a game).

When a client creates a lobby, this opens a port on the server.

When a client joins a lobby, the game then starts processing the pre-game information.

Information in the pre-game is the names of the players, who is selected to play which color, and which client goes first (the client assigned to the red color).

L2. Lobby Rules

L2.1. Lobbies consist of a number of players, whether or not the game is started or finished, and what port it is being hosted on the server.

A player can join a lobby via the port number, as this acts as our identity for each lobby.

L2.2. There will be a maximum of 10 lobbies that the server will allow in order not to crash the server.

L2.3. Lobbies will be locked down once two clients have connected.

If a third client tries to connect to a game-in-progress, an alert will be sent to that user saying "spectating is not allowed for a game-in-progress".

L2.4. If the server were to crash, all lobbies will be terminated without any winners assigned.

L2.5. A lobby can only be initialized when a player creates one. There will be no default lobbies maintained by the server.

L2.6. A lobby will only be assigned to a given range of ports from the server. This is to make sure that the ports are available for creation.

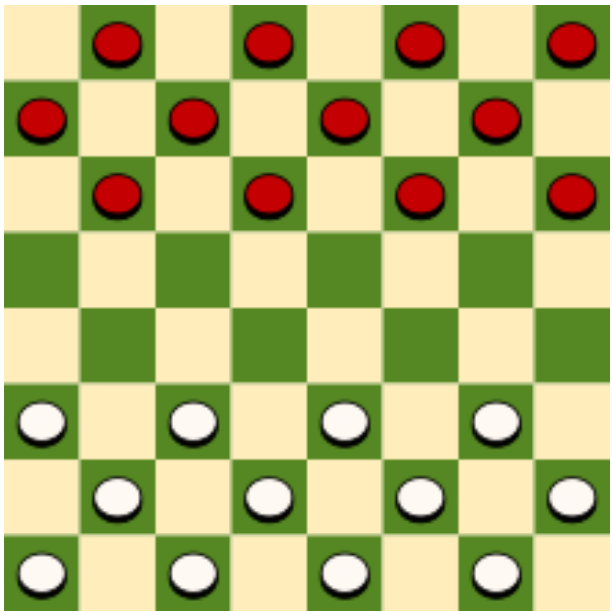
6.2 Gameplay

G1. The match begins with an 8x8 grid with each tile alternating between black and red in color.

G2. One player is assigned the black pieces and the other red.

G3. Each player's pieces start on opposite ends of the board, occupying every other space within the first three rows (for a total of 12 pieces for each player).

See attached image for an example checkers setup:



G4. The black player starts the match by taking their turn.

G5. During each turn, the active player selects a piece of their own to move according to the following rules:

- If the piece is bordering an enemy piece and there is a free space on the other side of that enemy piece, the piece must "jump" to the empty space, removing the enemy piece it moved over from the board

  Multiple jumps can be made in a single turn if the piece is in position to jump an additional enemy piece after completing a jump

- If the piece cannot jump, it may move diagonally one space to an unoccupied space

  If the piece is a non-king, it must move forward

  If the piece is a king, it can move in any direction

- A piece cannot jump over pieces of the same color as itself (friendly pieces)
- Two pieces cannot occupy the same space, regardless of color

G6. When a non-king piece has reached the edge of the board opposite its color's starting side, that piece will be crowned and turned into a king, allowing it to move in any direction.

6.2.1 Win Conditions

1. If player A has no more pieces, player B is the winner and vice versa.
2. If player A disconnects, player B is the winner and vice versa.

Bibliography

[1] The American Checker Foundation, *USA Checkers*, https://www.usacheckers.com/, 2019.

[2] W.J. Rayment, *History of Checkers or Draughts*, http://www.indepthinfo.com/checkers/history.shtml, 2004.