WebCheckers Design Documentation

The following template provides the headings for your Design Documentation. As you edit each section make sure you remove these commentary 'blockquotes'; the lines that start with a > character and appear in the generated PDF in italics.

Team Information

- Team name: Insert Name Here (Team B)
- Team members
    - Sree Jupudy
    - Kesa Abbas
    - Jon Pofcher
    - Scott Court
    - Chris Tremblay

Executive Summary

Purpose

Allow players to play checkers with other players that are signed in. Allow them to play on a board where pieces can be easily dragged and dropped.

Glossary and Acronyms

Term	Definition
VO	Value Object
MVP	Minimum-Viable Product
UI	User Interface

Requirements

As a player I want to be able to sign in and see other people I can play checkers against. Once I have selected and opponent I want to be able to play a game of checkers by the American Standard Rules. I want to be able to make a move by dragging a piece when it is my turn. I want to be able resign a game at any point and have it end, or end the game by winning or loosing. I want to be able to sign out at any point after signing in.

Definition of MVP

Every player will be able to sign in and then play a game. Two players must be able to play checkers by the American Standard Rules. Players must be able to choose to resign at any point which will end the game.

MVP Features

- As a player I want to make a move so that the game progresses.
- As a player I want to end the game so that a winner is declared.
- As a player I want to start a game so that I can play checkers with an opponent.
- As a player I want to sign in so I can play a game of checkers.
- As a player I want to sign out so I can stop checkers.

Roadmap of Enhancements

- Spectate, As a Player I want to spectate a game so that I can see the moves being made in real time.
- Replay, As a player, I want to be able to replay the game so that I can see what moves I used in the previous game.

Application Domain
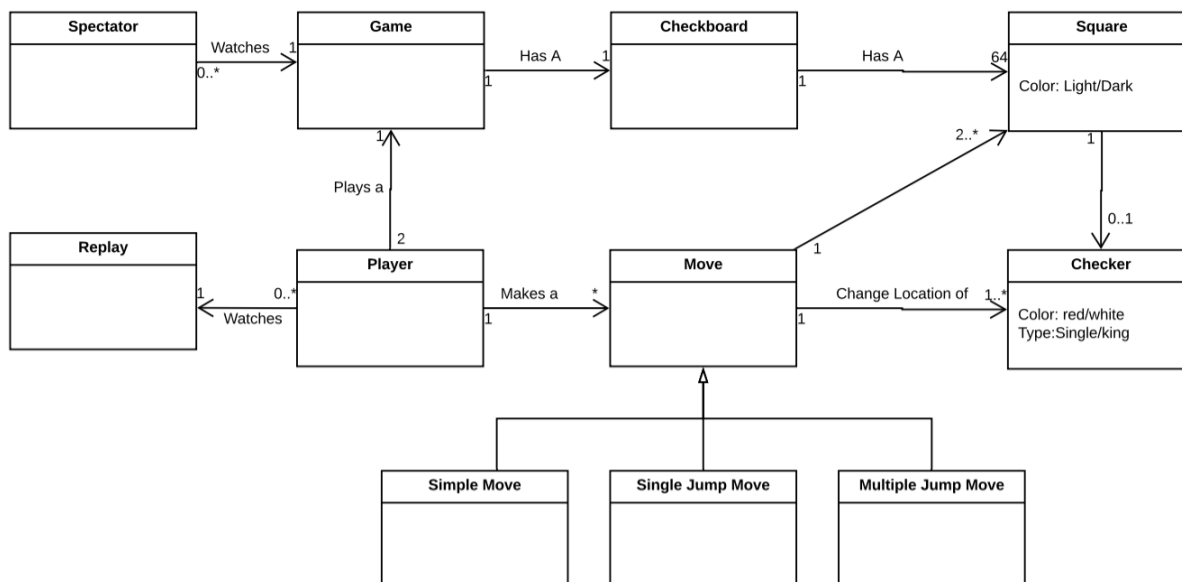
This section describes the application domain.

Figure 1: The WebCheckers Domain Model

A Player plays of game checkers, where they can make a move. A move changes the space the checker is at on the checkerboard. The checkerboard holds and displays the state of the game. A player plays a game of checkers against another player with a variable amount of spectators watching the game. A player can watch a replay of a game that has already been played.

Architecture and Design

This section describes the application architecture.

Summary

The following Tiers/Layers model shows a high-level view of the webapp's architecture.

As a web application, the user interacts with the system using a browser. The client-side of the UI is composed of HTML pages with some minimal CSS for styling the page. There is also some JavaScript that has been provided to the team by the architect.

The server-side tiers include the UI Tier that is composed of UI Controllers and Views. Controllers are built using the Spark framework and View are built using the FreeMarker framework. The Application and Model tiers are built using plain-old Java objects (POJOs).

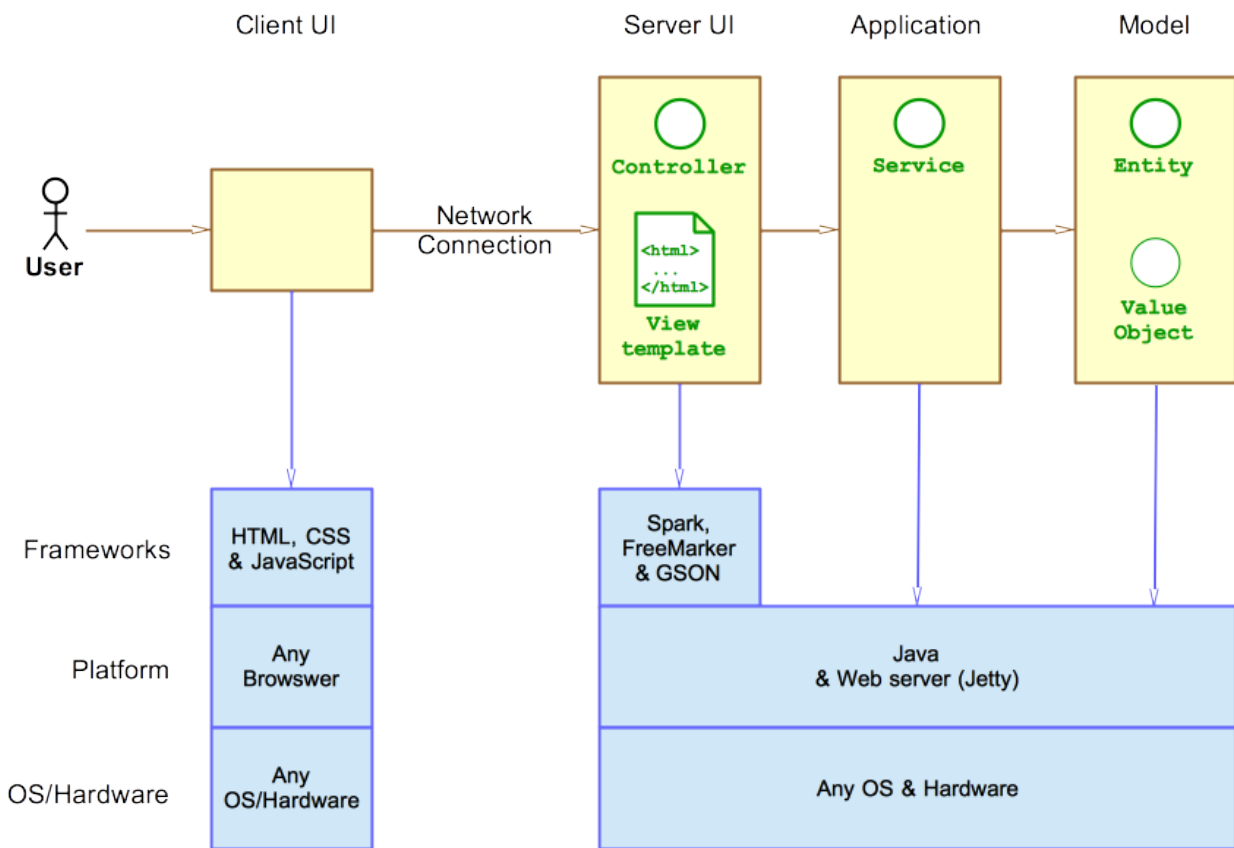Details of the components within these tiers are supplied below.

Figure 2: The Tiers & Layers of the Architecture

Overview of User Interface

This section describes the web interface flow; this is how the user views and interacts with the WebCheckers application.
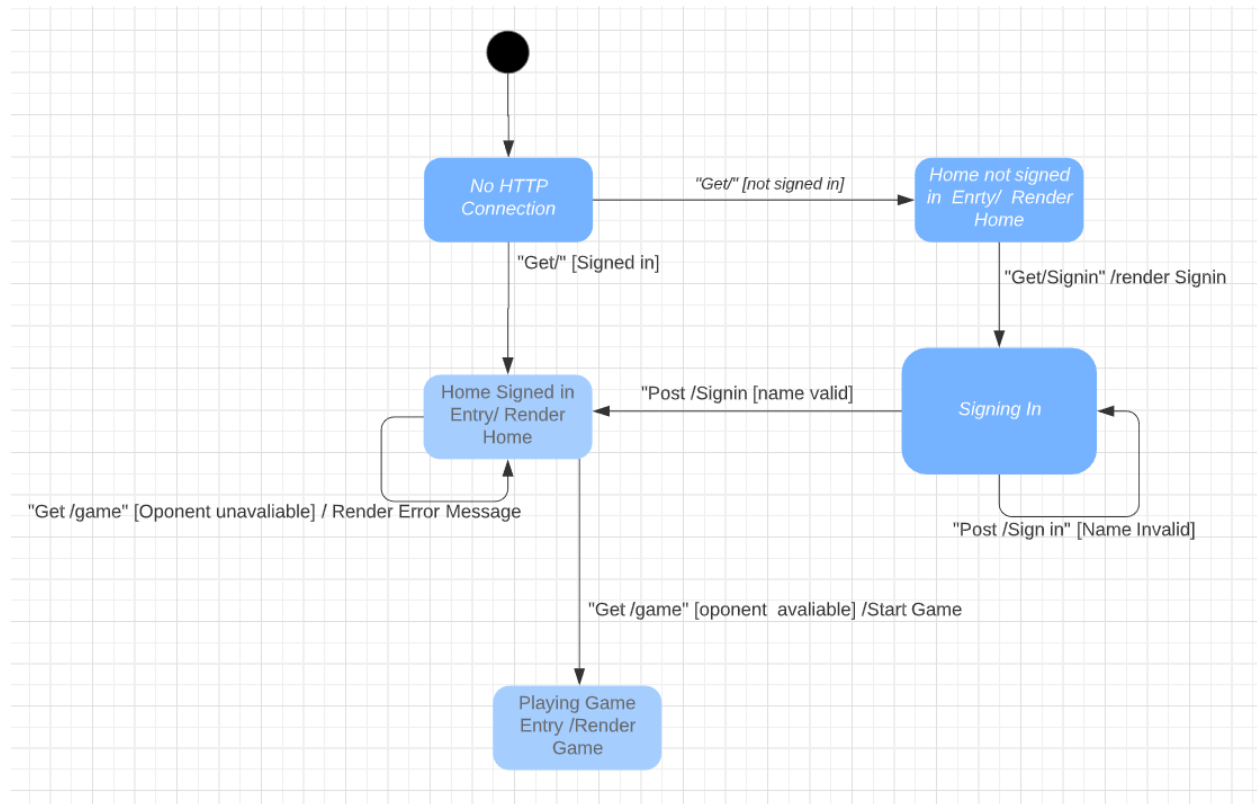

Figure 3: The WebCheckers Web Interface Statechart

When a player first connects to the WebCheckers site they will be met with home page and an option to sign in. Clicking the sign in button will take them to the sign in page where they can enter a name, and sign in with it. If the name is not acceptable it will redirect them back to the sign in page. If the name is accepted it will send them back to the home page where they can see other players that are signed in. Upon selecting another players name, it will send them to a game page where they can player their game of checkers.

UI Tier

The user can sign in with the parameters for the username as follows: - Must be equal to or more than 6 characters long - Cannot be than 25 characters long - It must contain at least 1 number - The first character cannot be a number - Cannot already be taken - Can contain spaces between words

The UI Tier allows the player to sign-in, sign-out, make moves and sign out. GET Routes in the UI Tier handle the redirection to the appropriate route. The routes and their specific functions: - GetGameRoute: It returns the current state of the game to the one of the players. It aides in the display of the correct orientation of the board for the player. - GetHomeRoute: It displays the messages to the user. - GetSignInRoute:It redirects user to the sign in page of the game. - PostBackUpMoveRoute: It asks the server to remove the last move made taht was validated. - PostCheckTurnRoute: After a valid move is done by the player, this route submits it to the server for validation. - PostResignRoute:It informs the server that the player is resigning the game, and

4

when the action is successful, it redirects the user to homepage. - PostSignInNameRoute: It posts the sign in page for the player, and - PostSignOutRoute: It post the dign out page to the player after the player signs out. - PostSubmitTurnRoute: Utilizing this action, it submits the user's turns. - PostValidateMoveRoute: It submits a single move made by the player to the server to check if that action is valid or not.

The server responsible for handling the POST and GET routes is the Jetty WebServer in the WebServer Class.
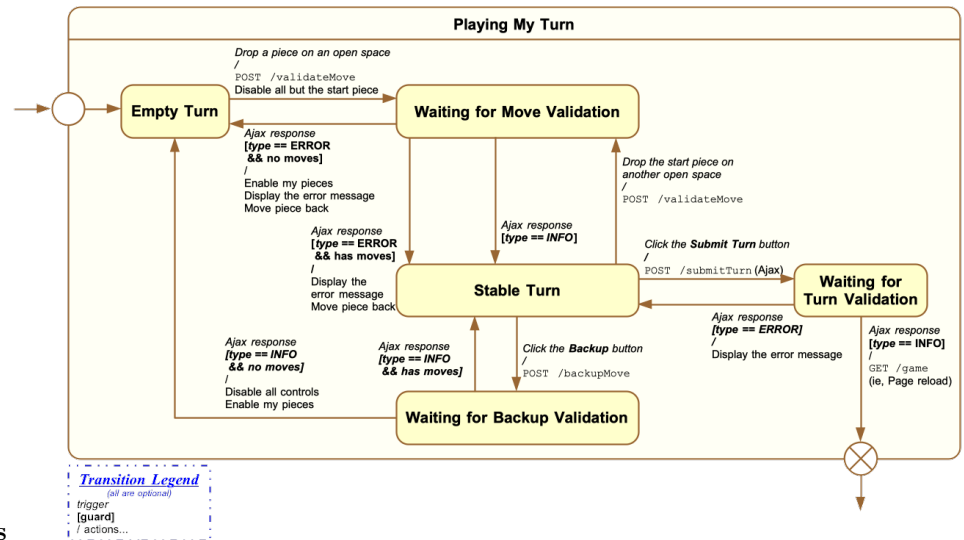
How a turn works
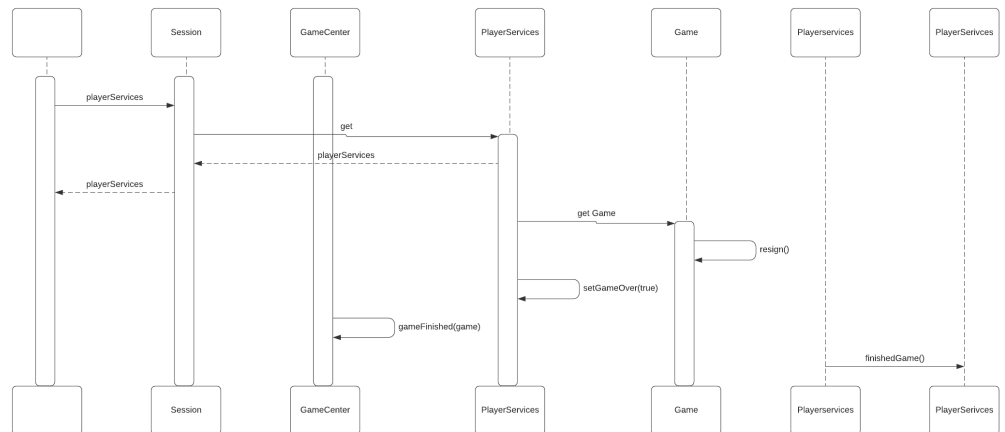
How Resigning works

Application Tier

The Application tier validates the player objects and manages the games and the players for the checker game. The Application tier is the main source of player and game creation logic. The functionality of Application tier includes: - It checks for the validity of the user name. - It keeps a list of users currently playing the game.
- It checks if the user name is on the list of the current players. - It adds the new player to the list of current players and - It checks to see if a new player signed in and faciliates the converstaion between the tiers that a new player joined in. - It checks to see if the player is playing a game or not, it ends that session of game which has no players in it. - It also keeps track of the number of games that is being currently played, and tells the server to start a new game when the players are signed in and ready to play.
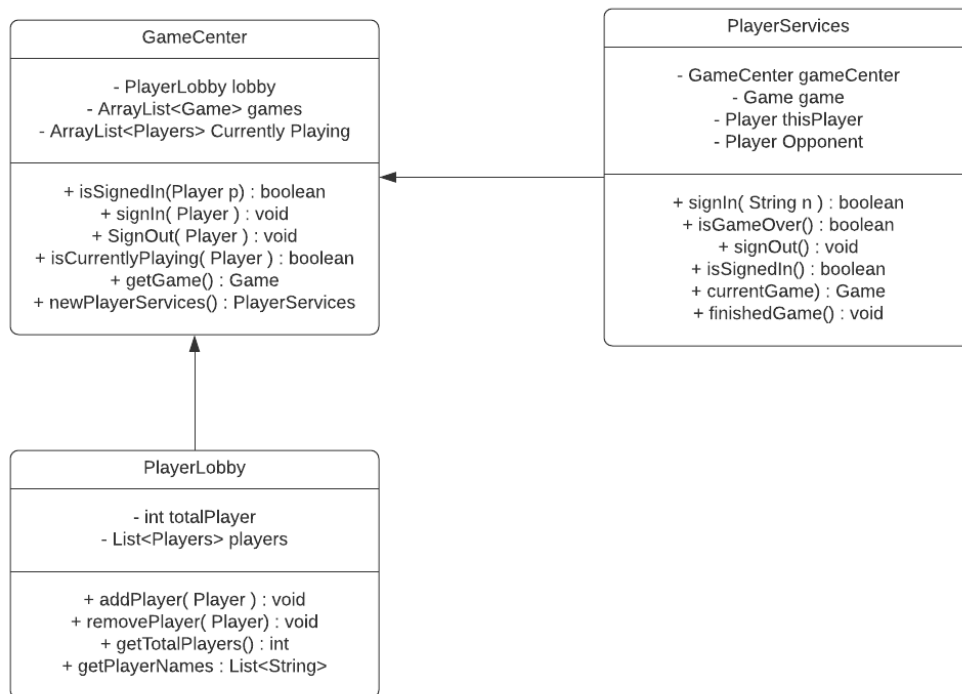
5

GameCenter

- PlayerLobby lobby
- ArrayList<Game> games
- ArrayList<Players> Currently Playing

+ isSignedIn(Player p) : boolean
+ signIn(Player) : void
+ SignOut(Player) : void
+ isCurrentlyPlaying(Player) : boolean
+ getGame() : Game
+ newPlayerServices() : PlayerServices

PlayerServices

- GameCenter gameCenter
- Game game
- Player thisPlayer
- Player Opponent

+ signIn(String n) : boolean
+ isGameOver() : boolean
+ signOut() : void
+ isSignedIn() : boolean
+ currentGame) : Game
+ finishedGame() : void

PlayerLobby

- int totalPlayer
- List<Players> players

+ addPlayer(Player) : void
+ removePlayer(Player) : void
+ getTotalPlayers() : int
+ getPlayerNames : List<String>

Figure 4: Application Class Diagram

Model Tier

The Game classes offers a way for checking if a player has and legal turns by doing this process - Check if it is the red or white player we are checking - if white player: go through each piece and see if there is a valid move for it to make - if red player: flip board and do the same thing. algorithm only works one way

The Model Tier is responsible for the domain enitites and the logic of the game. It communicates with the UI Tier and check if the game rules are being correctly adhered. It has CheckerBoard, Checker, Game, Move, Player, Position and Square classes. The CheckerBoard class creates a checkerboard for the players, and utilizes the checker and square to present a fully functional checker board. The move class is a place holder for the position and players of the Model Tier classes. The Checkerboard class contains the logic for the simple move, single jump, multiple jump moves and promotion to king. The game class starts the game for the player after the checkerboard is set, and when a move is made across the board, the move class checks for the validity of the move made by the user. If the move is right, depending on the type of move then the captured piece was removed from the checkerboard. The checkerboard class populates the board with the squares and checkers to present a fully functional checker board for the players to play game. It decides the color of the square and the checker piece on the board. It also checks for the move made by the user and if it is valid or not. The class also takes in account if the all the pieces were captured or if there were any moves left on the board. It passes this information to the wrapper classes which would then communicate it with the UI Tier classes and javascript for further implementation.
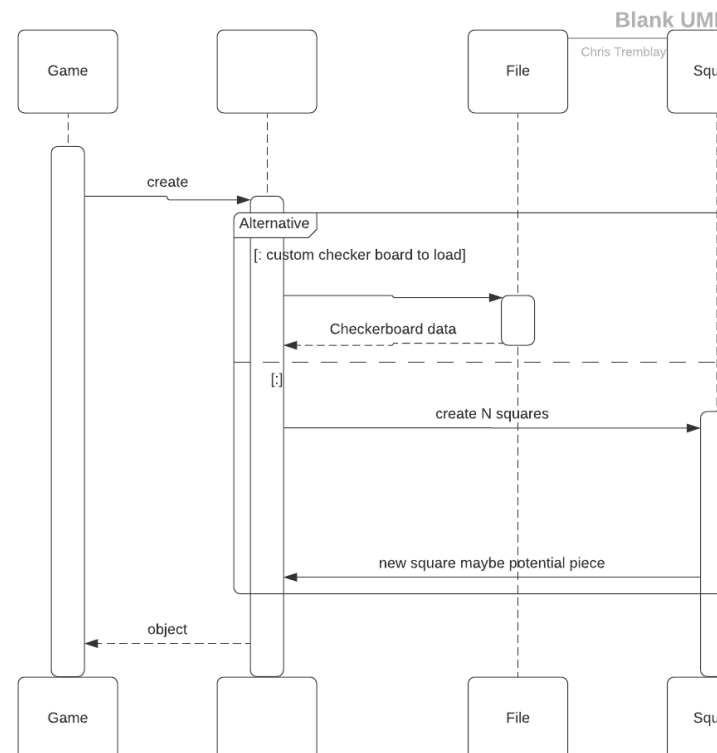
Creating a Game object with a custom board

Design Improvements

The improvement would be to remove the duplicate classes from the UI Tier such as Row, Spaces, Pieces, and Boardview since it violates the Object Oriented Design Principles. Since these classes have already been implemented in Model Tier classes. Most of the classes could be cleaned up and have small helper classes to have low coupling. The checkerboard class has the most of the logic of the game. This class can be decomposed into smaller helper classes such as single, single jump and multiple jump move classes. This would result in low coupling and high cohesion since the checker class would have only access to the object of the move classes. The design could be more efficient if the king logic would be broken into smaller helper classes such as promotion to king. The smaller classes would allow easier debugging when a fault arises.

Testing

Acceptance Testing

13 Stories have passed acceptance testing. A lot of the issues we found when test were signing in a out a lot could get annoying and slow down testing. Potential changes for auto sign in for testing might come. - current issue of validating king moves properly

Unit Testing and Code Coverage

90% of our code is covered. Anywhere from 90%-95% will be good for the team because there are lines of code, classes, or functions that cannot be reached by unit testing or wouldn't benefit from testing. For example *WebServer* and *Applicatiion*. Our strategy was to really hammer down on the model and ui for testing since that's where a lot of the where most of the logic is happening in the program.

Web Checkers a'la Spark/Java8

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.webcheckers.model		92%		81%	44	210	40	405	2	65	0	11
com.webcheckers		0%		0%	8	8	34	34	6	6	1	1
com.webcheckers.ui		93%		81%	15	90	29	329	3	53	0	19
com.webcheckers.application		92%		75%	14	56	14	126	3	34	0	3
com.webcheckers.util		90%		0%	2	10	1	14	1	9	0	2
Total	404 of 4,126	90%	84 of 414	79%	83	374	118	908	15	167	1	36

Total Coverage

com.webcheckers.model

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Checkerboard		90%		78%	32	114	24	208	0	15	0	1
Game		94%		89%	9	60	11	126	0	19	0	1
Square		80%		100%	1	9	4	17	1	8	0	1
Checker		93%		n/a	1	6	1	12	1	6	0	1
Position		100%		83%	1	7	0	10	0	4	0	1
Move.Type		100%		n/a	0	1	0	4	0	1	0	1
Move		100%		n/a	0	5	0	10	0	5	0	1
Player		100%		100%	0	5	0	9	0	4	0	1
Checker.Type		100%		n/a	0	1	0	3	0	1	0	1
Checker.Color		100%		n/a	0	1	0	3	0	1	0	1
Square.Color		100%		n/a	0	1	0	3	0	1	0	1
Total	148 of 1,945	92%	53 of 290	81%	44	210	40	405	2	65	0	11

Model Coverage

com.webcheckers.application

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classe
PlayerServices		82%		72%	7	21	9	47	2	12	0	
GameCenter		97%		72%	5	22	3	53	0	13	0	
PlayerLobby		95%		87%	2	13	2	26	1	9	0	
Total	39 of 519	92%	11 of 44	75%	14	56	14	126	3	34	0	

Application Coverage

com.webcheckers.ui

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
PostSignOutRoute		71%		83%	1	6	11	28	0	3	0	1
PostSignInNameRoute		87%		83%	4	16	5	42	0	4	0	1
Space		73%		50%	4	10	2	13	1	5	0	1
GetGameRoute		93%		92%	1	10	6	60	0	3	0	1
PostCheckTurnRoute		88%		66%	2	6	3	17	0	3	0	1
Row		96%		n/a	1	4	1	15	1	4	0	1
BoardView		95%		n/a	1	3	1	13	1	3	0	1
GetHomeRoute		100%		100%	0	7	0	29	0	3	0	1
WebServer		100%		n/a	0	3	0	21	0	3	0	1
PostSubmitTurnRoute		100%		75%	1	5	0	21	0	3	0	1
PostValidateMoveRoute		100%		100%	0	4	0	16	0	3	0	1
PostResignRoute		100%		n/a	0	3	0	17	0	3	0	1
GetSignInRoute		100%		n/a	0	3	0	11	0	3	0	1
PostBackUpMoveRoute		100%		n/a	0	3	0	12	0	3	0	1
GetGameRoute.ViewMode		100%		n/a	0	1	0	2	0	1	0	1
Piece.Type		100%		n/a	0	1	0	2	0	1	0	1
GetGameRoute.ActiveColor		100%		n/a	0	1	0	2	0	1	0	1
Piece.Color		100%		n/a	0	1	0	2	0	1	0	1
Piece		100%		n/a	0	3	0	6	0	3	0	1
Total	94 of 1,456	93%	14 of 74	81%	15	90	29	329	3	53	0	19

UI Coverage