CS451

Checkers Test Case Document

Summer 2017

Group members	Hajer Karoui, Samuel Nathanson, Curtis Bechtel, Julie Soderstrom
Faculty advisor	Dr. Filippos Vokolos, Ph. D.

Revision History

Name	Date	Reason for change	Revision
Hajer Karoui Samuel Nathanson Curtis Bechtel Julie Soderstrom	08/15/2017	Initial version	1.0
Hajer Karoui Samuel Nathanson Curtis Bechtel Julie Soderstrom	08/20/2017	Final version	2.0

Table of Contents

1. Introduction

1.1. Purpose of the document

The purpose of this document is to describe the test cases that will be used to assess the functionality of this project, and to set up the testing environment and process. Our project is a JavaScript checkers application that enables 2 players to play remotely against each other by connecting to a server through a public website.

1.2. Definitions

A game consists of a standard checkers board, pieces, and at least one online player. (further describe standard checkers board and pieces)

Public games are displayed on a list to all players interested in joining a game.

Private games are only visible to and accessible by players with the game's private ID key.

New games are ones which do not yet have a second player.

Active games are ones with two online players.

Checker piece: a piece that can be moved and which did not reach the last row yet.

King: a piece that reaches the last row (the King Row).

2. Testing Environments

2.1. Mocha on NPM

The majority of JavaScript testing will be made through the Node Package Manager, commonly known as NPM. This program is a command line executable which is compatible with shells such as Bash and Windows Command Line on operating systems including Ubuntu, Windows, and MacOS.

The tests will be run using an NPM package called Mocha. Mocha is a purely JavaScript unit testing engine with helpful testing tools and configuration options. It will also be used to develop some stress-tests for the server and database.

There will also be a code coverage tool through NPM called Istanbul. Istanbul is a purely JavaScript code coverage tool which depends on output from compatible testing engines such as Mocha. Istanbul will be used for generating a code coverage report (in HTML) periodically during development.

To run all tests, navigate to the project directory and enter "npm test". This will print out the results of all Mocha tests and create an HTML page in the coverage directory. This page will display the line-by-line coverage status.

NPM Version: NPM 3.1.0 for Bash
Mocha Version: Mocha 3.5.0
Istanbul Version: Istanbul 11.1.0
Operating System: Ubuntu 16.04 LTS

2.2. Chrome

Since it is one of the most widely used desktop web browsers, Google Chrome will be the primary target of client-side testing.

Client-side tests will focus on detecting incorrect functionality in the user interface and user interaction. These tests must be run manually and cannot be automated.

Version: Chrome 59.0.3071.86
Operating System: Ubuntu 16.04 LTS

2.3. Firefox

Mozilla Firefox will also be used in client-side testing of the application for the same purpose as Chrome.

Version: Firefox 55.0.2
Operating System: Ubuntu 16.04 LTS

2.4. Internet Explorer

Internet Explorer will also be used in client-side testing of the application for the same purpose as Chrome and Firefox.

Version: Internet Explorer 11
Operating System: Windows 10

3. Setup and Prerequisites

Prior to attempting any tests, the following prerequisites must be met:

1. The server and database must both be deployed with the latest code and running.
2. The server must be accessible from the internet through the designated URL: https://cs-451-checkers.herokuapp.com
3. The client must have an internet connection through one of the previously listed internet browsers.

4. Test Cases

4.1. Gameplay

Event	Execution	Expected	Actual
Move King/piece forward (legal)	1. Player clicks on the King/piece to be moved	King/piece moves into the chosen tile and the previous tile is empty.	

	2. Player clicks on an empty diagonal space in front of the selected piece 3. Player clicks "Submit Move" button		
Move King/piece into occupied space (illegal)	1. Player clicks on the King/piece to be moved 2. Player clicks on any occupied diagonal space 3. Player clicks "Submit Move" button	King/piece does not move. Player is notified that it's an illegal move.	
Move King/piece into direct front tile (illegal)	1. Player clicks on the King/piece to be moved 2. Player clicks on an empty space directly in front of their tile 3. Player clicks "Submit Move" button	King/piece does not move. Player is notified that it's an illegal move.	
Jump forward (legal)	1. Player clicks on the King/piece to be moved 2. Player clicks on an empty diagonal space that is after an occupied diagonal space 3. Player clicks "Submit Move" button	The opponent's piece should be removed and the King/piece should be moved to the 2nd diagonal free space and remain there	

Jump Forward into occupied space (illegal)	1. Player clicks on the King/piece to be moved 2. Player clicks on any occupied diagonal space 3. Player clicks "Submit Move" button	King/piece does not move. Player is notified that it's an illegal move.	
Jump Forward over empty space (illegal)	1. Player clicks on the King/piece to be moved 2. Player clicks on an empty diagonal space that is after another empty diagonal space 3. Player clicks "Submit Move" button	King/piece does not move. Player is notified that it's an illegal move.	
Multiple Jumps Forward (legal)	1. Player clicks on the King/piece to be moved 2. Player clicks on an empty diagonal space that directly follows an occupied diagonal space 3. Player clicks on another empty diagonal space 4. Continues this process until the player is done 5. Player clicks "Submit Move" button	The opponent's pieces that are "jumped over" should be removed and the King/piece should be moved to the last diagonal free space clicked, and remain there	
Multiple Jumps Forward with a jump into an	1. Player clicks on the King/piece to be moved	The King/piece does not move. Player is notified that it's an illegal move.	

occupied space (illegal)	2. Player clicks on an empty diagonal space that directly follows an occupied diagonal space 3. Player clicks on another occupied diagonal space 4. Continues this process until the player is done 5. Player clicks "Submit Move" button		
Multiple Jumps Forward with a non-diagonal jump (illegal)	1. Player clicks on the King/piece to be moved 2. Player clicks on an empty diagonal space that directly follows an occupied diagonal space 3. Player clicks on a space directly next to/in front of the previously selected space 4. Player clicks "Submit Move" button	The King/piece does not move. Player is notified that it's an illegal move.	
Move King Backwards (legal)	1. Player clicks on the King to be moved 2. Player clicks on an empty diagonal space behind the selected King 3. Player clicks "Submit Move" button	King has moved back into a free diagonal space and remains there	

Move piece backwards (illegal)	1. Player clicks on the piece to be moved 2. Player clicks on an empty diagonal space behind the selected piece 3. Player clicks "Submit Move" button	Piece does not move. Player is notified that it's an illegal move.	

Event	**Execution**	**Expected**	**Actual**
Move King/piece forward (legal)	1. Player clicks on the King/piece to be moved 2. Player clicks on an empty diagonal space in front of the selected piece 3. Player clicks "Submit Move" button	King/piece moves into the chosen tile and the previous tile is empty.	
Move King/piece into occupied space (illegal)	1. Player clicks on the King/piece to be moved 2. Player clicks on any occupied diagonal space 3. Player clicks "Submit Move" button	King/piece does not move. Player is notified that it's an illegal move.	
Move King/piece into direct front tile (illegal)	1. Player clicks on the King/piece to be moved 2. Player clicks on an empty space directly in front of their tile	King/piece does not move. Player is notified that it's an illegal move.	

	3. Player clicks "Submit Move" button		
Jump forward (legal)	1. Player clicks on the King/piece to be moved 2. Player clicks on an empty diagonal space that is after an occupied diagonal space 3. Player clicks "Submit Move" button	The opponent's piece should be removed and the King/piece should be moved to the 2nd diagonal free space and remain there	
Jump Forward into occupied space (illegal)	1. Player clicks on the King/piece to be moved 2. Player clicks on any occupied diagonal space 3. Player clicks "Submit Move" button	King/piece does not move. Player is notified that it's an illegal move.	
Jump Forward over empty space (illegal)	1. Player clicks on the King/piece to be moved 2. Player clicks on an empty diagonal space that is after another empty diagonal space 3. Player clicks "Submit Move" button	King/piece does not move. Player is notified that it's an illegal move.	
Multiple Jumps Forward (legal)	1. Player clicks on the King/piece to be moved 2. Player clicks on an empty diagonal space that directly follows an occupied diagonal space 3. Player clicks on another empty diagonal space	The opponent's pieces that are "jumped over" should be removed and the King/piece should be moved to the last diagonal free space clicked, and remain there	

	4. Continues this process until the player is done 5. Player clicks "Submit Move" button		
Multiple Jumps Forward with a jump into an occupied space (illegal)	1. Player clicks on the King/piece to be moved 2. Player clicks on an empty diagonal space that directly follows an occupied diagonal space 3. Player clicks on another occupied diagonal space 4. Continues this process until the player is done 5. Player clicks "Submit Move" button	The King/piece does not move. Player is notified that it's an illegal move.	
Multiple Jumps Forward with a non-diagonal jump (illegal)	1. Player clicks on the King/piece to be moved 2. Player clicks on an empty diagonal space that directly follows an occupied diagonal space 3. Player clicks on a space directly next to/in front of the previously selected space 4. Player clicks "Submit Move" button	The King/piece does not move. Player is notified that it's an illegal move.	
Move King Backwards (legal)	1. Player clicks on the King to be moved 2. Player clicks on an empty diagonal space behind the selected King	King has moved back into a free diagonal space and remains there	

	3. Player clicks "Submit Move" button		
Move piece backwards (illegal)	1. Player clicks on the piece to be moved 2. Player clicks on an empty diagonal space behind the selected piece 3. Player clicks "Submit Move" button	Piece does not move. Player is notified that it's an illegal move.	
Move King backwards into occupied space (illegal)	1. Player clicks on the King to be moved 2. Player clicks on an occupied diagonal space behind the selected King 3. Player clicks "Submit Move" button	King does not move. Player is notified that it's an illegal move.	
Move King into tile directly behind it (illegal)	1. Player clicks on the King to be moved 2. Player clicks on an empty space directly behind the selected King 3. Player clicks "Submit Move" button	King does not move. Player is notified that it's an illegal move.	
King jumps backward (legal)	1. Player clicks on the King to be moved 2. Player clicks on an empty diagonal space behind the selected King that is further behind an occupied diagonal space 3. Player clicks "Submit Move" button	The opponent's piece should be removed and the King should be moved backward to the 2nd diagonal free space and remain there	

Piece jumps backwards (illegal)	1. Player clicks on the piece to be moved 2. Player clicks on an empty diagonal space behind the selected piece 3. Player clicks "Submit Move" button	Piece does not move. Player is notified that it's an illegal move.	
King jumps backward into occupied space (illegal)	1. Player clicks on the King to be moved 2. Player clicks on an occupied diagonal space behind the selected King that is further behind an occupied diagonal space 3. Player clicks "Submit Move" button	King does not move. Player is notified that it's an illegal move.	
King jumps backwards over empty space (illegal)	1. Player clicks on the King to be moved 2. Player clicks on an empty space two spaces directly behind the selected King that is further behind an occupied space 3. Player clicks "Submit Move" button	King does not move. Player is notified that it's an illegal move.	
Multiple King jumps Backwards (legal)	1. Player clicks on the King to be moved 2. Player clicks on an empty diagonal space that is after an occupied diagonal space behind the selected King 3. Player clicks on another empty diagonal space	The opponent's pieces that are "jumped over" should be removed and the King should be moved to the last diagonal free space and remain there.	

	4. Continues this process until the player is done 5. Player clicks "Submit Move" button		
Multiple piece/king jumps backwards with a jump into an occupied space (illegal)	1. Player clicks on the King/piece to be moved 2. Player clicks on an empty diagonal space that is behind an occupied diagonal space 3. Player clicks on an occupied diagonal space 4. Continues this process until the player is done 5. Player clicks "Submit Move" button	The King does not move. Player is notified that it's an illegal move.	
Multiple Jumps backwards with a non-diagonal jump (illegal)	1. Player clicks on the King/piece to be moved 2. Player clicks on an empty diagonal space that directly follows an occupied diagonal space 3. Player clicks on a space directly next to/behind of the previously selected space 4. Player clicks "Submit Move" button	The King does not move. Player is notified that it's an illegal move.	
Multiple Jumps Backwards with a piece (illegal)	1. Player clicks on the piece to be moved 2. Player clicks on an empty diagonal space that directly follows an	The piece does not move. Player is notified that it's an illegal move.	

	occupied diagonal space behind the selected piece 3. Player clicks on another empty diagonal space 4. Continues this process until the player is done 5. Player clicks "Submit Move" button		

4.2. End Game

Event	Execution	Expected	Actual
Time is depleted	1. Player waits until the game timer reaches zero.	A message appears to the player indicating that the game has ended and that they have lost.	
Opponent's time is depleted	1. Opponent waits until the game timer reaches zero.	A message appears to the player indicating that the game has ended and that they have won.	
Pieces are depleted	1. Opponent makes moves until they have captured all of the player's pieces.	A message appears to the player indicating that the game has ended and that they have lost.	
Opponent's pieces are depleted	1. Player makes moves until they have captured all of the opponent's pieces.	A message appears to the player indicating that the game has ended and that they have won.	
Forfeit	1. Player 1 clicks the "Forfeit" button.	A message appears to both players indicating that the game has ended and that player 1 lost and player 2 won.	

Opponent Forfeit	1. Opponent presses the "Forfeit" button.	A message appears to the player indicating that the game has ended and that they have won.	

4.3. Client

Event	Execution	Expected	Actual
Launch Application	1. Navigate to https://cs-451-checkers.herokuapp.com/index.html	Main page loads on the browser. The new page displays a menu on the left side of the screen with "Host Game" and "Join Game" buttons.	
Join Game: Page Requested	1. Launch the main page 2. Click the "Join Game" button	Main screen displays a page with "Join Game" box, which contains a form with two input text boxes: Username and Game ID. The page also contains a list of all new public games.	
Host Game: Page Requested	1. Launch the main page 2. Click on the "Start Game" button	Main screen displays a form with a textbox for the username, radio buttons to select the mode of the game, and radio buttons to choose the color combination of the tiles. The form contains a submit.	
Host Game: Private	1. Enter username 2. Select **Private** as the game mode 3. Select a color 4. Click the "Submit" button	Game successfully created: host is notified through a prompt displaying the Game-ID.	

Host Game: Public	1. Enter username 2. Select **Public** as the game mode 3. Select a color 4. Click the “Submit” button	Game successfully created: host is notified through a prompt displaying the Game-ID. Host is taken to the main page where the game information is observable in the list of new games.	
Join Game: success	1. Enter valid Game ID 2. Enter player username 3. Click the “Submit” button	Player successfully added to the chosen game after game ID validation. Host is notified, checker page loads for both players and the game starts.	
Join Game: Invalid ID	1. Enter invalid Game ID 2. Enter player username 3. Click the “Submit” button	User is notified that the ID entered is invalid, and prompted to re-enter valid information.	

4.4. Database Manager

Event	**Execution**	**Expected**	**Actual**
Create Random ID	1. Call the createRandomID function	The generated ID has the correct length and character set (as specified by global constants).	
Create URL	1. Call the createURL function	The generated URL conforms to the MongoDB URL format.	
Connect when Database is Up	1. Ensure that the database is running. 2. Call the connect function	The request returns a success code and loads all database collections into the colls object.	
Connect when Database is Down	1. Ensure that the database is not running.	The request returns an error code and the db object is set to null.	

	2. Call the connect function		
Add Player	1. Call the addPlayer function with a valid player object	The player object is added to the players collection in the database.	
Add Game	1. Call the addGame function with a valid game object	The game object is added to the games collection in the database.	
Get Player	1. Call the getPlayer function with a valid player ID	The correct player object is returned unmodified from the database.	
Get Game	1. Call the getGame function with a valid game ID	The correct game object is returned unmodified from the database.	
Update Player	1. Call the updatePlayer function with a valid player object	The player object in the database with the same ID is replaced with the new one.	
Update Game	1. Call the updateGame function with a valid game object	The game object in the database with the same ID is replaced with the new one.	

4.5. Server

Event	**Execution**	**Expected**	**Actual**
Get Games Request	1. Send a POST request to the get-games endpoint.	A list of all new public games are returned to the sender.	
New Game Request	1. Send a POST request to the new-game endpoint.	New player and game objects are added to database. Player and game objects are returned to sender.	

Invalid New Game Request	1. Send a POST request with bad JSON format to the new-game endpoint.	Database is unchanged. Error code is returned to sender.	
Join Game Request	1. Send a POST request to the join-game endpoint.	A new player object is added to the database. The game is updated to reflect the new player. Both objects are returned to the sender.	
Invalid Join Game Request	1. Send a POST request with bad JSON format to the join-game endpoint.	Database is unchanged. Error code is returned to sender.	
Make Move Request	1. Send a POST request to the make-move endpoint.	The move is checked for validity. If so, the game is updated and returned to sender.	
Invalid Make Move Request	1. Send a POST request with bad JSON format to the make-move endpoint.	Database is unchanged. Error code is returned to sender.	
Get Updates Request	1. Send a POST request to the get-updates endpoint.	If there are updates to be received, the list of messages and current game state are returned.	
Invalid Get Updates Request	1. Send a POST request with bad JSON format to the get-updates endpoint.	Database is unchanged. Error code is returned to sender.	
Send Message Request	1. Send a POST request to the send-message endpoint.	Message is added to the opponent's list of messages within the database.	
Invalid Send Message Request	1. Send a POST request with bad JSON format to the send-message endpoint.	Database is unchanged. Error code is returned to sender.	

Get Homepage Request	1. Send a GET request to the root endpoint.	The application homepage is returned and all necessary resources are loaded.	

4.6. Game UI

Event	Execution	Expected	Actual
Initial Board State	1. Player starts a new game.	Board is populated with twelve normal pieces on each side.	
Surrender	1. Player clicks the “Forfeit” button.	Opponent gets notified. End game with a message with the winner’s name.	
Request Draw	1. Player clicks the “Request Draw” button.	The opponent receives a notification, prompting them to either accept or decline the draw requested by the other player.	
Accept Draw	1. Player clicks the “Accept Draw” button in the prompt.	Game ends with a message popping up to both players signaling a draw and end of the game, with no winners.	
Decline Draw	1. Player clicks “Decline Draw” in the prompt.	The player who requested the draw is notified through a prompt that their opponent declined the draw, and the game continues. (opponent’s turn).	
Pause	1. Player clicks the “Pause” button.	Message appears on the screen for both players and the timer pauses.	
Continue	1. Player clicks the “X” button on the top right corner of the overlay screen while game is paused.	Game resumes, and the timer resumes, board is visible to both players.	

Making any move	1. Player makes any move on the board	Button “Undo Move” turns from red to green and is enabled, signaling the possibility to undo the move.	
Undo move	1. Player makes any move 2. Player clicks on “Undo move” button	The move is undone, and the piece goes back to its original space.	
Information Box	1. Player clicks the “Help” button on the side menu.	The timer is paused and an overlay screen appears, listing the rules of the game, with an option to exit.	

5. References

[1] Karoui, Nathanson, Bechtel, Soderstrom. (Remote Checkers Requirements Specifications Document). Drexel University. Philadelphia, PA, United States, 2017.