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
AI	Artificial Intelligence

Requirements

This section describes the features of the application.

- Signign In - A user can log in with a valid user name and then be able to view all of the other players currently logged in as well. A valid username is one that only contains alphanumeric characters.
- Start Game - A game can be started between two players, player 1 chooses player 2 from the home screen and then start a game
- Checker Board - A game board can be made with alternating black and whites spaces and that is 8x8
- Checker Pieces - Pieces can made and placed onto a game board. Pieces also have a type, single and king with their own restrictions.
- Resignation - Either player when a game is active can resign a game, this in turns makes the player that did not resign the winner of the game
- Game Play -
  - Simple Move - The active player has the ability to move their pieces to a valid position
  - Single Jump - The active player has the ability to make a valid jump

  - Multiple Jump - The active player can make valid multi-jumps
  - King - Any piece that is moved to the opposite side of the board then a piece will be kinged and has the ability to move backwards as well as forwards
  - Wining/Losing - The players have a way to lose and win, that is if all of pieces of the opponent have be captured, a player resigns or no more moves can be made.
- Move Helper - During a game the active player can use the help button to show them a jump if there or a move that can be taken.
- AI Player - After a player has signed in they may choose the AI player to player against the computer in a round of checkers.

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

- Signing in
- Start a game
- Playing a game
- Resigning from a game
- Winning/Losing

Roadmap of Enhancements

We plan to include two top-level enhancements to our program, Move Helper and AI player. We plan to implement them in that order as well.

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

Our design consists of three architectural tiers: Application Tier, Model Tier, and UI tier.

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).

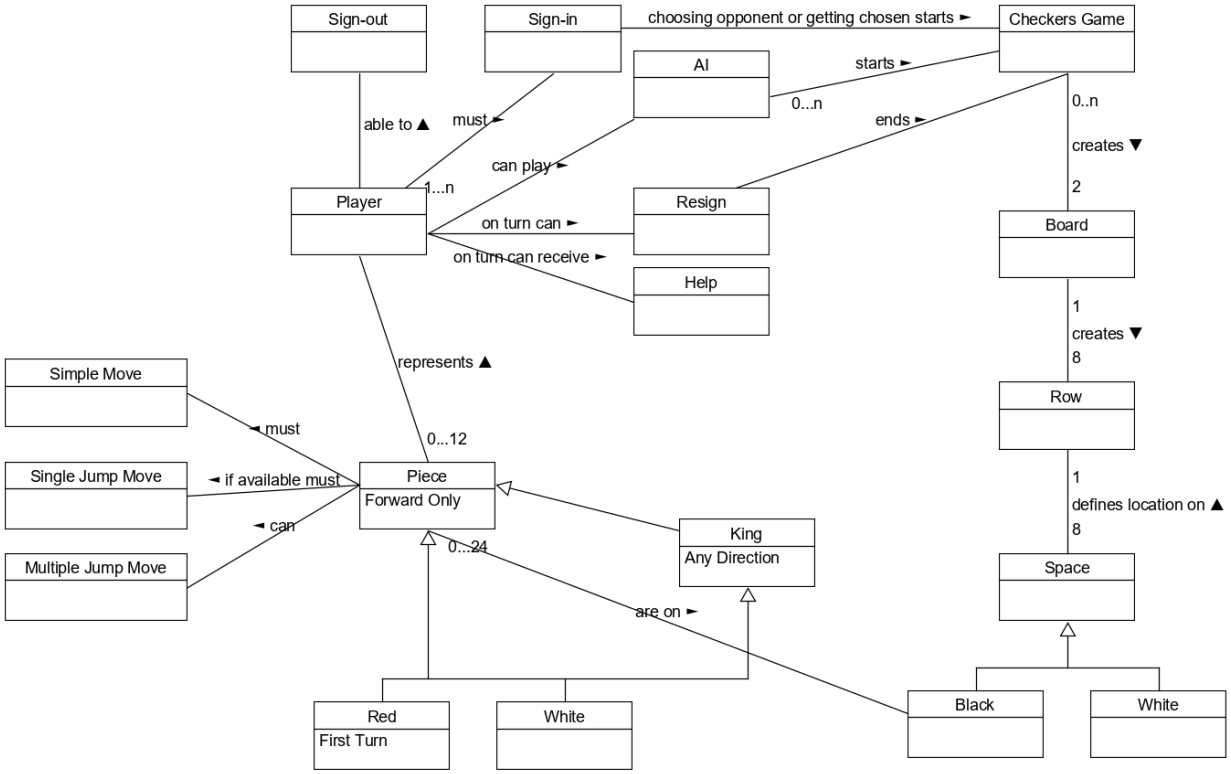

Figure 1: The WebCheckers Domain Model

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 user when first navigating to the home page of WebCheckers will see a login button in the navbar and the number of players currently logged in. After clicking on the sign in button the player will be brought to the sign in page where they may enter a valid username. A valid username is defined as one that is not already in use and that does not contain more than one or only special characters. After signing in the player will see a list of all currently active players.

After the player has selected another player to play against both players are put into the game. After the start of the game the players are able to move pieces, resign, signout, and ask for help from the move help button. Once the game is over both players can click the exit button to be brought back to the home page.

UI Tier

The UI tier of the WebCheckers application is responsible for handling all of the changes to the view of a player on the server. The data for the game that is currently active is stored in a HashMap on the game object. This data is used by the game.ftl that sets up the checkers board based on the VM information.

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

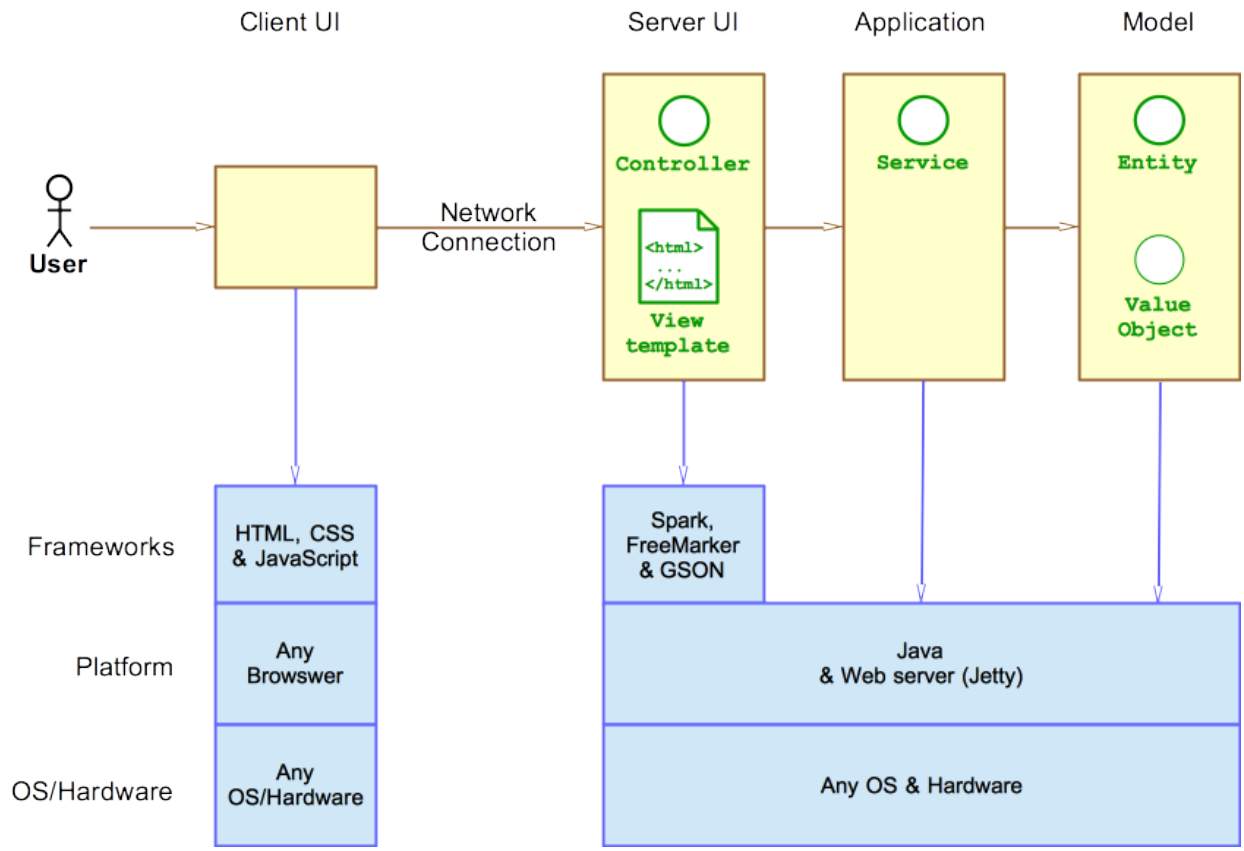

Figure 2: The Tiers & Layers of the Architecture

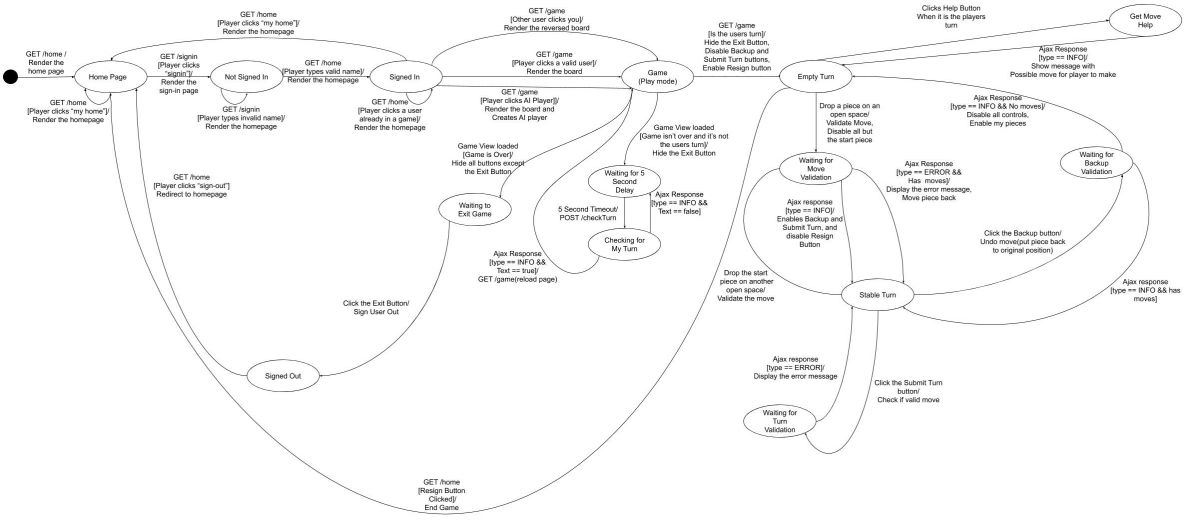

Figure 3: The WebCheckers Web Interface Statechart

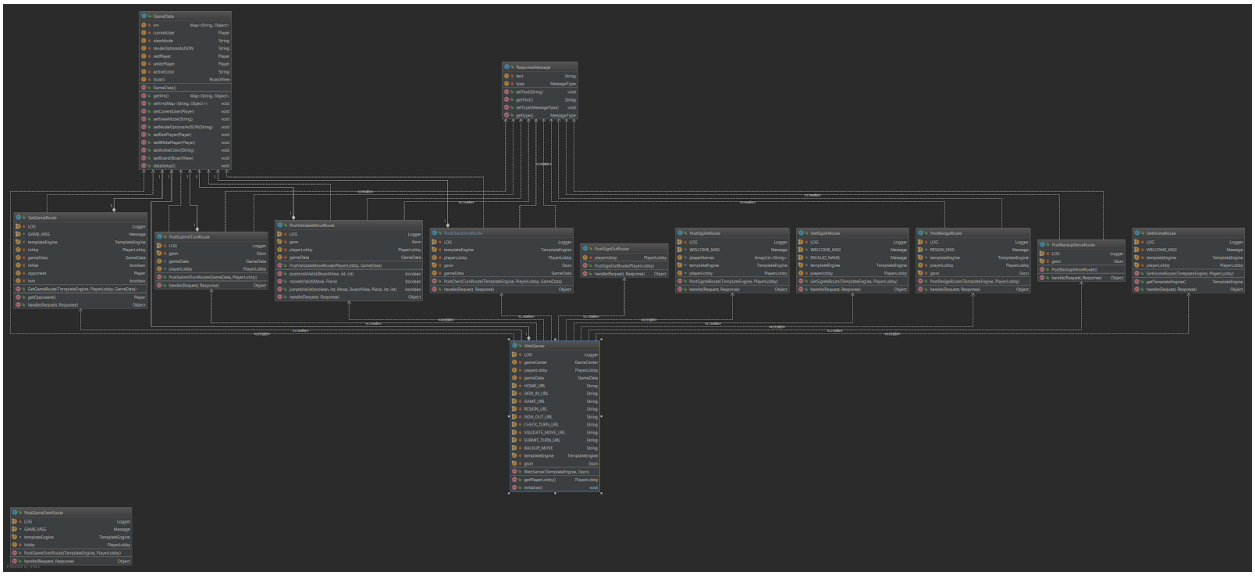

Figure 4: The WebCheckers UI Tier UML Diagram

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. Optionally the player can click on the AI player to play against the computer. The AI player works by taking the first possible jump or if no jumps can be made then the AI will choose the first available move. 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. For player 1 the red pieces are closer to them, and for player 2 the white pieces are closer to them.

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.

During a game the active player can click the help button to have a available move or jumps displayed. This is done by the PostHelpMoveRoute and MoveChecks.

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

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.

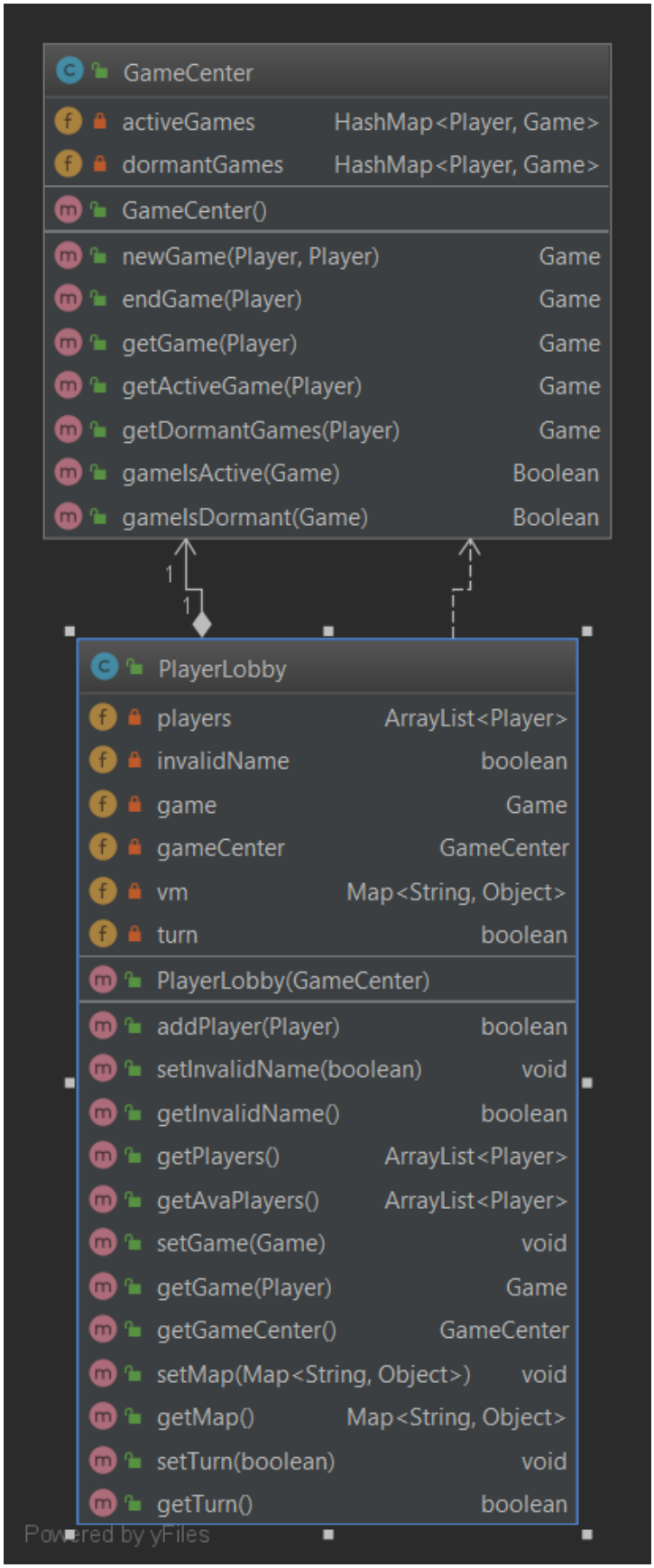

Figure 5: The WebCheckers UI Tier UML Diagram

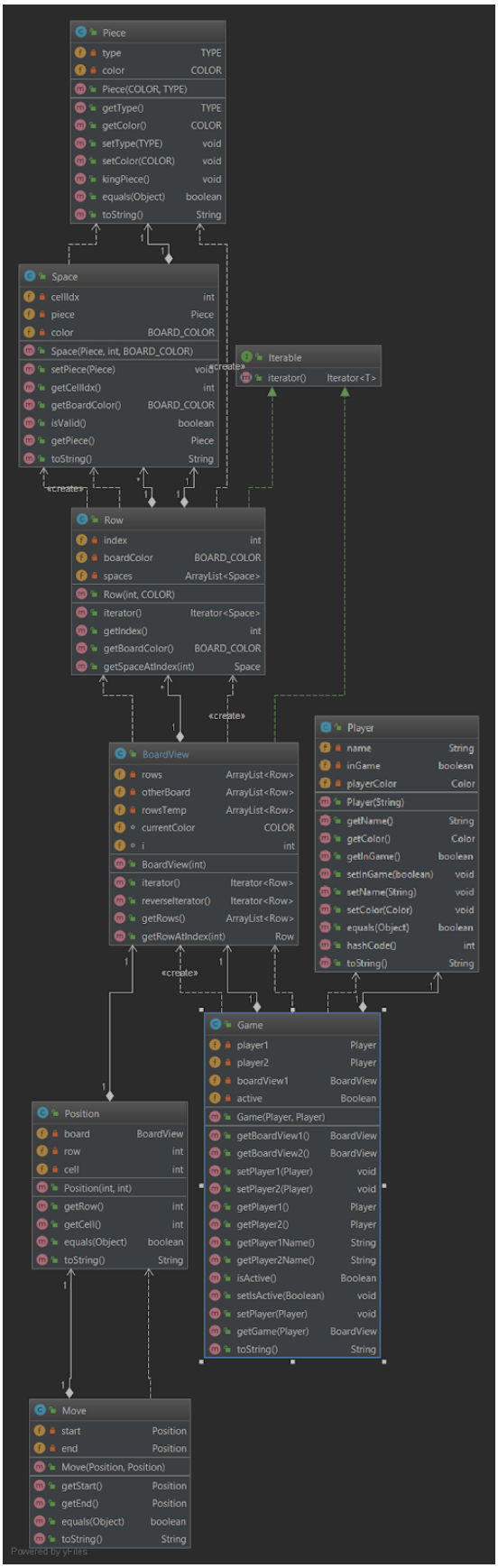

Figure 6: The WebCheckers UI Tier UML Diagram

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

We performed unit testing for all architectural tiers but had some trouble fully testing some of the UI tier classes due to advanced game logic conditions that were very difficult to recreate.

Acceptance Testing

Number of stories that passed every acceptance criteria test: 16

Number of stories with some failed acceptance test: 0

Number of stories with no testing: 0

Unit Testing and Code Coverage

UI Tier Code Coverage

Model Tier Code Coverage

Application Tier Code Coverage

com.webcheckers.ui

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
AIHandler		56%		66%	10	24	25	63	1	6	0	1
PostValidateMoveRoute		58%		29%	12	15	24	54	0	3	0	1
WebServer		0%		n/a	2	2	19	19	2	2	1	1
MoveChecks		84%		77%	13	42	16	109	3	9	0	1
PostSubmitTurnRoute		55%		40%	4	8	15	39	0	3	0	1
GetGameRoute		92%		75%	4	13	3	55	0	3	0	1
MoveHelper		100%		96%	2	31	0	61	0	5	0	1
BoardHandler		100%		92%	2	17	0	46	0	3	0	1
GetHomeRoute		100%		77%	4	13	0	33	0	4	0	1
PostResignRoute		100%		100%	0	5	0	24	0	3	0	1
PostCheckTurnRoute		100%		80%	2	8	0	25	0	3	0	1
PostSignInRoute		100%		100%	0	6	0	24	0	3	0	1
PostSignOutRoute		100%		100%	0	4	0	20	0	2	0	1
PostHelpMoveRoute		100%		100%	0	6	0	20	0	3	0	1
PostBackupMoveRoute		100%		100%	0	4	0	19	0	3	0	1
GetSignInRoute		100%		100%	0	4	0	16	0	3	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	633 of 3,890	83%	65 of 288	77%	55	208	102	635	6	64	1	18

Figure 7: The WebCheckers UI Tier Code Coverage

com.webcheckers.model

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Game		91%		77%	19	69	7	122	0	24	0	1
ValidateMove		94%		85%	11	44	7	80	3	9	0	1
Row		96%		100%	1	13	1	24	1	6	0	1
Position		97%		87%	1	11	0	13	0	7	0	1
Player		97%		75%	1	14	0	24	0	12	0	1
Piece		97%		87%	1	12	0	17	0	8	0	1
BoardView		100%		100%	0	25	0	49	0	21	0	1
Move		100%		87%	1	9	0	12	0	5	0	1
Space		100%		75%	1	12	0	18	0	10	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	93 of 1,937	95%	35 of 214	83%	36	213	15	367	4	106	0	13

Figure 8: The WebCheckers Model Tier Code Coverage

com.webcheckers.application

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
PlayerLobby		96%		88%	2	17	2	39	0	8	0	1
GameCenter		100%		100%	0	13	0	31	0	8	0	1
Total	5 of 299	98%	2 of 28	92%	2	30	2	70	0	16	0	2

Figure 9: The WebCheckers Application Tier Code Coverage