Console-Based Checkers Game
Software Test Case Specifications *

Belser, C.
cmb923@drexel.edu

Brennan, Z.
zab37@drexel.edu

Horsey, K.
kth37@drexel.edu

van Rijn, Z.
zwv23@drexel.edu

August 29, 2017

Abstract

This document is intended to be used in conjunction with the provided `requirements.pdf` and `design.pdf` documents. **Note:** this document is *Proprietary and Confidential*; duplication or reproduction of any content herein must be explicitly granted in writing.

Contents

*CS451:002 Group 2, Drexel University

Revision History

Revision	Date	Author(s)	Description
0.90	August 14, 2017	KH	created initial LaTeX version
0.91	August 19, 2017	ZV	remove all unused code
0.92	August 19, 2017	ZB	Added game start test cases
0.93	August 20, 2017	CB	Added user interface test cases
1.00	August 20, 2017	All	completed documentation

Component	Value	Unit
OS	Linux Mint 18.1	Serena
RAM	12	GB
HDD	949	GB
Docker	17.06.0-ce	(native)

Figure 1: System Spec. Corwin Belser

1 Introduction

1.1 Purpose of Document

This document is to describe the various tests performed on the Console-Based Checkers Game. This game is designed to be played by two people on separate computers, connected to each other over a network.

1.2 Definitions, Acronyms, Abbreviations

1. *Tux* - Linux-based servers available to all Computer Science students at Drexel University.
2. *Boardstate* - The data structure that contains the current state of the board, and the locations of each piece
3. *Board* - Refers to the visual representation of the *BoardState*
4. *Game* - A capitalized **Game** shall be used in place of the full name of Console-Based Checkers, where appropriate

1.3 References

This document may feature terms and references which can be found in the Requirements and Design documents that are associated with the Console-Based Checkers Game project.

2 Testing Environments

Console-Based Checkers and it's associated software and test cases have been run and tested within the following environments.

2.1 E1: Docker (Corwin Belser)

See fig. 1.

2.2 E2: Docker (Kris Horsey)

See fig. 2.

Component	Value	Unit
OS	Windows 10 Home	1607
RAM	8	GB
HDD	237	GB
Docker	5.1.24	r117012

Figure 2: System Spec. Kris Horsey

Component	Value	Unit
OS	Ubuntu 16.04	LTS
RAM	8	GB
HDD	243.5	GB
Docker	1.12.6	(native)

Figure 3: System Spec. Zach Brennan

2.3 E3: Docker (Zach Brennan)

See fig. 3.

2.4 E4: Docker (Zach van Rijn)

See fig. 4.

2.5 E5: Jenkins-enabled Build Platform

See fig. 4.

2.6 E6: Drexel University (`tux.cs`) Platform

Docker is not installed, but Console-Based Checkers is built with static binaries, and therefore can be copied to Tux to be executed. The known specifications for Tux can be found in fig. 5

3 Setup Information and Prerequisites

Before the program is run the following prerequisites must be met.

Component	Value	Unit
OS	CentOS 7	20170322
RAM	64	GB
HDD	40	GB
Docker	17.06.0-ce	r02c1d87

Figure 4: System Spec. Zach van Rijn

Component	Value	Unit
OS	Ubuntu 16.04	(unknown)
RAM	64	GB
HDD	(varies)	GB
Docker	n/a	n/a

Figure 5: System Spec. Drexel's `tux.cs` System

- Make sure your computer is connected to the Internet.
- Have Docker installed and properly configured.
- Followed all the instructions in the Requirements documentation to initiate the game.

4 Issue Tracking

We will be using the following URL for issue tracking:
`https://git.zv.io/me/CS451-Checkers/issues`

5 Test Cases

This document outlines all of the testing that will be performed on Console-Based Checkers, and the expected results of those tests, to ensure that it will retain all functionality that was outlined in the Requirements documentation.

5.1 Test Case 1: Start Game

5.1.1 Description

This case consists of the different components needed for a user to host or join a game of Checkers.

5.1.2 Preconditions

The user has Docker installed on their computer and an Internet connection.

5.1.3 Scenario

Test Case			
Description	Execution Steps	Expected Result	Actual Result
Launch Server	Run the Server application inside of Docker, with the command outlined in the Requirements Document Section 4.2.1	The Server runs, and begins listening for connections on the specified port	**PASS**
Launch Game	Run the Client application inside of Docker, with the port to attempt connection as the argument	The Client runs, and connects to the Server that is listening on the specified port, receiving an ID	**PASS**
Join Game - Success	Be the first or second Client to connect to the Server	Receive an ID from the Server of either 1 or 2, indicating that the Client is recognized as a player	**PASS**
Join Game - Failure	Be the first or second Client to connect to the Server	The Client can not connect to the Server on the specified port, and informs the user of the error	**PASS**
Watch Game - Success	Connect to a Server that already has at least 2 Clients connected	Receive an ID from the Server that is greater than 2, indicating that the Client is recognized as a spectator	**?**
Watch Game - Failure	Connect to a Server that already has at least 2 Clients connected	The Client can not connect to the Server, or has the connection closed by the Server if the Spectator Queue is full. The user is then informed of this error	**PASS**

5.2 Test Case 2: Play Game

5.2.1 Description

This case describes all of the moves and potential states that can occur in a game of Checkers.

5.2.2 Preconditions

The user has successfully connected to a Server, and has an ID of either 1 or 2, indicating that it is one of the active players, and not a spectator.

5.2.3 Scenario

Test Case			
Description	Execution Steps	Expected Result	Actual Result
Piece Move	Select a piece and an open board location	The piece moves to a new location	**PASS**
Piece Invalid Move	Make an illegal move	The Server rejects the move,a warning is displayed, and it remains the same players turn	**PASS**
Piece Jump	Select a piece and pick an open board location beyond an opposing piece	The opposing piece is removed and jumping piece relocates	**PASS**
Piece Crowning	Move the piece to the last row on the board	The crowned piece becomes a king	**PASS**
King Move	Select a crowned piece and an open board location	The crowned piece moves to a new location	**PASS**
King Invalid Move	Make an illegal move with a crowned piece	The server rejects the move,a warning is displayed, and it remains the same players turn	**PASS**
King Jump	Select a crowned piece and pick an open board location beyond an opposing piece	The opposing piece is removed and jumping piece relocates	**PASS**

5.3 Test Case 3: End Game

5.3.1 Description

This case describes all the ways a game of Console-Based Checkers can be terminated.

5.3.2 Preconditions

The user is currently in an active game instance of Console-Based Checkers.

5.3.3 Scenario

Test Case			
Description	Execution Steps	Expected Result	Actual Result
Win Game - Capture	You capture your opponent's last piece	The client will ask if the user wants to disconnect or play again. "You Win" message displayed.	**PASS**
Win Game - No Moves	You create a situation in which your opponent has no valid moves to make	The client will ask if the user wants to disconnect or play again. "You Win" message displayed.	**PASS**
Lose Game - Capture	Your opponent captures your last piece	The client will ask if the user wants to disconnect or play again. "You Lose" message displayed.	**PASS**
Lose Game - No Moves	Your opponent creates a situation in which you have no valid moves to make	The client will ask if the user wants to disconnect or play again. "You Lose" message displayed.	**PASS**
Game Times Out	The user does nothing for 5 mins and lets the game's timer run out.	The other user gets a notification that their opponent disconnected	**PASS**
User gets Disconnected	This occurs when a user gets removed from the game before a time out	The other user gets a notification that their opponent disconnected	**?**

5.4 Test Case 4: User Interface

5.4.1 Description

This case describes the various user interface functionality.

5.4.2 Preconditions

The client is successfully receiving board states and move requests from the server.

5.4.3 Scenario

Test Case			
Description	Execution Steps	Expected Result	Actual Result
Board Display	Client receives board state from server	1. Checker board is rendered to the screen 2. Checker pieces are rendered in their respective locations	**PASS**
Move Highlighting	Client generates list of possible moves to be selected by player, passing them to the user interface component	Pieces available to be moved have a highlight rendered around them	**PASS**
Move Selection	Client renders the highlighting on available moves	1. The first piece is rendered with a blinking highlight 2. Pressing the left or right arrow key moves the blinking highlight in the respective direction. 3. Pressing the enter key changes the highlight color off the selected piece & removes the highlight from all other pieces. The possible destinations for the selected piece are then highlighted as in step 1.	**PASS**
Passive Message Display	Client receives a passive message from the server	Passive message is rendered to the top of the terminal, overwriting any previous message & remaining until overwritten	**PASS**
Active Message Display	Client receives an active message from the server	Active message is rendered in the center of the screen over top of the checkers board. Pressing the enter key removes this message.	**PASS**

6 Glossary

Below is a comprehensive list of some of the terms and language that we use within this document, knowledge of which will lead to a more effective understanding of our product's design and aid in future communications regarding our product.

- **Checkerboard** NxN (typically 8x8) game board composed of alternating black/red (or other) squares on which game pieces *Checkers* reside
- **Piece** Standard Checkers disc with limited movement, specifically only forward-diagonal motion
- **King** Checkers piece that can move along any diagonals, forward or backward
- **Move** The act of changing the location of a piece on the board when it is that player's turn
- **Jump** The act of removing an opposing player's piece from the board, occurring in a straight diagonal fashion – e.g., "hopping over" the opponent
- **Crowning** The act of changing a standard game *Piece* to a *King*
- **Active Message** A message that interrupts gameplay, and is placed in the center of the board, requiring interaction from the player to clear it and allow play to resume
- **Passive Message** A message that does not interrupt gameplay, and is displayed above the board