
geometry: margin=1in

PROJECT Design Documentation

Team Information

- Team name: Team E
- Team members
 - Ronald Torrelli
 - Cameron Riu
 - Evan Price
 - Mathew Klein

Executive Summary

The application must allow players to play checkers with other players who are currently signed-in. The game user interface (UI) will support a game experience using drag-and-drop browser capabilities for making moves.

Purpose

The purpose of the WebCheckers application is to allow a group of users to play a game of checkers with each other online.

Glossary and Acronyms

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

Requirements

This section describes the features of the application.

Signign In

Start Game

Checker Board

Checker Pieces

Resignation

Game Play

Replay

Spectator Mode

Definition of MVP

The MVP is the most basic form of the program. Having only the most basic of features for time of release.

MVP Features

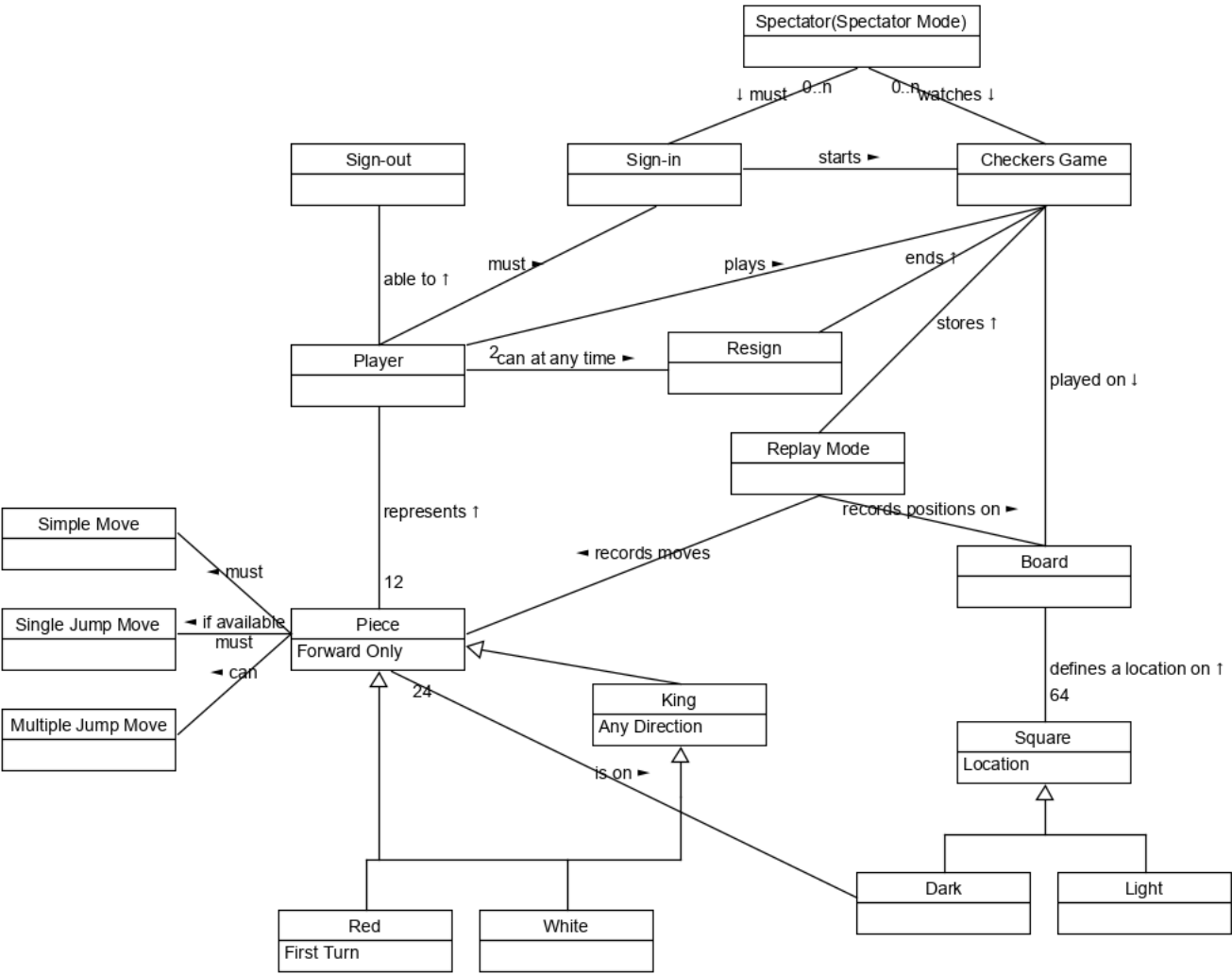
The MVP features are the ability to sign in, select an opponent, start a game, perform basic checkers moves (Move, Single and multi-jump, kinging and winging/losing).

Roadmap of Enhancements

We plan to include two top-level enhancements to our program, Spectator Mode and Replay Mode. We plan to implement them in that order as well.

Application Domain

This section describes the application domain.



The application domain of WebCheckers is split up into an application, model, and UI tier to differentiate responsibilities of certain classes. The application tier is where we handle the games being played and control

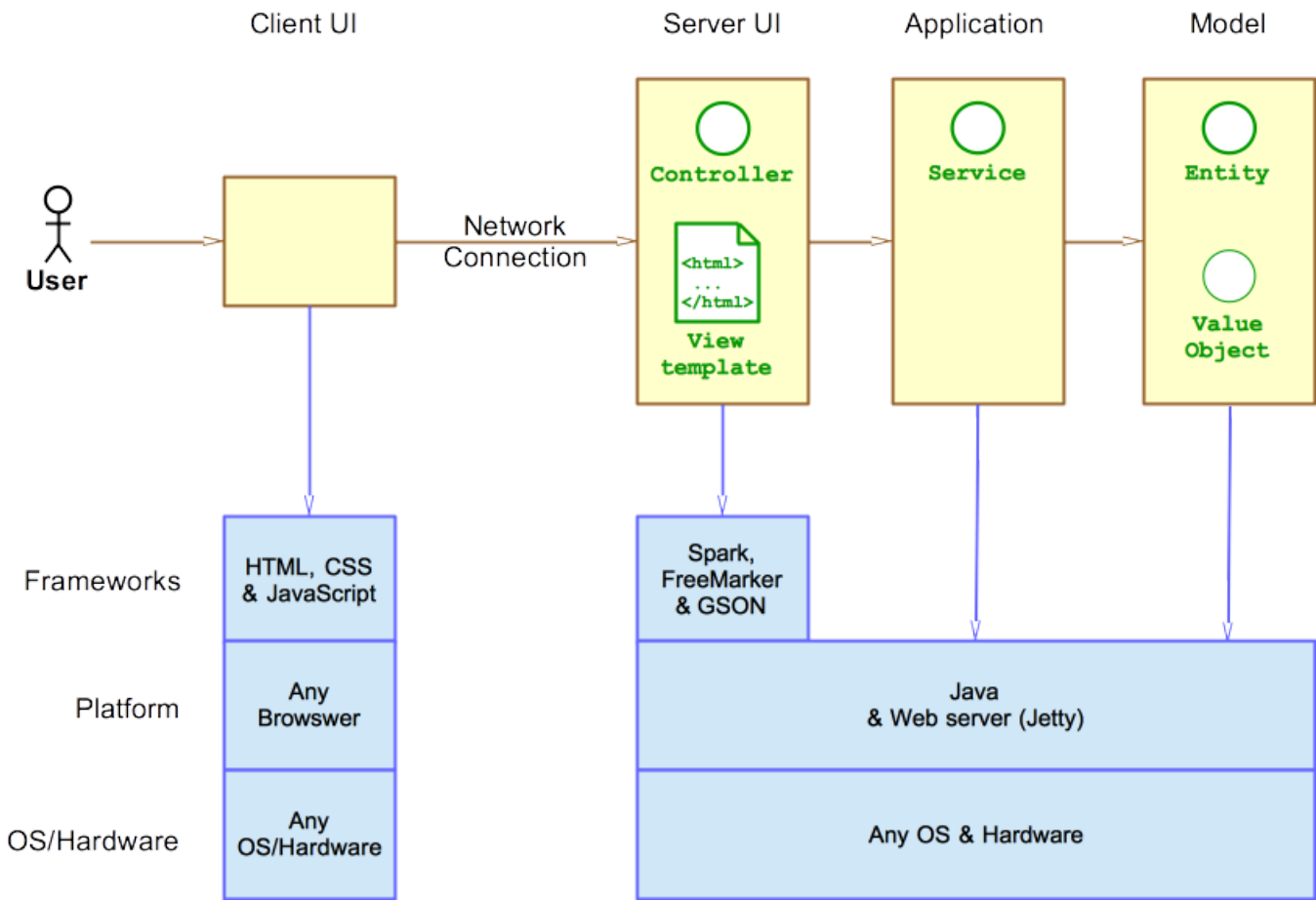
players in and out of a game. The model tier controls the actual game play such as making the board and making moves. The UI tier is responsible for the web server itself and each route that a user can play. Each route is dependent on whether the user has signed in or not.

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.

Overview of User Interface

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























The server is run on the WebServer class and contains GET and POST route calls, starting with GetHomeRoute which renders the home page. When a player signs in they call GetSignInRoute to allow them to see the sign in page and join the server with a valid name which calls PostSignInRoute that renders the home page and shows all other player names that are online instead of just the number of players. Players can sign out as well and that calls PostSignOutRoute.









































Once a player is signed in they can click on another players name to call GetGameRoute which displays the checkers board. The player that was clicked on will automatically call GetGameRoute from the home page after they are clicked on and will be displayed a version of the board.

After both players have joined the game the red player can make a move first. The white player will call PostCheckTurnRoute which will not allow them to make a move until the red player makes a move which calls PostValidateMoveRoute that verifies whether the move is allowed to be made, and then submits the move which calls PostSubmitMoveRoute and makes the move on the server.

Once a player has moved a piece they can use the Backup button to call PostBackUpMoveRoute which returns the piece. The player also has the option to press the Resign button that calls PostResignRoute and removes the player from the game. When the game is over, PostGameOverRoute is called that displays the proper messages to each player.

Application Tier

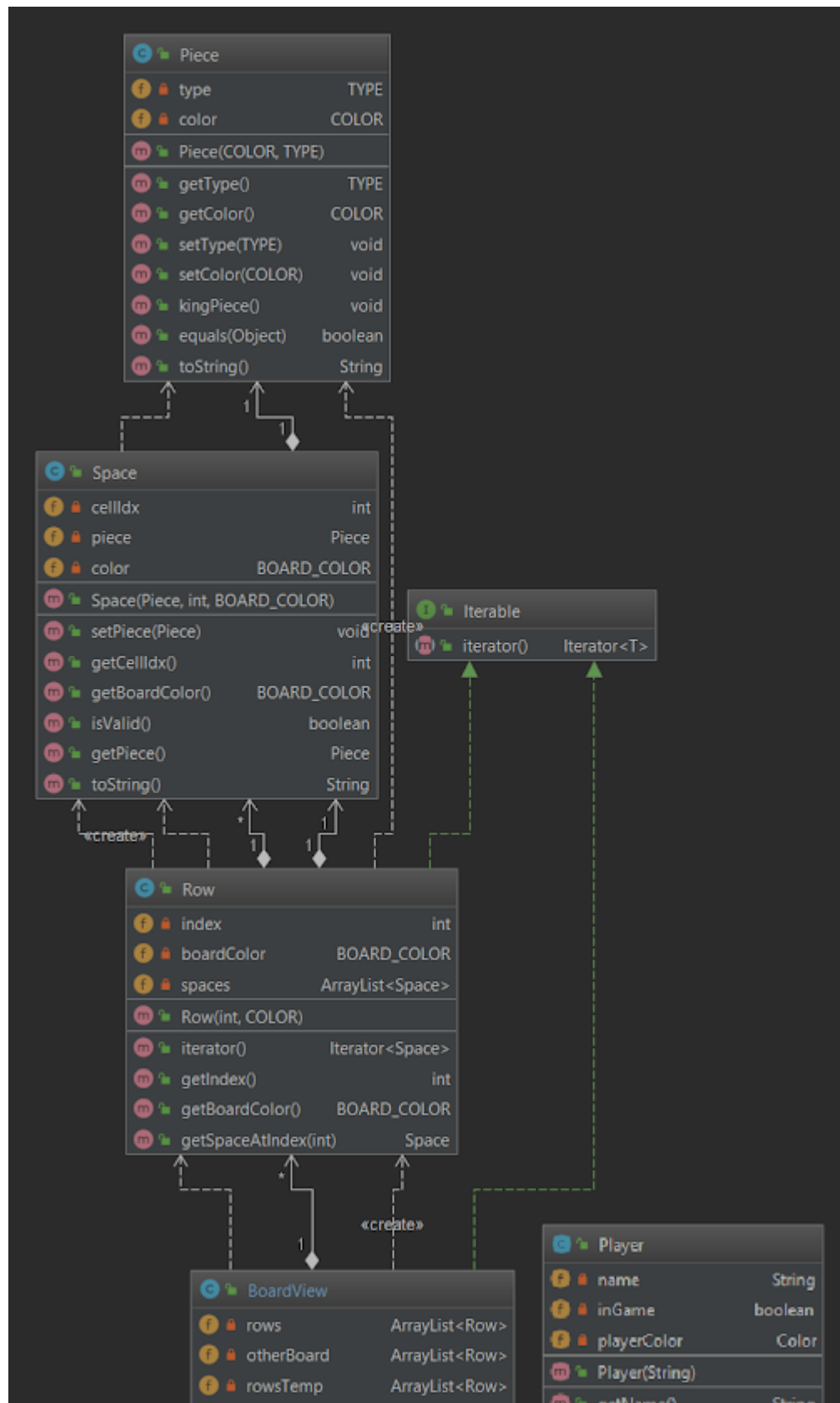
		GameCenter	
		activeGames	HashMap<Player, Game>
		dormantGames	HashMap<Player, Game>
		GameCenter()	
		newGame(Player, Player)	Game
		endGame(Player)	Game
		getGame(Player)	Game
		getActiveGame(Player)	Game
		getDormantGames(Player)	Game
		gameIsActive(Game)	Boolean
		gameIsDormant(Game)	Boolean

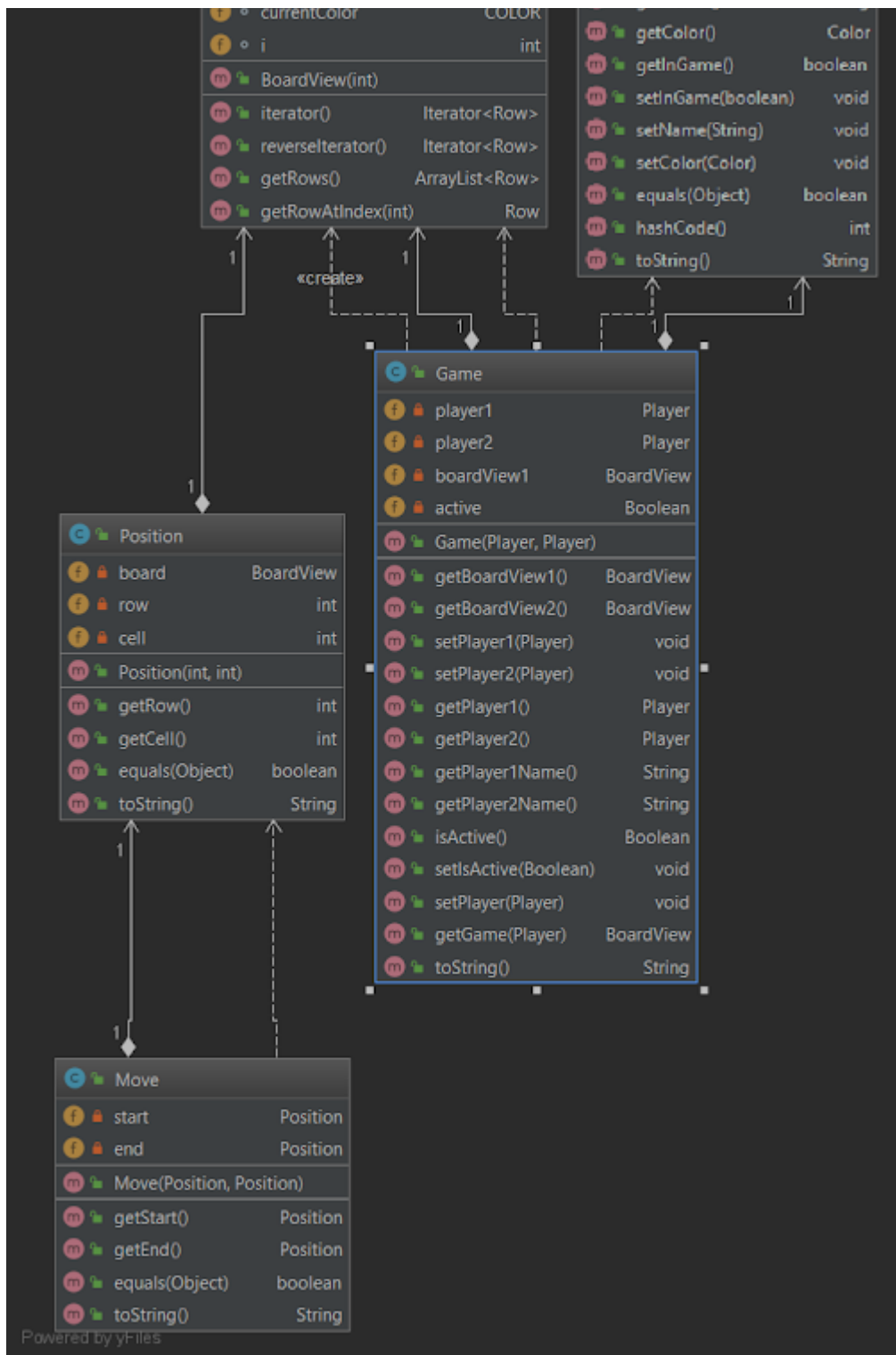
		PlayerLobby	
		players	ArrayList<Player>
		invalidName	boolean
		game	Game
		gameCenter	GameCenter
		vm	Map<String, Object>
		turn	boolean
		PlayerLobby(GameCenter)	
		addPlayer(Player)	boolean
		setInvalidName(boolean)	void
		getInvalidName()	boolean
		getPlayers()	ArrayList<Player>
		getAvaPlayers()	ArrayList<Player>
		setGame(Game)	void
		getGame(Player)	Game
		getGameCenter()	GameCenter
		setMap(Map<String, Object>)	void
		getMap()	Map<String, Object>
		setTurn(boolean)	void
		getTurn()	boolean

Powered by yFiles

The Application tier of the WebCheckers application is responsible for handling the games that are being run on the server. We have a GameCenter class that acts as the hub for all the games on the server. It contains a list of active and dormant games that can be accessed by providing a player in one of those games. The PlayerLobby class is where we store player information on the server. Every player that signs into the server will only be successfully added to the lobby if they enter a valid name. The instance of GameCenter is stored in PlayerLobby in order to access from the UI tier.

Model Tier





The Model tier of the WebCheckers application is responsible for the rules and moves of a game of checkers. The BoardView class is how the player would view the board based on whether they are the red or white player. The Game class allows us to access the elements of a game such as the players and board view. The Row class sets up the rows of the board with Spaces that contain a piece, an index, and the color of the space. The Move class uses Positions both before and after to verify specific moves made by the player. The Player class is where we store player information such as name and color.

Design Improvements

If the project were to continue, we would improve the design of our GameCenter class. As it stands, this class performs most of the game logic for multiple games, however; some of that responsibility is also shared with

the PlayerLobby class. If we were to continue with this project, we would remove some of the game logic within PlayerLobby and put it into GameCenter. The metrics of our code show that the PostValidateMoveRoute contains a heavy amount of logic. This logic could be improved by being placed elsewhere to allow the PostValidateMoveRoute to simply focus on performing after those conditions are met, not checking those conditions itself.

Testing

This section will provide information about the testing performed and the results of the testing.

Acceptance Testing

Number of stories that passed every acceptance criteria test: 0

Number of stories with some failed acceptance test: 3

Number of stories with no testing: 5


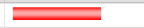
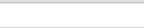

























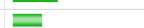








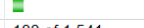


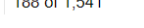
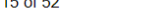



Some issues to note: The other player's board will revert to the first players board after the first player submits their turn. The moves made by a player do not stay on the board after the move is submitted.

Unit Testing and Code Coverage

####UI Tier Code Coverage

 Web Checkers a la Spark/Java8 >  com.webcheckers.ui

com.webcheckers.ui


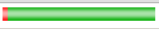
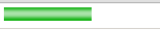










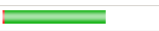















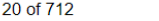


Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
 WebServer		0%		n/a	4	4	23	23	4	4	1	1
 GetGameRoute		90%		50%	4	8	5	50	0	3	0	1
 PostValidateMoveRoute		94%		62%	6	13	3	41	0	5	0	1
 GetHomeRoute		93%		78%	4	12	0	33	1	5	0	1
 PostSignInRoute		93%		83%	1	6	2	25	0	3	0	1
 GameData		96%		n/a	1	11	2	37	1	11	0	1
 PostGameOverRoute		100%		n/a	0	3	0	26	0	3	0	1
 PostSubmitTurnRoute		100%		100%	0	4	0	24	0	3	0	1
 PostCheckTurnRoute		100%		100%	0	4	0	20	0	3	0	1
 PostResignRoute		100%		n/a	0	3	0	19	0	3	0	1
 GetSignInRoute		100%		100%	0	4	0	17	0	3	0	1
 PostBackupMoveRoute		100%		n/a	0	3	0	12	0	3	0	1
 PostSignOutRoute		100%		n/a	0	2	0	9	0	2	0	1
 ResponseMessage.MessageType		100%		n/a	0	1	0	1	0	1	0	1
 ResponseMessage		100%		n/a	0	5	0	7	0	5	0	1
Total	188 of 1,541	87%	15 of 52	71%	20	83	35	344	6	57	1	15

(Note that the WebServer's code coverage is 0% because although it is part of the UI tier, testing it was not necessary)



####Model Tier Code Coverage

 Web Checkers a'la Spark/Java8 >  com.webcheckers.model


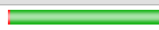
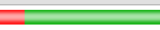

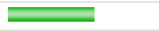

com.webcheckers.model

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
 BoardView		96%		100%	1	11	1	22	1	7	0	1
 Row		95%		100%	1	12	1	20	1	5	0	1
 Move		94%		62%	3	9	0	12	0	5	0	1
 Position		92%		75%	2	9	0	11	0	5	0	1
 Player		97%		75%	1	12	0	20	0	10	0	1
 Piece		97%		75%	2	12	0	17	0	8	0	1
 Game		100%		100%	0	16	0	27	0	14	0	1
 Space		100%		100%	0	9	0	12	0	7	0	1
 Player.Color		100%		n/a	0	1	0	2	0	1	0	1
 Piece.COLOR		100%		n/a	0	1	0	2	0	1	0	1
 Space.BOARD_COLOR		100%		n/a	0	1	0	2	0	1	0	1
 Piece.TYPE		100%		n/a	0	1	0	2	0	1	0	1
Total	20 of 712	97%	8 of 58	86%	10	94	2	149	2	65	0	12

####Application Tier Code Coverage

 Web Checkers a'la Spark/Java8 >  com.webcheckers.application

com.webcheckers.application

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
 GameCenter		97%		84%	4	21	4	46	0	8	0	1
 PlayerLobby		100%		100%	0	17	0	36	0	12	0	1
Total	5 of 353	98%	4 of 36	88%	4	38	4	82	0	20	0	2