a— geometry: margin=1in —

PROJECT Design Documentation

Team Information

- Team name: NAAN (C)
- Team members
    - Ariana Waldorf
    - Nick Torcello
    - Nicholas Heimbach
    - Amrit Gautam

Executive Summary

This document describes a web application for many users to play a game of checkers. It describes the process required for implementation of the game. The application allows for many games to be running at one time as well as ensures that no one is playing more than one game at a time.

Purpose

The goal of this project is to build an application that allows players to play web based checkers with other players who are currently in the game server. The game supports functionality such as drag and drop for moves for easier user experience. It also supports resignation, single and double jumps and other features that may include an AI player or game specacator

Requirements

This section describes the features of the application.

The checkers games follows a set of requirements that allow it be playable by the american set of rules as well as make it able to play online. 1. It follows the american rules of checker 2. Spectator mode allows other logged in players to view the game 3. replay mode allows a player logged in to replay a completed game

Definition of MVP

The MVP of this product includes having every player sign-in before playing a game and can also sign out after a game is completed. This includes not allowing users to sign-in with the same name. Another aspect is that two players must be able to play a game using the american rules of checkers and the game will check before each submitted move that is adhearing to these rules. The final part of the MVP is that at any point if a player wishes to resign then they may do say, forfeiting the game and returning them to the home page.

MVP Features

1. player sign-in
2. start a game
3. resignation
4. make a move
5. validate move
6. single jump
7. multiple jump
8. end of game
9. king-me
10. waiting for turn

Roadmap of Enhancements

1. spectator mode 2.replay mode

Application Domain

This section describes the application domain.

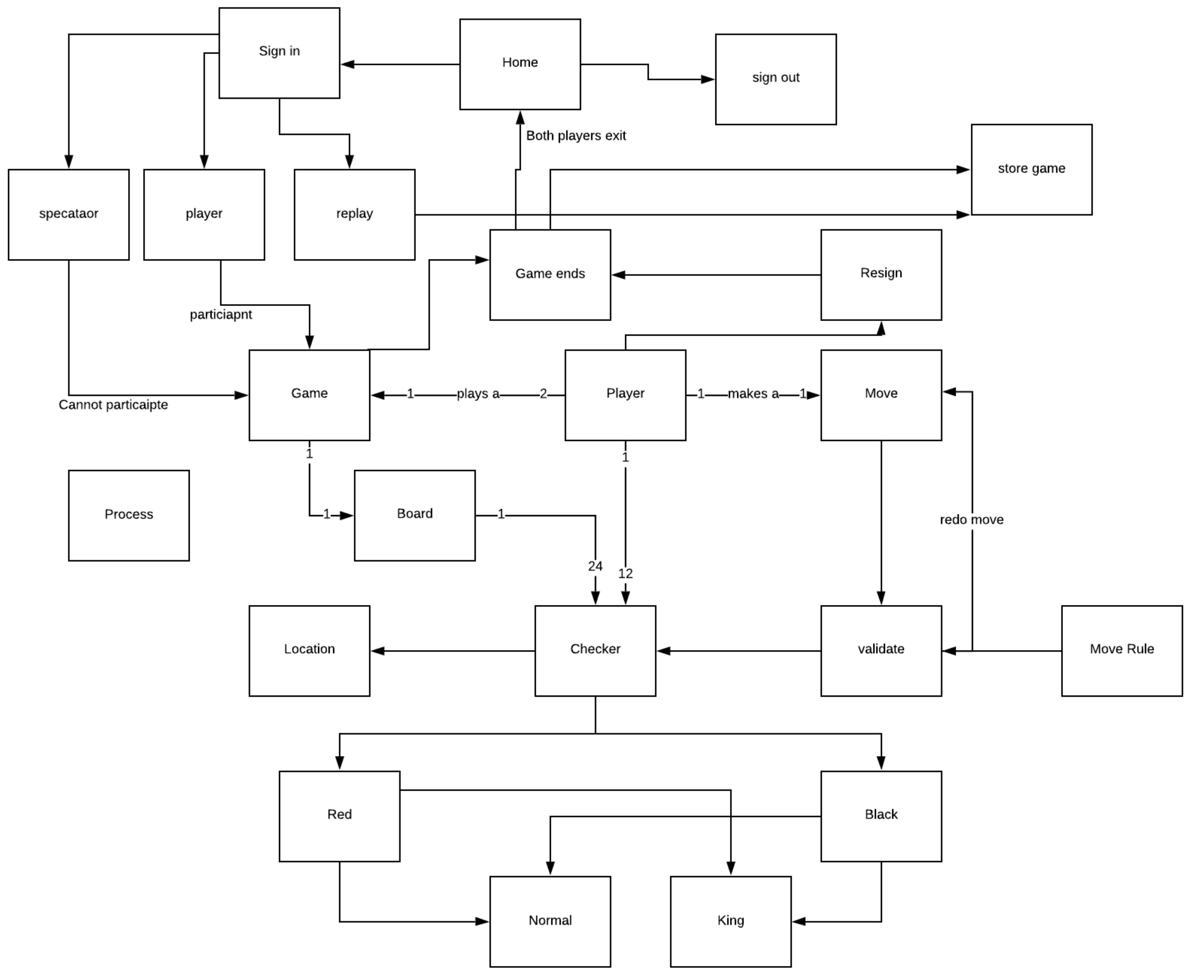


Figure 1: The WebCheckers Domain Model

The domain model provides the common understanding of the application. The table below describes some of the most important behaviors of our domain model.

Term	Definition
Home	Game homepage, the main page players see after entering the URL
Signin	Players sign-in so that they can play a game of checkers.
Signout	If players want to move out of the game they use Signout
Game	Players play the game in a board with checker pieces
Player	The user who controls the game

Term	Definition
board	Game consists a board, which stores checker pieces. The board is 8x8 array
Move	Player makes a move in a board, The rule of a valid and invalid move is followed
Checker	Checker have red or black and a normal or king pieces
Rule	The application follow the American checkers rules

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.

Application's user interface is a detailed behavior of the implementation code. GET and POST routes are used to transition from one page to another.

UI Tier

Looking at the UI section of our project, we strictly broken each important request into its or handler/controller. Looking at all of our UI Component please refer to the chart below of all of the components and their specific function.

GetGameRoute: route that is called when /game is called. Will start a game is no game is already in progress and will play the game.

GetHomeRoute: route that will is called when / is entered or on the homepage. Will handle printing the players and spectator and replay games.

GetReplayGameRoute: route that is called when a user attempts to replay a game. Will initalize the replay game.

GetSignInRoute: route that will redirect the user to /signin

getSpectatorGame: route that will initialize a spectator game.

getStopReplay: route that exit a replay game when the user presses the exit button.

GetStopWatching: route that will stop a spectator game for a player that is currently in spectator mode

PostReplayBackwardRoute: route that is triggered when the user presses the backup button on the replay game.

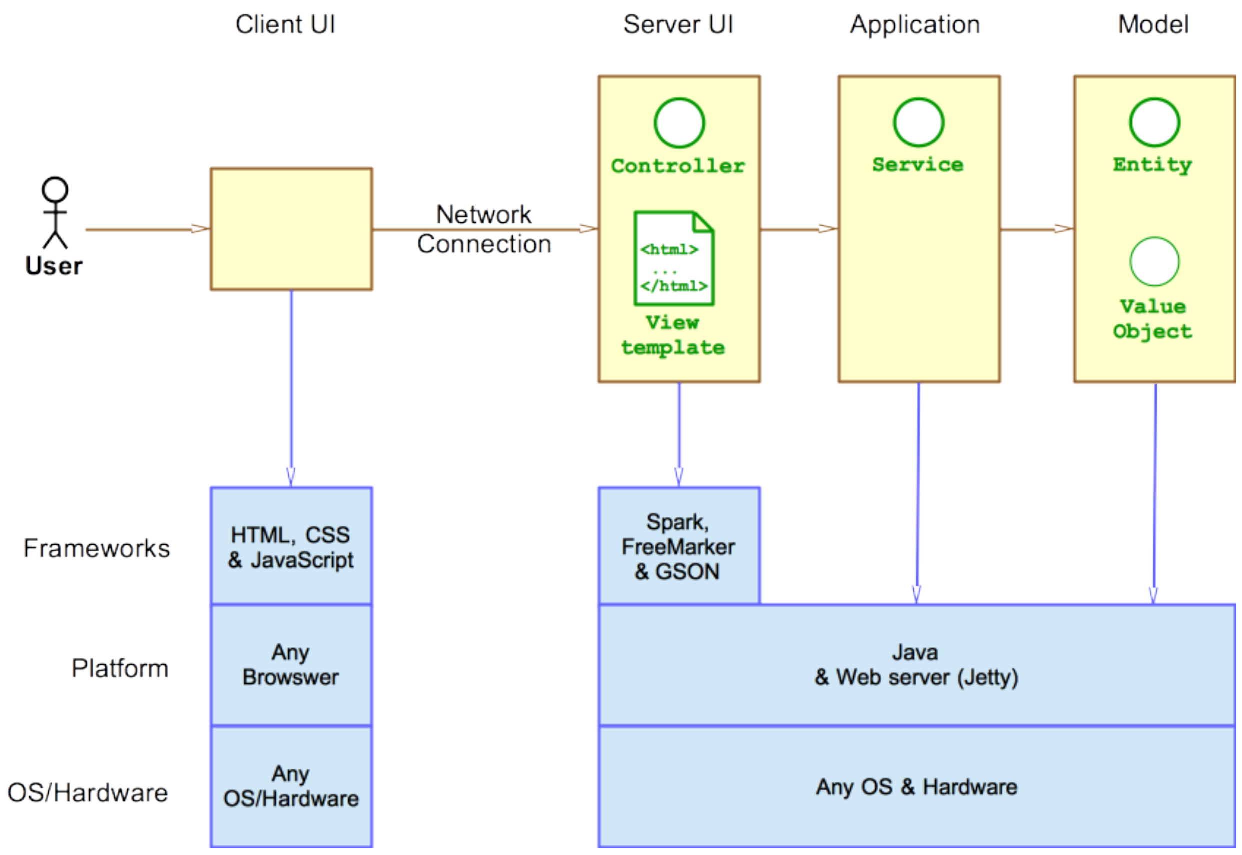
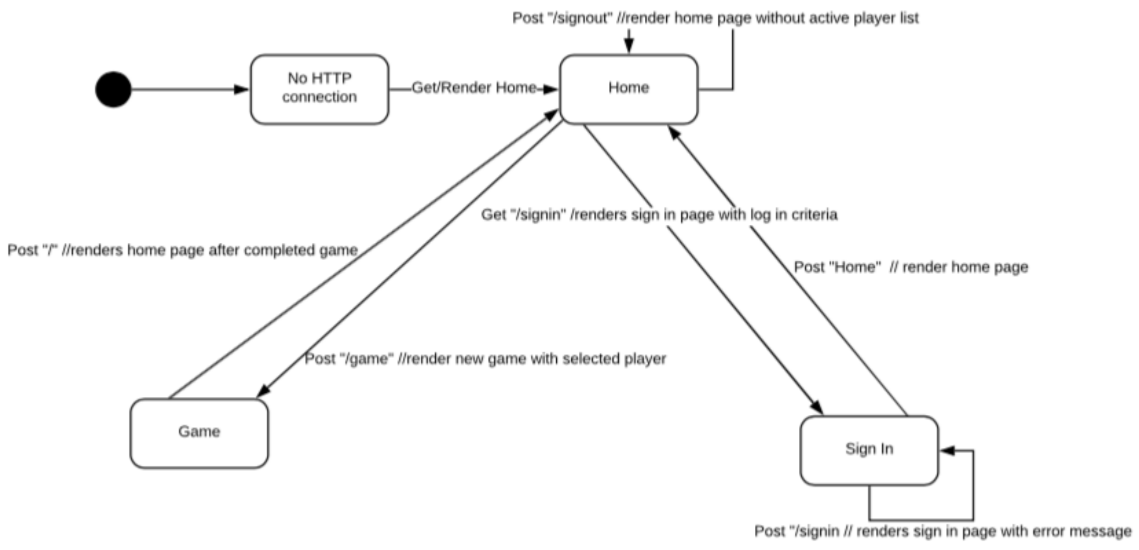


Figure 2: The Tiers & Layers of the Architecture



Post "/signout" //render home page without active player list
No HTTP connection
Get/Render Home
Home
Get "/signin" /renders sign in page with log in criteria
Post "/" //renders home page after completed game
Post "Home" // render home page
Post "/game" //render new game with selected player
Game
Sign In
Post "/signin // renders sign in page with error message

Figure 3: The WebCheckers Web Interface Statechart

PostReplayForwardRoute: route that is triggered when the user presses the forward button on the replay game.

PostSignInRoute: route that is triggered when the user presses the sign in button on the signin page. This will create the user and add them to the correct locations for the purpose of the application.

PostSignOutRoute: route that is triggered when a user that is signed in presses the sign out button.

Each one of these controller/ routes handle their own important factors. This demonstrate the separate of concerns and object oriented design. Separating each class allows there to not be to much of a load on one of these classes. They all do their own part and work together to make a successful application.

Application Tier

While some of these routes expressed down below can be thought of as the UI tier, while they are the routes that the user may click on when routing somewhere different, they do hold some pretty important functionality that is logic base and would belong more to in the application tier.

BackupMoveControl: Strictly handled the backup functionality of the project. when a user selects the backup button on the view end it will trigger this controller and will handle all of the required steps to backing up a move. That may include updating the board, and making sure that the piece and more specificly the board was not changed or updated with the move that was backed up.

CheckTurnControl: Strictly checked if it is the current users turn. This is a Ajax call that is used periodically while the opposite player waits for their to. Contain logic to check if it is the current users turn.

ResignGameControl: route that is called when a user presses the resign button in a game. This controller will handle if the user is allowed to resign or not. If the resign is approved will route the user back to the homepage.

SubmitTurnControl: will validate the move for force jumps and submit the turn, update the boards, and allow the other player to make their turn now.

ValidateMoveControl: will validate a move before it is placed on the board and make sure that it is a valid move.

HandleDoubleJump: Contains all of the logic that is involved in a double jump. It will check if it is a valid double jump and if it is will remove the pieces.

Move: The move class is important for the overall project. This move class separates the concern of having to deal with the positions on the move. As a player moves a piece it convert that into a Move object which can be implemented throughout the application to play the game.

MoveManager: important util class that will manage moves throughout the game

MoveValid: will valid the moves through the game and is used in conjunction with other validation classes like HandleDoubleJump. Both of these classes represent separating functionality into subgroups.

MoveTracker: will keep a backup of a board for the case of replay. This class throughout the game live will store backups that will be used later on when trying to replay a game.

Model Tier

Jsontils: helper class that will convert the class to Json for the use in Ajax.

There are also many different components that can be categorized as in the model tier or also known as the business tier. Some of these components are basic components that help define the business problem such as Game, Piece, BoardView, PlayerLobby, and Space. These component represent more of business problem in the overall problem and without them it would be structured vastly different.

Design & Improvements

Our improved application design has some high principles such as Liskov Substitution principle.Our Boardview class tightly coupled with Move, Piece, Row and Space classes as well as Row Iterator, Space Iterator. The board requires to be an Iterable of Iterables which is being implemented through the Board and Row Iterable<> interface. This is useful when Free marker template wants to Iterate over rows and columns of the board.

Other principles used on our application:

Single responsibility: we focused on making sure each class had a seperate responsibility and none would overlap.

High cohesion: with our focus on single responsibility we also choose to use high cohesion in many areas to keep classes small as well as helped keep the code easy to build onto.

Low coupling: While we used high cohesion in some spots, throughout the project we used low coupling so that different sections would not be reliant on each other

One of the most difficult aspect of the project that we ran into was the implementation of the double and single jump. We were able to get a good solution, but the logic definitely took us sometime to figure out.

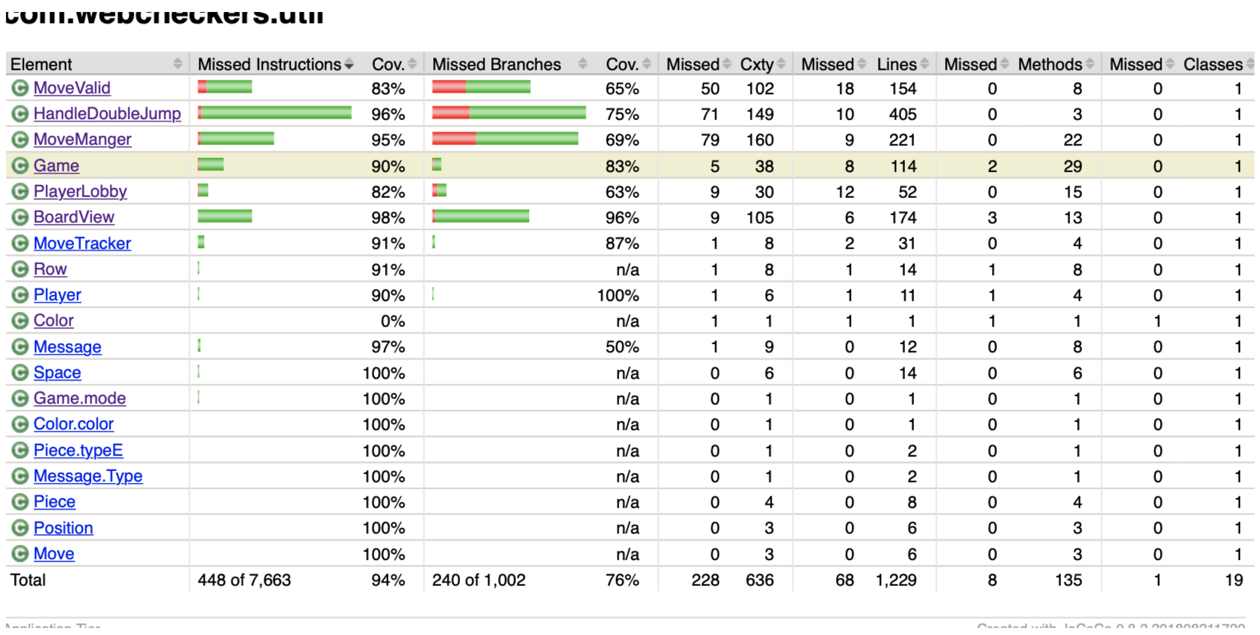
Testing

Acceptance Testing

Most of our user stories from sprint 1 , sprint 2 and sprint 3 team have passed all of their acceptance criteria. we had our own acceptance criteria for sprint 2 and sprint 3. With our own acceptance criteria we were able to develop our own ideas and approaches for each functionality and enhancement. We chose spectator mode and replay mode for our enhancement for sprint 3 and we plan to apply same testing methods for each enhancement.

Unit Testing and Code Coverage

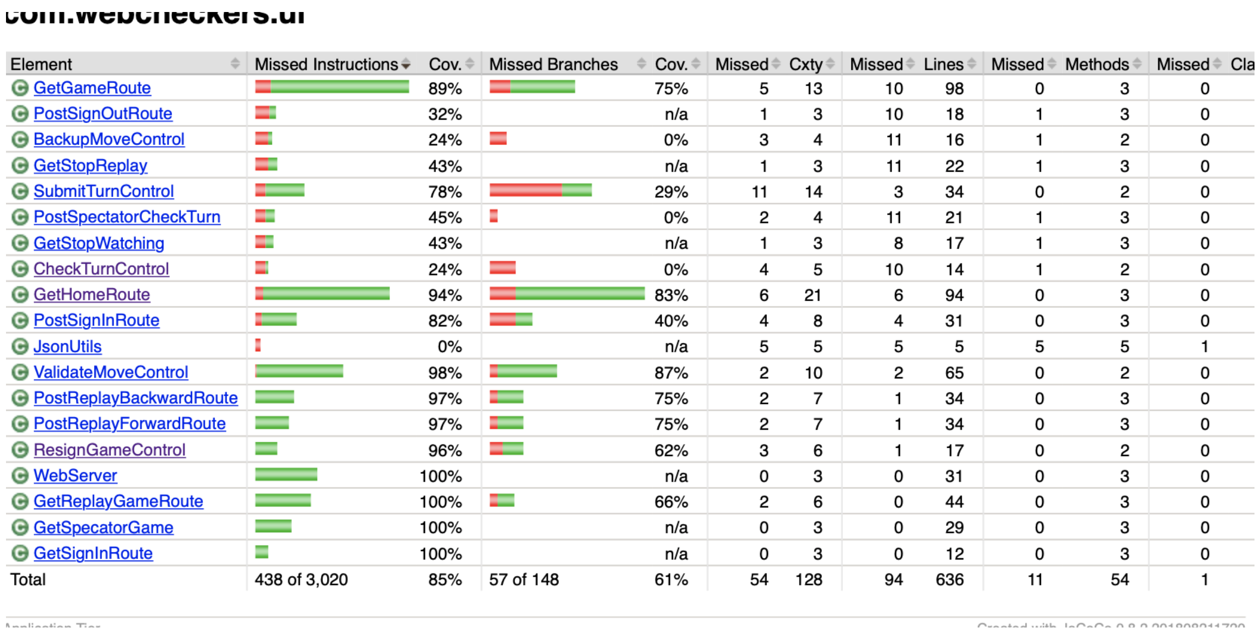
Our strategy for testing was to be as thorough as possible and cover every path through our code. Testing were done to test every methods and logical functions on our code. Our aim was to hit 90% or higher for all tiers. Most of our code was run in testing and ss seen from the results below we do have some branches missed. We were able to hit 95% coverage for model and application tier , 85% coverage for UI tier and 90% overall coverage.



com.webcheckers.util

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
MoveValid		83%		65%	50	102	18	154	0	8	0	1
HandleDoubleJump		96%		75%	71	149	10	405	0	3	0	1
MoveManger		95%		69%	79	160	9	221	0	22	0	1
Game		90%		83%	5	38	8	114	2	29	0	1
PlayerLobby		82%		63%	9	30	12	52	0	15	0	1
BoardView		98%		96%	9	105	6	174	3	13	0	1
MoveTracker		91%		87%	1	8	2	31	0	4	0	1
Row		91%		n/a	1	8	1	14	1	8	0	1
Player		90%		100%	1	6	1	11	1	4	0	1
Color		0%		n/a	1	1	1	1	1	1	1	1
Message		97%		50%	1	9	0	12	0	8	0	1
Space		100%		n/a	0	6	0	14	0	6	0	1
Game.mode		100%		n/a	0	1	0	1	0	1	0	1
Color.color		100%		n/a	0	1	0	1	0	1	0	1
Piece.typeE		100%		n/a	0	1	0	2	0	1	0	1
Message.Type		100%		n/a	0	1	0	2	0	1	0	1
Piece		100%		n/a	0	4	0	8	0	4	0	1
Position		100%		n/a	0	3	0	6	0	3	0	1
Move		100%		n/a	0	3	0	6	0	3	0	1
Total	448 of 7,663	94%	240 of 1,002	76%	228	636	68	1,229	8	135	1	19

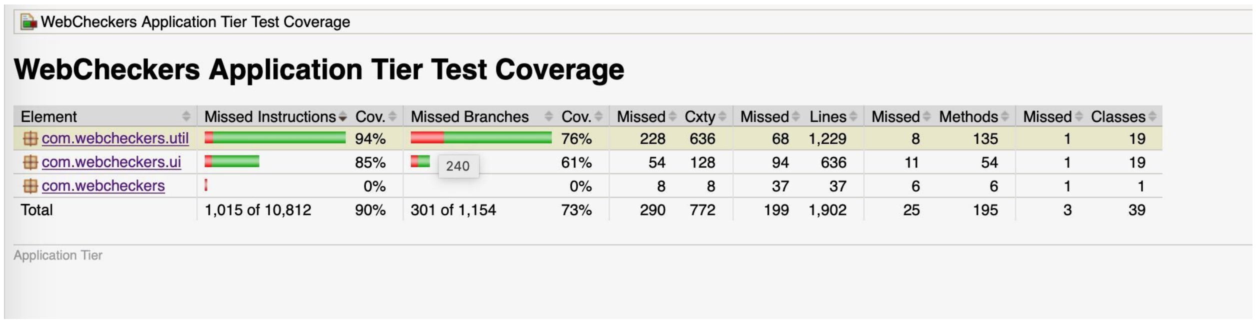
Figure 4: Model and Application Code Coverage



com.webcheckers.ui

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Cla
GetGameRoute		89%		75%	5	13	10	98	0	3	0	
PostSignOutRoute		32%		n/a	1	3	10	18	1	3	0	
BackupMoveControl		24%		0%	3	4	11	16	1	2	0	
GetStopReplay		43%		n/a	1	3	11	22	1	3	0	
SubmitTurnControl		78%		29%	11	14	3	34	0	2	0	
PostSpectatorCheckTurn		45%		0%	2	4	11	21	1	3	0	
GetStopWatching		43%		n/a	1	3	8	17	1	3	0	
CheckTurnControl		24%		0%	4	5	10	14	1	2	0	
GetHomeRoute		94%		83%	6	21	6	94	0	3	0	
PostSignInRoute		82%		40%	4	8	4	31	0	3	0	
JsonUtils		0%		n/a	5	5	5	5	5	5	1	
ValidateMoveControl		98%		87%	2	10	2	65	0	2	0	
PostReplayBackwardRoute		97%		75%	2	7	1	34	0	3	0	
PostReplayForwardRoute		97%		75%	2	7	1	34	0	3	0	
ResignGameControl		96%		62%	3	6	1	17	0	2	0	
WebServer		100%		n/a	0	3	0	31	0	3	0	
GetReplayGameRoute		100%		66%	2	6	0	44	0	3	0	
GetSpecatorGame		100%		n/a	0	3	0	29	0	3	0	
GetSignInRoute		100%		n/a	0	3	0	12	0	3	0	
Total	438 of 3,020	85%	57 of 148	61%	54	128	94	636	11	54	1	

Figure 5: UI Code Coverage



WebCheckers Application Tier Test Coverage

WebCheckers Application Tier Test Coverage

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.webcheckers.util		94%		76%	228	636	68	1,229	8	135	1	19
com.webcheckers.ui		85%	240	61%	54	128	94	636	11	54	1	19
com.webcheckers		0%		0%	8	8	37	37	6	6	1	1
Total	1,015 of 10,812	90%	301 of 1,154	73%	290	772	199	1,902	25	195	3	39

Application Tier

Figure 6: Overall Code Coverage