Software Test Case Document

Checkers

By: Justin Seara, Roger White, Joe Chew, Gregory Kilmer

Table of Contents

Table of Contents 2

Introduction 3

Definitions, Acronyms, and Abbreviations 3

References 3

Testing Environments 3

My Computer 3

Joes Computer 3

Mike's Computer 3

Roger's Computer 3

Setup Information and Prerequisites 3

Test Cases 3

Appendix 3

Introduction

The purpose of this document is to describe the testing approaches used while evaluating the functionality and performance of the Checkers program as to meet the requirements outlined in the requirements document. Our Checkers program is being created in Unity (C#), and will allow two players to play a game of checkers from two remote locations.

Definitions, Acronyms, and Abbreviations

Please refer to the Appendix and Glossary sections for any definitions and abbreviations.

References

The document may feature terms and references which can be found in the preceding requirements and design documents related to our Checkers program.

Testing Environments

Justin’s Computer

Machine Name	Windows PC	DB Directory	N/A		
OS and Version	Windows 7: 16 GB RAM: 256 SSD: 1 TB HDD	Interpreter Platform		Client Server/Back-end	Unity
Tester Name	Justin Seara	Test Date	8/17/2017		
New Log				State	**PASS**

Joe’s Computer

Machine Name	MacBook Pro	DB Directory	N/A		
OS and Version	Mac OS X Sierra: 16 GB RAM: 1 TB HDD:	Interpreter Platform		Client Server/Back-end	Unity
Tester Name	Joe Chew	Test Date	8/17/2017		
New Log				State	**PASS**

Greg’s Computer

Machine Name	Windows PC	DB Directory	N/A		
OS and Version	Windows 10: 8 GB RAM: 500 GB SSD	Interpreter Platform		Client Server/Back-end	Unity
Tester Name	Greg Kilmer	Test Date	8/17/2017		
New Log				State	**PASS**

Roger’s Computer

Machine Name	Macbook Pro	DB Directory	N/A		
OS and Version	Mac OS Sierra 16 GB RAM 1 TB HDD 2.9 GHz Intel Core i7	Interpreter Platform		Client Server/Back-end	Unity
Tester Name	Roger White	Test Date	8/17/2017		
New Log				State	**PASS**

Setup Information and Prerequisites

The following prerequisites must be met to run the program.

- The program can be run on any system, so long as the system has access to the internet, and can utilize a modern web browser (Mozilla Firefox, Google Chrome, etc).
-

Test Cases

Starting Game

Description

This case consists of covering the steps required to host or join a game on a computer

Precondition

An internet connection

Scenario

Test Case					
ID	Req	Description	Execution Steps	Expected Result	Actual Result
A1		Launch App	Open app via Unity	GUI window launches in browser showing start screen	
A2		Open Server Menu	Click “Server Options” in Main Menu	New GUI screen showing server options appears	
A3	R1.4, R1.5	Create Server	1: Click “Create Server” in the server options menu 2: Name server and add a password	1: New Create Server modal opens 2: new server is created with name and password	
A4	R1.1-R1.3	Join Server	1: Click “Join Server” in the server options menu 2: select the desired server and enter password	1: Server is found in the Join Server screen 2: Server is joined and a game begins	

Playing Game (Valid and Invalid Moves) Greg-Mike

Description

The case consists of covering (testing) the steps involved in mutual play of the game by the host and the opponent. The test cases validate if the logic behind the moves has been correctly implemented and follows the rules of the game as set forth by the Checkers Rules provided. [CHECK]

Precondition

An internet connection, a game is already set up between the host and the opponent.

Scenario

Test Case					
ID	Req	Description	Execution Steps	Expected Result	Actual Result
B1		Move normal piece forward	1. Select Piece 2. Select unoccupied destination in front of piece and diagonal	Piece will be moved to the new destination.	
B2		Move normal piece backwards	1. Select Piece 2. Select unoccupied destination behind piece and diagonal	Piece will not be moved to a new destination	
B3		Move normal piece not on a diagonal	1. Select Piece 2. Select unoccupied destination adjacent to piece	Piece will not be moved to a new destination	
B4		King piece movement	1. Select King Piece 2. Select unoccupied destination diagonally in front of piece 3. Select King Piece 4. Select unoccupied destination diagonally behind piece	The piece will be moved to a new location	
B5		Move piece to occupied location	1. Select Piece 2. Select destination that is occupied	The piece will not be moved to a new location	
B6		Piece capture	1. Select Piece	Piece will be	

			2. Select legal destination diagonally on the other side of opponent piece	moved to new destination and opponents piece will be removed from the game	
B7		King a piece	1. Select normal piece 2. Select legal destination at opponents end of board	Piece will be moved to new destination and will become a king piece	

Ending Game (Roger)

Description

This case covers the steps involved in ending the game at the proper time and and the logic behind selecting the winner of the game according to the conditions set forth by the Checkers Rules provided. This case also covers the steps involved in restarting or exiting the game.

Precondition

An internet session, and a game currently being played between two players.

Scenario

Test Case					
ID	Req	Description	Execution Steps	Expected Result	Actual Result
C1		Determine if there is a winner.	1. Count number of pieces in play on each color. 2. If one color has zero pieces in play, there has been a victory.	Game Over Conditional will be set to true.	

C2		Determine which side won	1. The color with the final move before the Game Over conditional was set to true is the winning color. 2. Double check by counting the active pieces of the 'winning' color and making sure the number is greater than one.	Winning color is the last color that moved before game over condition.	
C3		Display the Game Over Menu, with options to restart or quit.	1. Disable all in game buttons. 2. Enable Game over menu in the foreground.	Users will be unable to move any game pieces, and will see the game over menu in the foreground.	
C4		Restart the game if both users agree to.	1. If user 1 selects 'Rematch' wait for user 2 to make a decision. 2. If user 2 also selects 'rematch', keep server active and restart the game.	Game will restart if both users select 'Rematch' in the game over screen.	
C5		Return user to the menu if disagreement over restarting the game.	1. If user 1 selects 'Rematch' wait for user 2 to make a decision. 2. If user 2 selects 'Exit' or takes too long to decide, remove user 1 from the server and send him to the main menu.	Send the user to the main menu if the opponent does not wish to participate in a rematch.	
		Exit Game	1. If a user selects	User gets	

			‘Exit’, remove him from the server, and send him back to the main menu.	sent to main menu after selecting ‘Exit’ on the game over menu.	

Appendix

Mythical Man Month

What rule of thumb for scheduling a software task does Brooks propose?

For some years I have been successfully using the following rule of thumb for scheduling a software task:
l/3 planning
l/6 coding
l/4 component test and early system test
l/4 system test, all components in hand.
This differs from conventional scheduling in several important ways:

Brooks contends that *conceptual integrity* is the most important characteristic in system design.
What does *conceptual integrity* refer to?

"Conceptual integrity in turn dictates that the design must proceed from one mind, or from a very small number of agreeing resonant minds."

What is the fundamental problem with system maintenance?

DIRECT QUOTE: “The fundamental problem with program maintenance is that fixing a defect has a substantial (20-50 percent) chance of intro- ducing another. So the whole process is two steps forward and one step back.”

In his famous work "*No Silver Bullet - Essence and Accident in Software Engineering*", Brooks presented several attacks that address the essence of the software problem. Which of these attacks do you consider to be the most promissing?

1. The fraction devoted to planning is larger than normal. Even so, it is barely enough to produce a detailed and solid specification, and not enough to include research or exploration of totally new techniques. 2. The half of the schedule devoted to debugging of completed code is much larger than normal. 3. The part that is easy to estimate, i.e., coding, is given only one-sixth of the schedule.