ContagionCheckers Design Documentation

VibeCheckers

Modification Log

Date	Version	Description	Author(s)
4/5/2020	1.1	Initial: Copy from Markdown	**Joshua Yoder**
4/6/2020	1.2	Updated with Sprint 2 Info	Joshua Yoder
4/6/2020	1.3	Updated State Diagram	Jonathan Baxley
4/15/2020	1.4	Updated with Sprint 3 info	Jonathan Baxley
4/21/2020	1.5	Updated with Sprint 4 info	Joshua Yoder

Team Information

Team Name: VibeCheckers

Team Members: **Jonathan Baxley**
Dhaval Shrishrimal
Joshua Yoder
Joe Netti
Andre Decosta

Executive Summary:

This project is WebCheckers, it will provide an online tool for people to play checkers with their friends, provide statistics, play against AI, and spectate games. The game is currently functional in accordance with our Minimum Viable Product but we plan on adding enhancements. A user can sign-in, enter a unique username and password, choose a game, play a game of checkers according to American rules, and end the game. We put our own twist on web checkers with our visual redesign enhancement. Our checkers is now based visually off of the game plague inc. We took the current situation everyone is dealing with and tried to have fun with it in how we changed the visuals for the game.

Purpose:

The goal of this project is to learn more about java spark web framework. The user group is anyone who wants to play checkers online. Our goal for users is that they have a positive experience playing WebCheckers online either against a real person or our AI.

Glossary and Acronyms:

Term	Definition
UI	User Interface
HTML	HyperText Markup Language
MVP	Minimum Viable Product
CSS	Cascading Style Sheets
SVG	Scalable Vector Graphics
AI	Artificial Intelligence

Requirements:

This section describes the features of the application.

- *Sign-in*: Feature allows users to sign-in and create accounts from with to play checkers on. Also allows users to sign-out.
- *Game Creation*: Allows signed-in users to create a game with other signed-in users. Doesn't allow users to play with players in other games.
- *Play Checkers*: Checkers needs to be playable according to American Checkers Rules. This includes a couple epics including move making, resignation, and win conditions.
- *Visual Enhancement: The Webcheckers app must visually match our contagion theme.*
- *Spectator*: Users must be able to spectate other users' games.

Definition of MVP:

The minimum viable product is a website that has a functioning sign-in page, a home page that allows the user to pick who to play against, a working checkers game that properly follows American Checkers rules, allows players to quit, has a proper win condition. In addition to this minimum viable product, it must also have enhancements of spectating games and visual enhancements.

MVP Features:

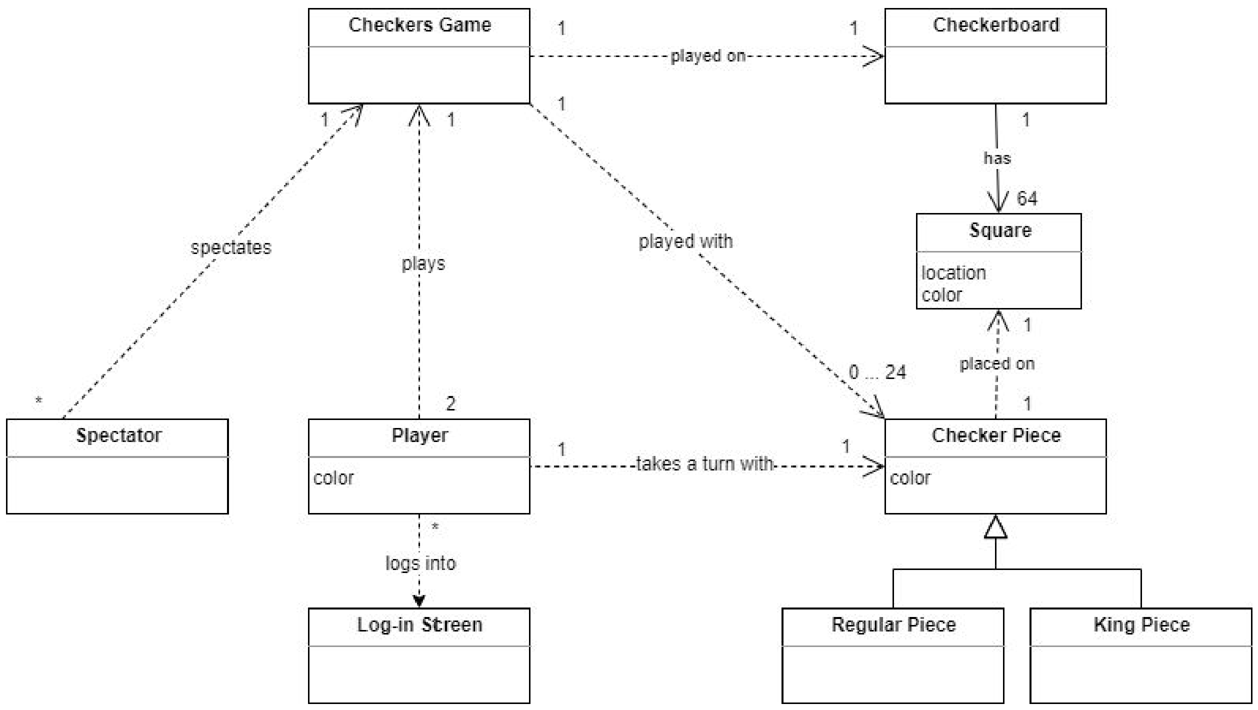
- **Sign-In**
- **Game Creation**
- **Game Resignation**
- **Move Making**
- **King Crowning**
- **King Movement**
- **Win Condition**

Enhancements:

- **Visual/Theme Enhancements**
- **Spectate game**

Application Domain:

Overview of Major Domain Areas



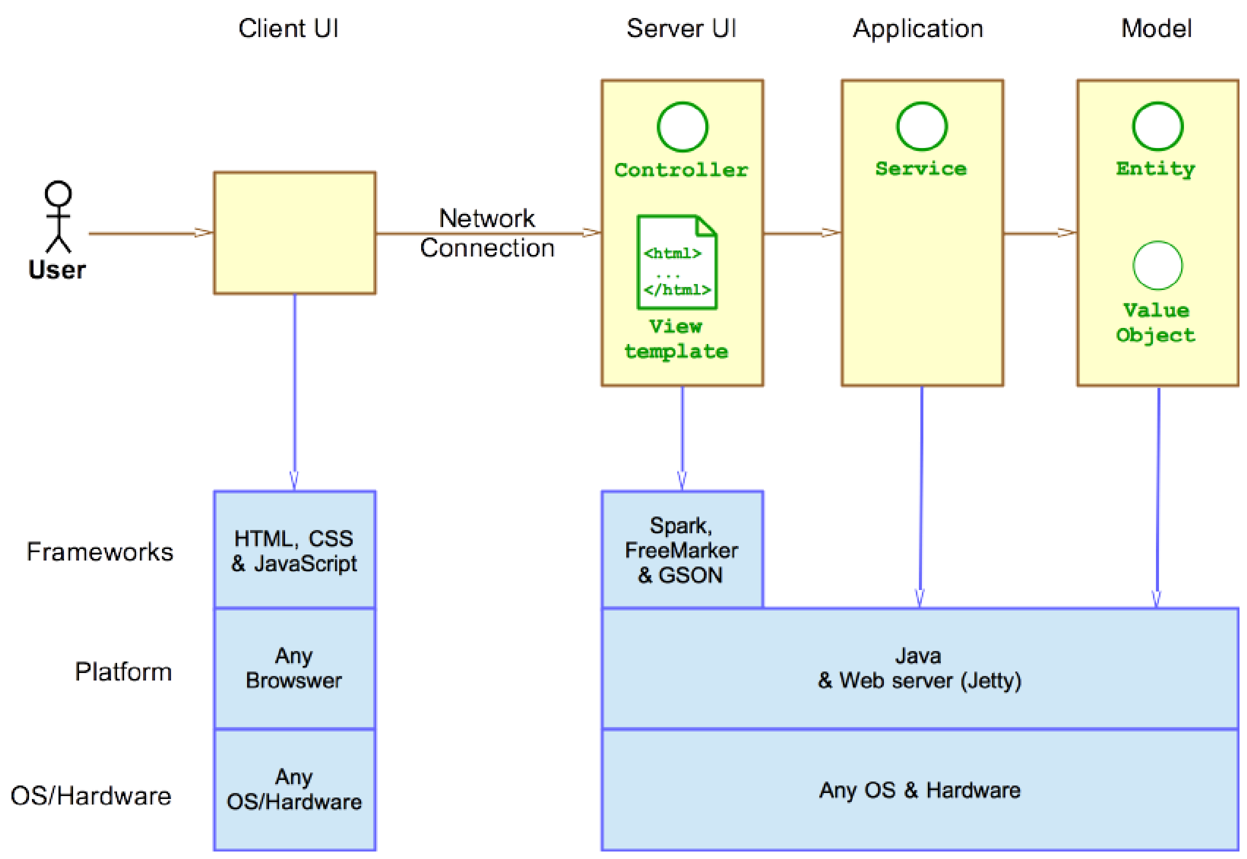
- A *checker piece* can either be a regular piece or a king piece, and has a color attribute. A single checker piece lies on a *square*.
- A *player* can either be a human or an AI player, and has a color attribute. Human players can log into the application via the *Log-In Screen*. A single player takes a turn with a single checker piece. Two players play a single checkers game.
- Many *spectators* can spectate a given checkers game.
- A *checkers game* is played on a single *checkerboard*, with 0 to 24 *checker pieces*.
- A *checkerboard* has 64 *squares*.
- A *square* has location and color attributes.

Application Architecture:

This section describes the application architecture.

Summary:

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



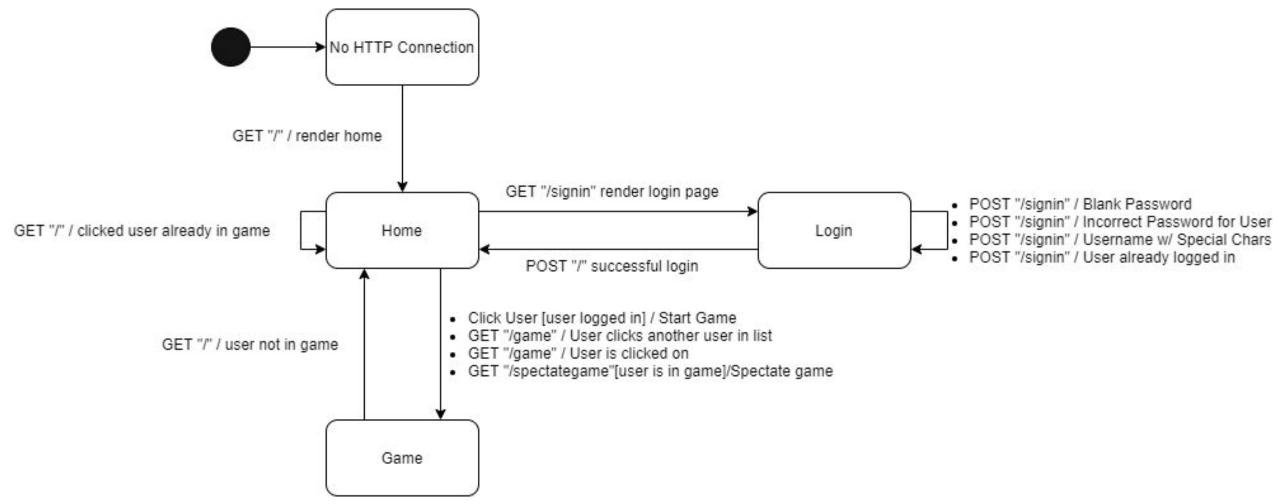
As a web application, the user interacts with the system using a browser. The client-side of the UI is composed of HTML pages with some minimal CSS for styling the page. There is also some JavaScript that has been provided to the team by the architect.

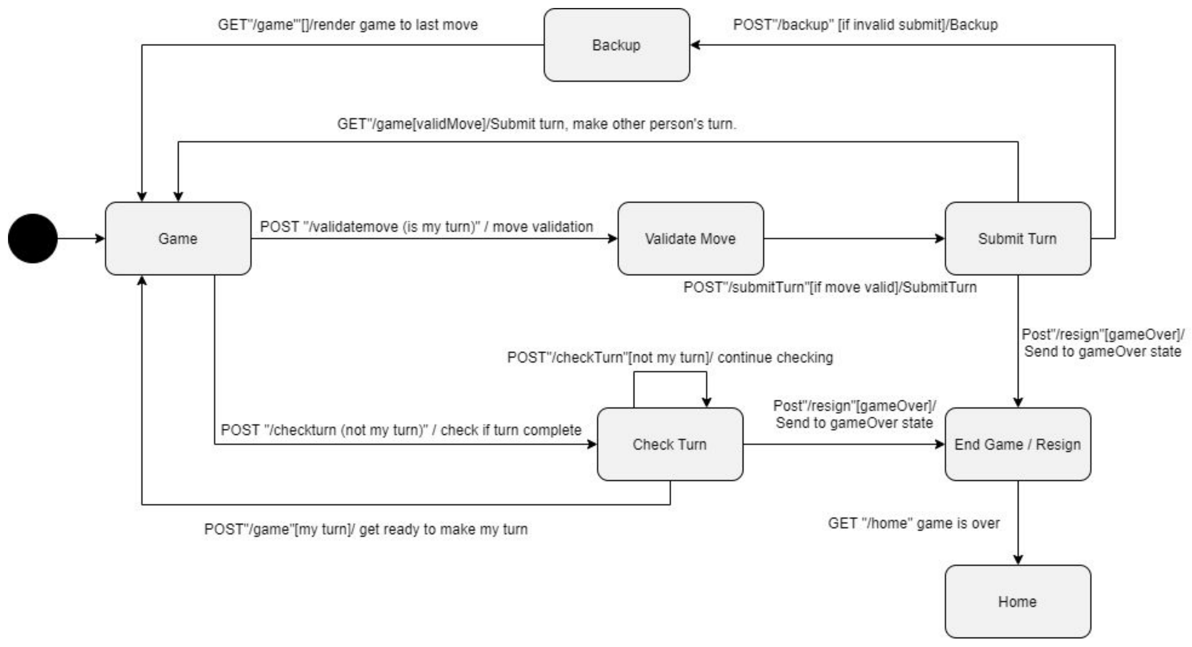
The server-side tiers include the UI Tier that is composed of UI Controllers and Views. Controllers are built using the Spark framework and View are built using the FreeMarker framework. The Application and Model tiers are built using plain-old Java objects (POJOs).

Details of the components within these tiers are supplied below.

Overview of User Interfaces:

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


No HTTP Connection
GET "/" / render home
Home
GET "/signin" render login page
Login
POST "/" successful login
POST "/signin" / Blank Password
POST "/signin" / Incorrect Password for User
POST "/signin" / Username w/ Special Chars
POST "/signin" / User already logged in
GET "/" / clicked user already in game
GET "/" / user not in game
Click User [user logged in] / Start Game
GET "/game" / User clicks another user in list
GET "/game" / User is clicked on
GET "/spectategame"[user is in game]/Spectate game
Game


GET"/game""[]/render game to last move
Backup
POST"/backup" [if invalid submit]/Backup
GET"/game[validMove]/Submit turn, make other person's turn.
Game
POST "/validatemove (is my turn)" / move validation
Validate Move
Submit Turn
POST"/submitTurn"[if move valid]/SubmitTurn
Post"/resign"[gameOver]/ Send to gameOver state
POST"/checkTurn"[not my turn]/ continue checking
POST "/checkturn (not my turn)" / check if turn complete
Check Turn
Post"/resign"[gameOver]/ Send to gameOver state
End Game / Resign
POST"/game"[my turn]/ get ready to make my turn
GET "/home" game is over
Home

The UI tier is responsible for maintaining the communication between the users and the Application. The WebServer class is the center of the UI tier, which receives all the GET and POST requests. When the User first enters the website, a GET / request is received, and the home page is rendered by the template engine. The User has the option to sign-in on the navigation-bar.

If the user clicks the sign-in button, a GET /signin is received by the WebServer and the sign-in page is loaded. If the User does not have an account, he can enter a unique username and password to create an account or sign-in with an existing account by entering the correct username and password. While creating a new account there are a few restrictions on the validity of the username and password.

- Username should at least contain one alpha-numeric character and cannot contain any special characters except for spaces.
- Passwords cannot be an empty string.

After the user enters the username and password a POST / request is sent and the Application verifies the username and password. If the username or password is incorrect the user is displayed with an appropriate error message and the user has the option to enter the username and password again. If the username and password is correct the home page is loaded and an attribute is stored in the session with the ID Player.

The home page displays the username and a sign-out button on the navigation- bar. Also, the names of all the available players are displayed. Each name is a button, which when clicked sends a POST /game to the WebServer which then loads up the game page for the user. When the home page of the player clicked by the user refreshes, he is redirected to the game page and a GET /game is received by the WebServer. If the Player clicked by the user is already in a game, the user is displayed an error notifying the user that the selected Player is already in a game. If the user clicks on the sign-out button a POST /signout request is sent to the WebServer which then removes the session attribute ID PLAYER and the user is redirected to the home page and GET / is received by the WebSever.

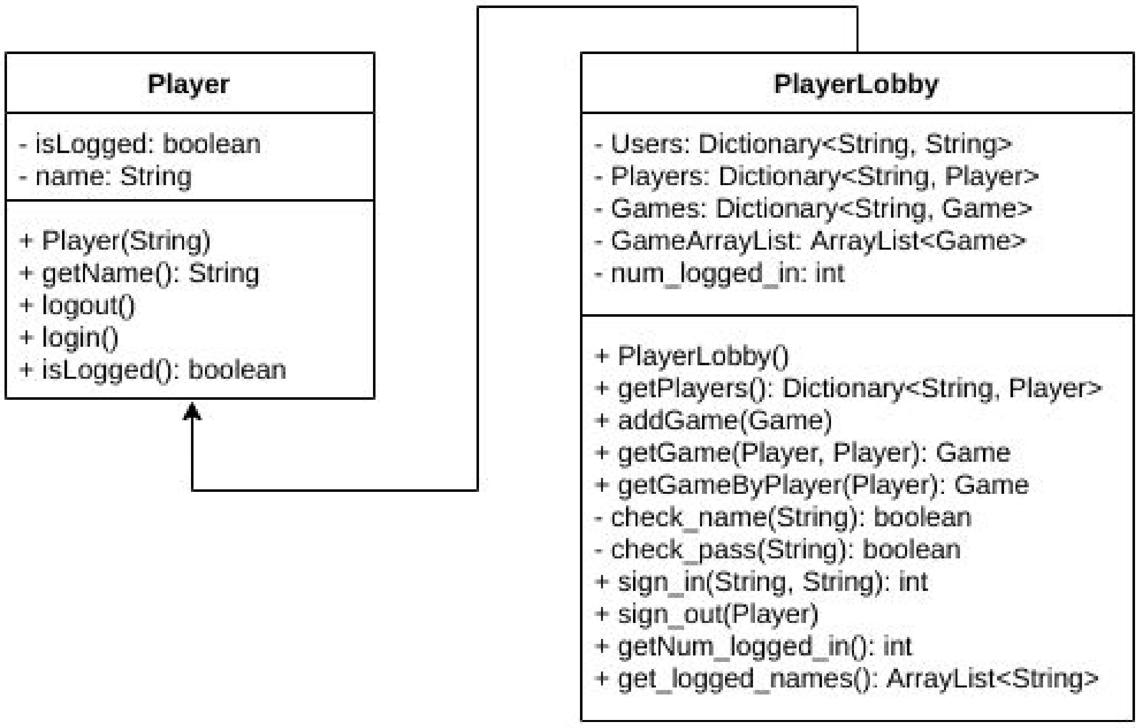
The User starting the game is the red player and goes first. The Game Page has a message box notifying that either it is your turn and the instructions to move a piece OR its your opponent's turn. When it's your turn you have the option to back-up one move, play a valid move, resign, or submit your turn, else you only have the option to resign.

When the game is running there is a Post/ValidateMove to check if a player has used a valid move. Post/SubmitMove will then submit the move if it meets all requirements and end the current user's turn. Post/CheckTurn is running for the player whose turn it’s not. It will continuously check if it's their turn or if the game is over. Once it is their turn it will change the state to it being their turn. If the game ends it will send them to the game over state. The move rules follow the American Checker Rules and will force the user to follow those guidelines.

Overview of Application Interface:

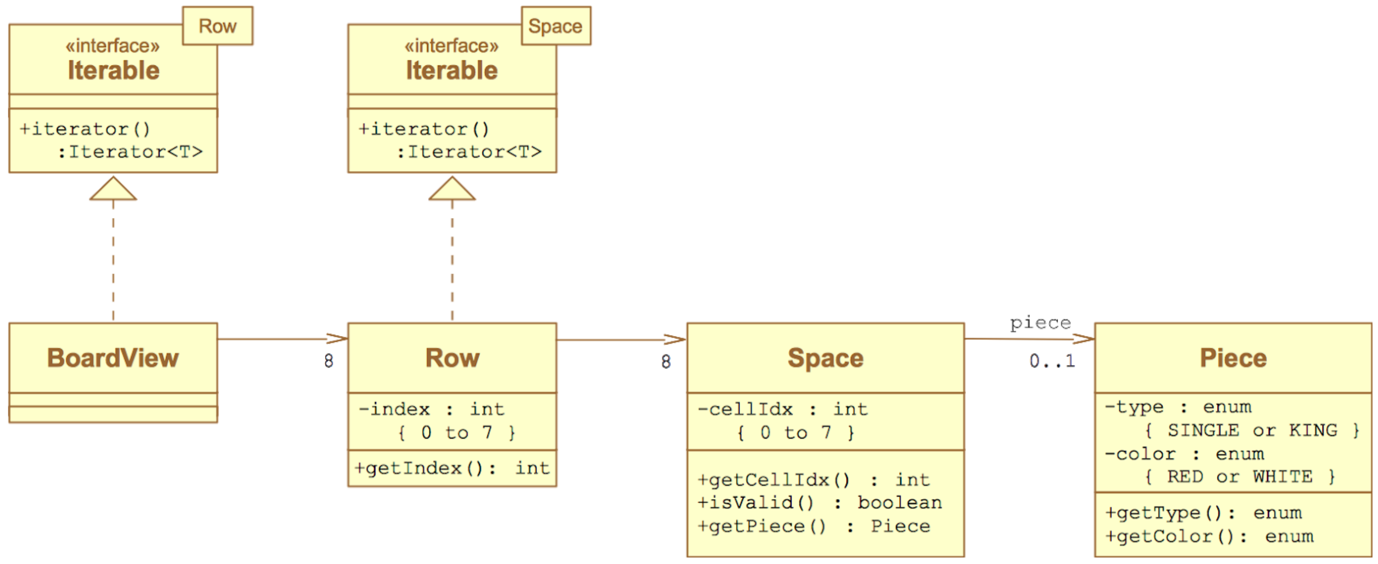
All Players that have an account are stored in the PlayerLobby. All games of players in the lobby are stored in the PlayerLobby, but in future releases a GameCenter will be taken over this functionality.

The Player and PlayerLobby classes has the following structure:


Player
- isLogged: boolean
- name: String
+ Player(String)
+ getName(): String
+ logout()
+ login()
+ isLogged(): boolean
PlayerLobby
- Users: Dictionary<String, String>
- Players: Dictionary<String, Player>
- Games: Dictionary<String, Game>
- GameArrayList: ArrayList<Game>
- num_logged_in: int
+ PlayerLobby()
+ getPlayers(): Dictionary<String, Player>
+ addGame(Game)
+ getGame(Player, Player): Game
+ getGameByPlayer(Player): Game
- check_name(String): boolean
- check_pass(String): boolean
+ sign_in(String, String): int
+ sign_out(Player)
+ getNum_logged_in(): int
+ get_logged_names(): ArrayList<String>

The PlayerLobby class sign’s players in and out, checks usernames and passwords of players, and stores games of players.

Overview of Model Interface:



The above UML contains classes that are in the model. The classes in model specify the structure of the components of a checkers game–from pieces, to board, to game. Specifically: each game has a board, each board has 8 rows of spaces, each space may have a piece, and pieces have a type and color. This very *clean* design provided by the beloved product owner has minimal coupling and high cohesion. Board “talks” to Row, Row “talks” to Space, Space “talks” to Piece. We added a game class to store the board of a game and players of the game. Game is the high-level class that is called to make moves in a game. Space also has a position represented by an int row and an int column. Game has a subclass called turn that keeps track of the turn as well.

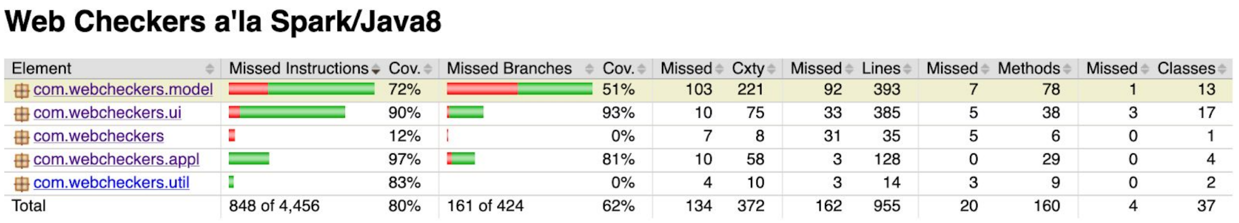
Testing:

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

Acceptance Testing:

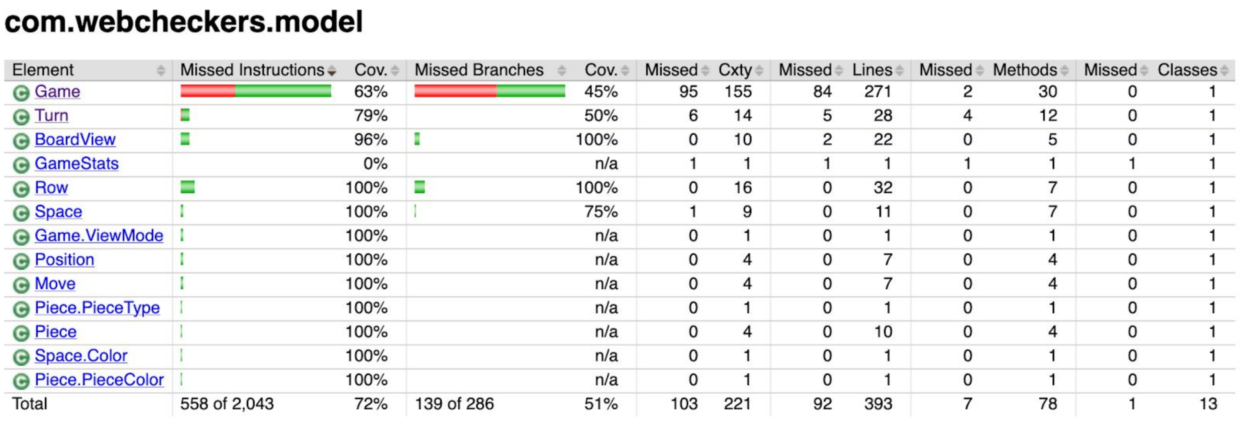
- So far all sprint 0, sprint 1, and sprint 2 user stories have passed all acceptance criteria.
- So far 0 user stories have failed any acceptance criteria.
- The remainder of user stories for sprint 3 have not yet had any testing.

Unit Testing and Code Coverage:



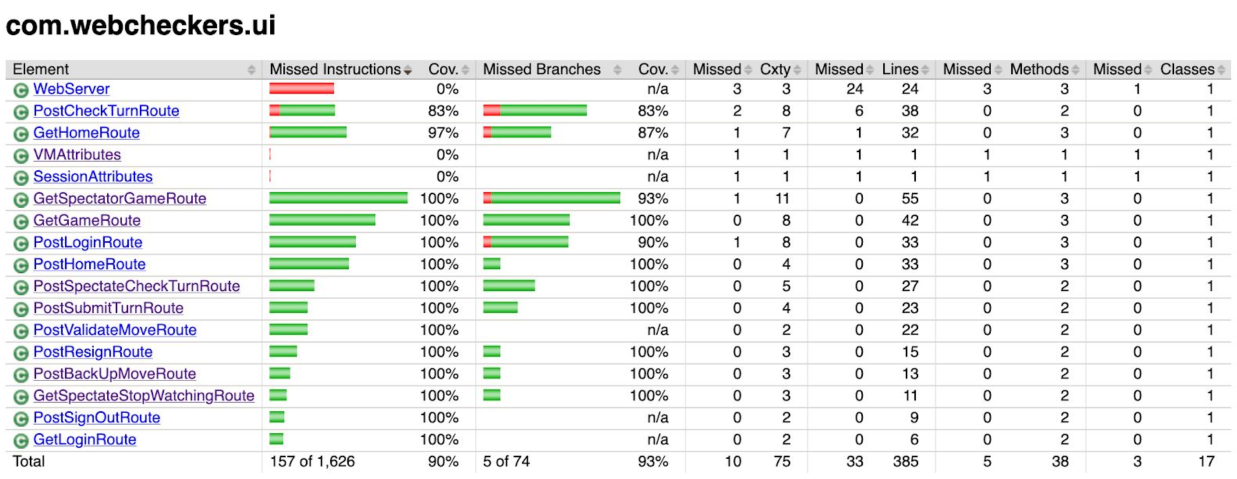
Web Checkers a'la Spark/Java8

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.webcheckers.model		72%		51%	103	221	92	393	7	78	1	13
com.webcheckers.ui		90%		93%	10	75	33	385	5	38	3	17
com.webcheckers		12%		0%	7	8	31	35	5	6	0	1
com.webcheckers.appl		97%		81%	10	58	3	128	0	29	0	4
com.webcheckers.util		83%		0%	4	10	3	14	3	9	0	2
Total	848 of 4,456	80%	161 of 424	62%	134	372	162	955	20	160	4	37



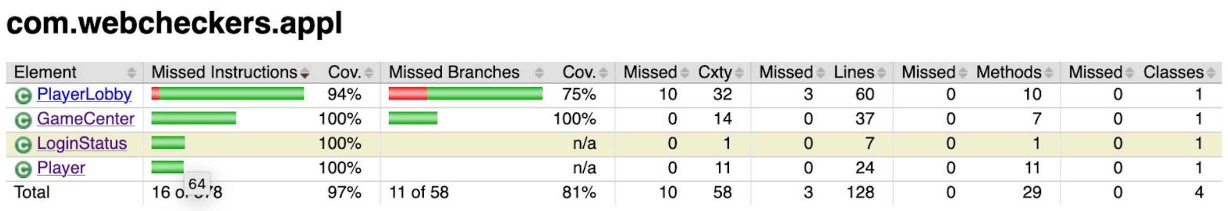
com.webcheckers.model

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Game		63%		45%	95	155	84	271	2	30	0	1
Turn		79%		50%	6	14	5	28	4	12	0	1
BoardView		96%		100%	0	10	2	22	0	5	0	1
GameStats		0%		n/a	1	1	1	1	1	1	1	1
Row		100%		100%	0	16	0	32	0	7	0	1
Space		100%		75%	1	9	0	11	0	7	0	1
Game.ViewMode		100%		n/a	0	1	0	1	0	1	0	1
Position		100%		n/a	0	4	0	7	0	4	0	1
Move		100%		n/a	0	4	0	7	0	4	0	1
Piece.PieceType		100%		n/a	0	1	0	1	0	1	0	1
Piece		100%		n/a	0	4	0	10	0	4	0	1
Space.Color		100%		n/a	0	1	0	1	0	1	0	1
Piece.PieceColor		100%		n/a	0	1	0	1	0	1	0	1
Total	558 of 2,043	72%	139 of 286	51%	103	221	92	393	7	78	1	13



com.webcheckers.ui

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
WebServer		0%		n/a	3	3	24	24	3	3	1	1
PostCheckTurnRoute		83%		83%	2	8	6	38	0	2	0	1
GetHomeRoute		97%		87%	1	7	1	32	0	3	0	1
VMAttributes		0%		n/a	1	1	1	1	1	1	1	1
SessionAttributes		0%		n/a	1	1	1	1	1	1	1	1
GetSpectatorGameRoute		100%		93%	1	11	0	55	0	3	0	1
GetGameRoute		100%		100%	0	8	0	42	0	3	0	1
PostLoginRoute		100%		90%	1	8	0	33	0	3	0	1
PostHomeRoute		100%		100%	0	4	0	33	0	3	0	1
PostSpectateCheckTurnRoute		100%		100%	0	5	0	27	0	2	0	1
PostSubmitTurnRoute		100%		100%	0	4	0	23	0	2	0	1
PostValidateMoveRoute		100%		n/a	0	2	0	22	0	2	0	1
PostResignRoute		100%		100%	0	3	0	15	0	2	0	1
PostBackUpMoveRoute		100%		100%	0	3	0	13	0	2	0	1
GetSpectateStopWatchingRoute		100%		100%	0	3	0	11	0	2	0	1
PostSignOutRoute		100%		n/a	0	2	0	9	0	2	0	1
GetLoginRoute		100%		n/a	0	2	0	6	0	2	0	1
Total	157 of 1,626	90%	5 of 74	93%	10	75	33	385	5	38	3	17



com.webcheckers.appl

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
PlayerLobby		94%		75%	10	32	3	60	0	10	0	1
GameCenter		100%		100%	0	14	0	37	0	7	0	1
LoginStatus		100%		n/a	0	1	0	7	0	1	0	1
Player		100%		n/a	0	11	0	24	0	11	0	1
Total	16 o[illegible]78	97%	11 of 58	81%	10	58	3	128	0	29	0	4

- Currently we have overall code coverage of 80%
  - 72% Model Layer
  - 90% UI Layer
  - 97% Application Layer
- All sprint 1, 2, and 3 classes and methods have unit tests written for them.

Code Metrics:

We will not be doing Code Metrics in this sprint.

Design Quality and Possible Revisions:

Currently we are quite happy with how our design for the game turned out. If we find that there is anywhere we can improve we will take a look at it and address on a case by case basis.