PROJECT 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.* hi

Team Information

- Team name: SWEN-261 Team 4
- Team members
  - Ben Gurevich
  - Ryan Wolford
  - Olaf Hichwa
  - Ardit Koti
  - Jonah Rosenberg

Executive Summary

This document helps convey design implementations and states that will be encapsulated in Sprint 2 of the Webcheckers Project.

Purpose

Help streamline the popular board game Checkers so that it can be played with a simple interface with anyone you want to play with. With our spectator mode and replay feature, it allows for the best possible experience when playing checkers.

Glossary and Acronyms

Term	Definition
VO	Value Object

Requirements

Placeholder

Definition of MVP

In order to satisfy the Product Owner, the MVP needs to include these 3 key features: First, every player must be able to sign-in(no invalid names allowed) before starting a game. Once done, they must be able to sign-out of the site. Upon signing in, two players must be able to play a game of checkers with each other, if available for a game. Finally, one of the players, may choose to resign, which will end the game instantly.

MVP Features

In order to satisfy the Product Owner, certain Epics and top-level stories need to be integrated into the project:

Player Sign-in:

The player before finding a game must be able to sign-in with a unique and valid name.

Start a Game:

The player must be able to challenge another player if they are available to play a game.

Game Rules:

The player should be able to make moves that are legal based off the American Rules.

Ending the Game:

At any point in the game, one of the players should be able to resign their game, which will end the game immediately. If none choose to, then the game will end once one of the players lose all of their pieces.

Sign-out

Once the player is done playing WebCheckers, they will be able to sign-out of the game, returning to the original front page.

Roadmap of Enhancements

Two possible future enhancements are as follows:

Spectator mode

While signed-in, players should be able to view on-going game that they are not playing.

Replay Backlog

Every game that is played will be saved to a backlog for later viewing.

Replay Viewing

Once a player is signed-in they will be able to select a game that they played previously to step through.

Once in a replay, the player will be able to step forward, backward, or exit the replay

Application Domain

This section describes the application domain.

We start off with the Player model for the Webcheckers application. The Player has 4 options, regarding what they can do during their usage of the site. The player will be able to interact with the board and that is when they will be able to move their piece (if allowed). If the player is not playing a game, they can either spectate a match that is being played currently, or they can watch a game that has been previously 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.

Overview of User Interface

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

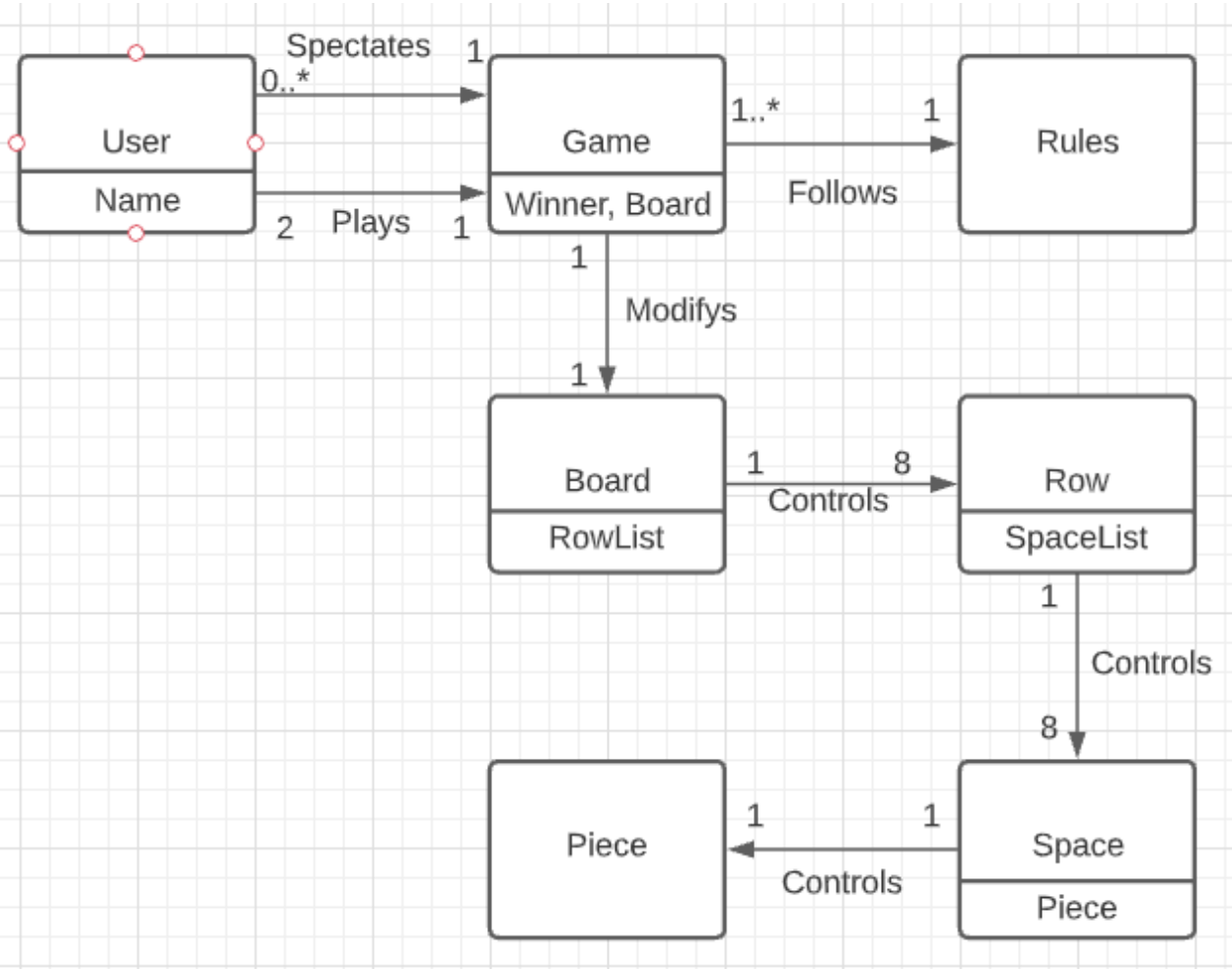


Figure 1: The WebCheckers Domain Model

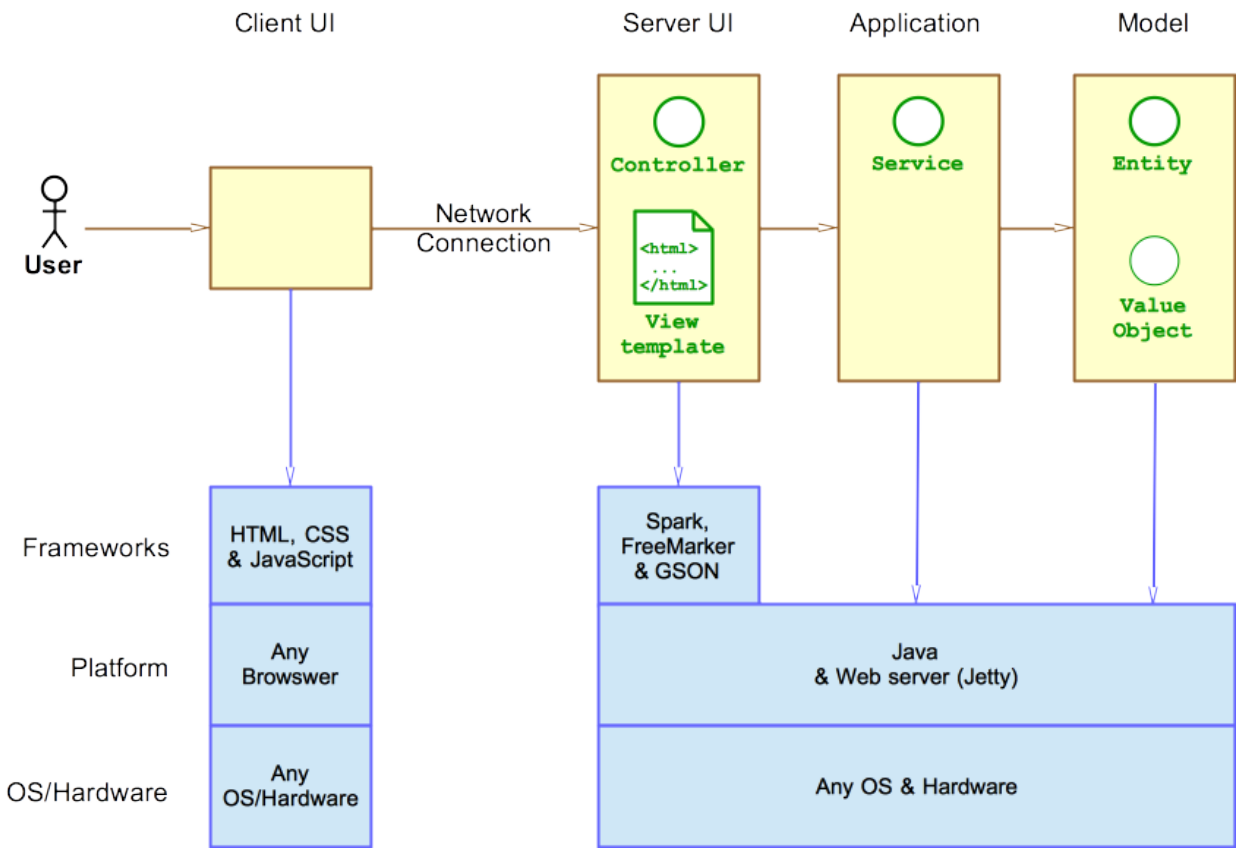


Figure 2: The Tiers & Layers of the Architecture

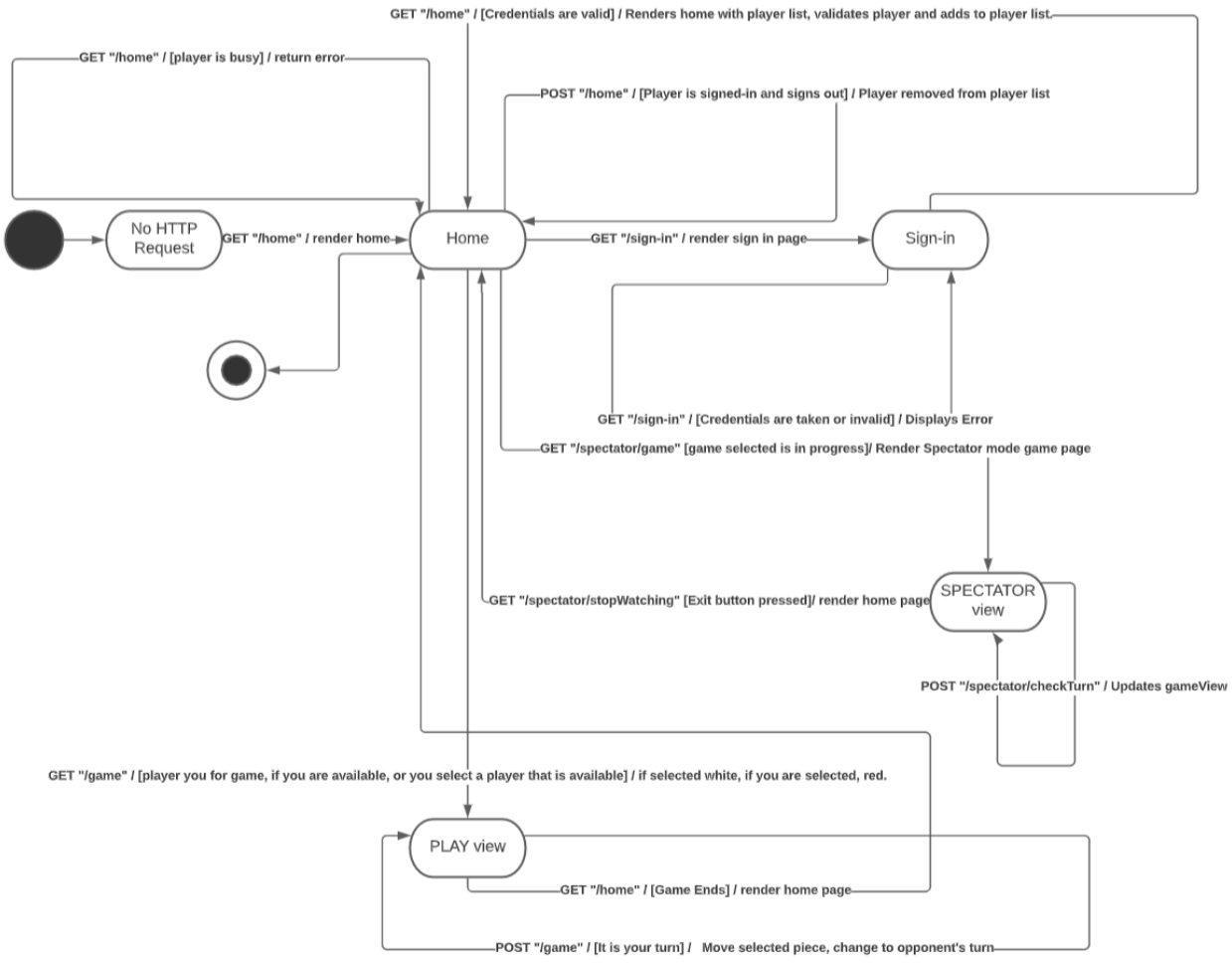
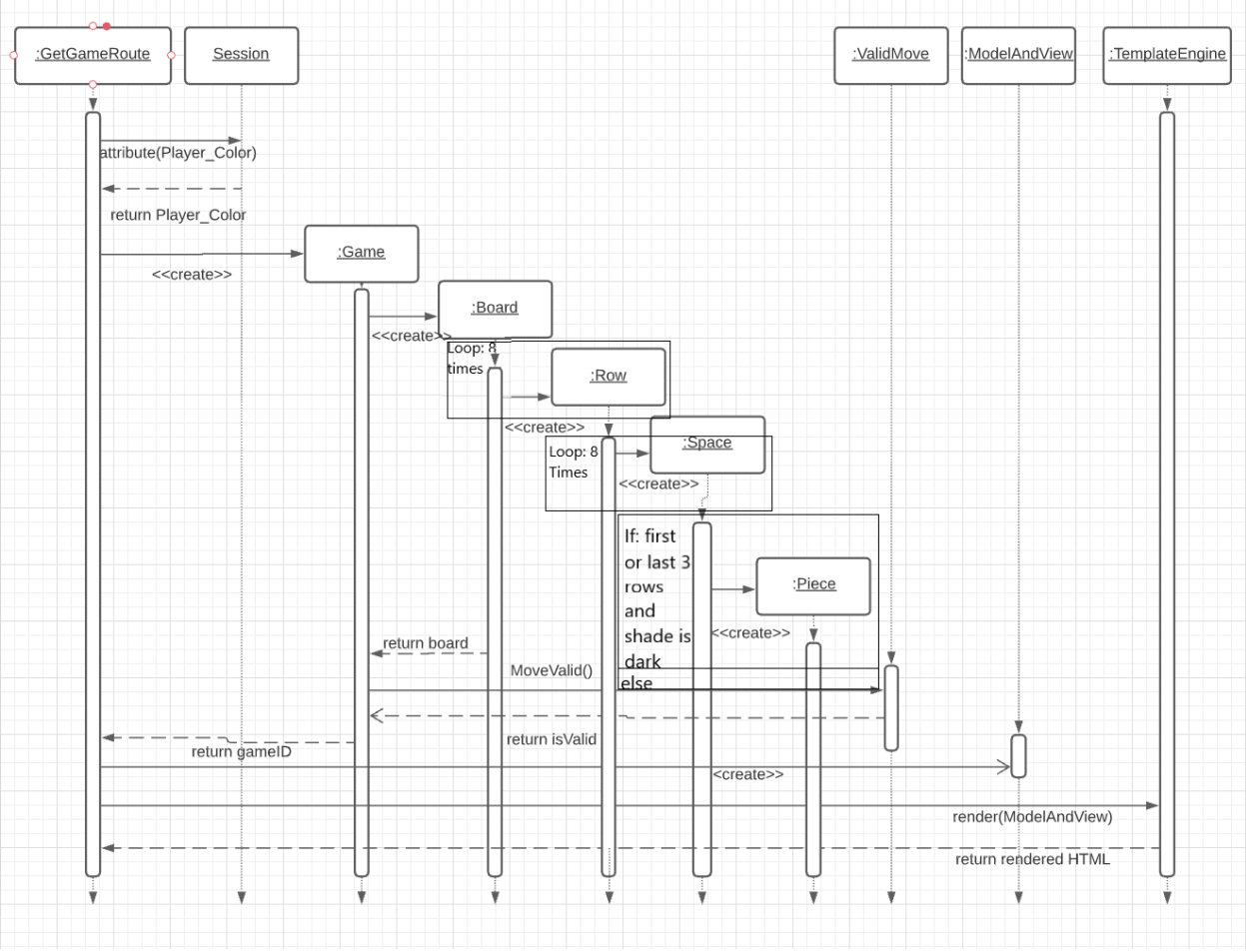


Figure 3: The WebCheckers Web Interface Statechart

This Figure shows the basic state functionality of all stories that will be completed in sprint 2. Important additions from sprint 1 include the player sign-out route from home, the game view post, and the returning to home on game end.

UI Tier

The UI Tier of the application is responsible acting as an interface between the user, and the back end of the application. It takes their interactions with the application and updates their view accordingly. The core of this entire Tier is the Webserver class which is responsible for instantiation of all the routes that the program needs to respond to the user input. GetHomeRoute is used to transfer the view of the user to the Home page of the application. If the user has not signed in at this point than only the amount of players currently signed in is displayed. Otherwise, GetSignIn is used to route the user to the signIn screen and prompt them for their login information. PostSignIn is used to parse the login information entered. If the login is invalid an error message is displayed to the User so that they can enter correct information. Otherwise, the player is routed back toward the home page to engage with other users PostSignOut is called when the user interacts with the sign-out button. This removes them from the active player base and makes them unable to join or watch games. It then directs them to the home page. GetGame is used to start a game when one player challenges another. Both players are moved into the game view and shown the starting position of the board.



GetSpecStop is used to stop the spectator from actually spectating the game, either because they want to watch another game or because the game they are currently watching stopped. GetSpectatorRoute is used specifically so that a player is actually able to watch the game that they want to watch. So, all entities need to be passed through the .ftl pages just like in GetGameRoute. PostBackupRoute is used so that when a player selects a move, they will be able to change their mind and go backwards in a sense. They will be able to change their move to whatever they want, IF there is a move that they can go back to.

PostCheckTurnRoute is used to check whether its the player's turn or not, while checking the conditions that the game is still going on. PostHomeRoute is used so that either when the user wants to leave or when the game is over, they are able to go back to the homepage. PostSpectatorUpdater is used so that as the moves are changing within the game, the spectator is updated on what is happening in current time, and it is constantly checking whether the game is over or not. PostSubmitRoute is used so that when a player has locked in a move, they are able to submit their turn by using the submit button on the game page. Finally, PostValidateMove is used so that player moves are checked and validated to make sure they can be moved, while checking the conditions of the game.

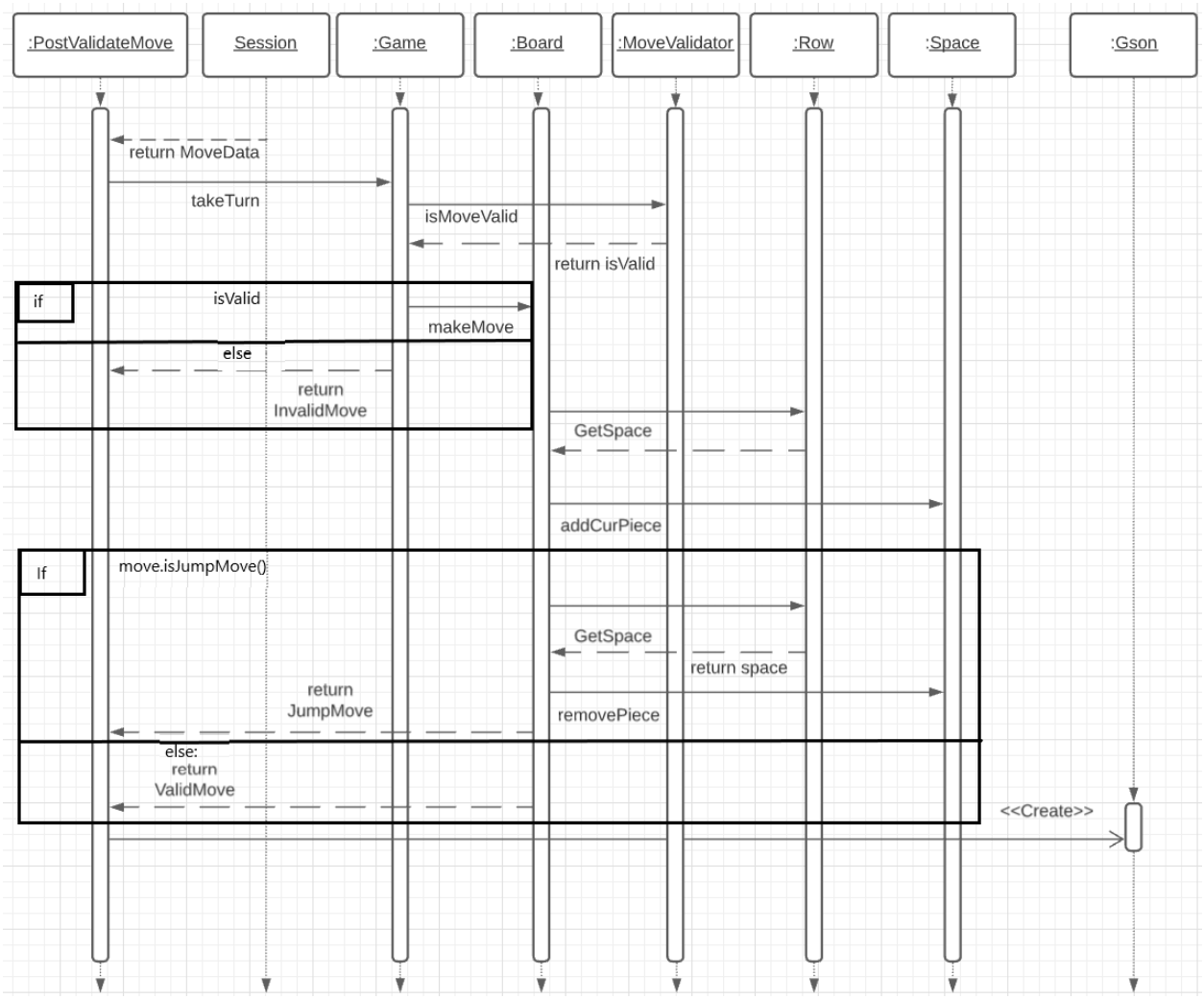


Figure 4: A Sequence Diagram from PostValidateMove

Application Tier

The Application Tier includes the Message, PlayerException, MoveValidator, DevMode, and GameCenter class. The Message class is a useful and concise class that will display either an error type message or a information type message. The PlayerException class hasn't been used yet, but it still has valuable functionality that can come in handy later.

The DevMode class has been extremely useful for us, as it lets us demonstrate different features within the WebCheckers game. Whether we want to show what happens during the endgame of a checkers match or what happens when a multi-jump is detected, we can easily show off these scenarios with the DevMode class.

During Sprint 3, we decided that the best way to check if a move was valid or not, was to implement a MoveValidator class. This allows us to check every condition for a move and to see which moves are possible for the player to make. This is the center of all of our logic regarding move making, and it is has proved to be a great tool.

Along with MoveValidator, we decided as a team that the best possible way to manage games was to create a GameCenter class. The GameCenter class allows us to get available games, create games, and see the player lobby entirely. This change definitely made implementing spectator way easier.

Model Tier

The Model Tier encompasses all the logic within the game based off the American Checker Rules. Within the actual tier, you have the Board and Player classes. The board class creates the actual board using the Row class to make it 2D. The player class is used for setting their name for the game. The Game class creates two Players, red and white. The PlayerLobby class handles the name and password of the Player. Within that class, usernames and passwords are checked to see if they are valid. The Piece class has multiple states, including Color (Red or White), Variety (Regular or King), or State (Alive or Dead). The Space class goes through each spot on the checkerboard and whether the spot is vacant, invalid, or not, will add a piece based on the American Rules.

The Move class within the Model tier takes a starting position and an ending position and makes sure to check conditions on whether there are jumps. The actual Position class takes in a row and a cell and that is how the actual position of the piece is determined.

ViewMode is an enum which has two entities, Spectator and Play. Essentially determines whether the current user is a player or is just someone who is watching the game.

Design Improvements

Regarding our project after Sprint 3, there are certainly designs that can be changed and that can be further improved. First off, like we mentioned back in Sprint 2, we didn't seem to follow the vision document as closely as we should have. So, repetitive code can be seen sometimes within our project along with some unnecessary design choices. Yet, after our mistakes made in the previous sprints, we think that our design choices have made things easier for us as a team. For example, sometime during Sprint 3, we decided as a team that a GameCenter class would help organize games and help us manage who watches and plays games. This change actually helped us get our spectator mode working functionally. ## Testing

Acceptance Testing

Three user stories fully passed all acceptance criteria tests. One story currently has failed half its tests and five stories did not reach the stage of testing.

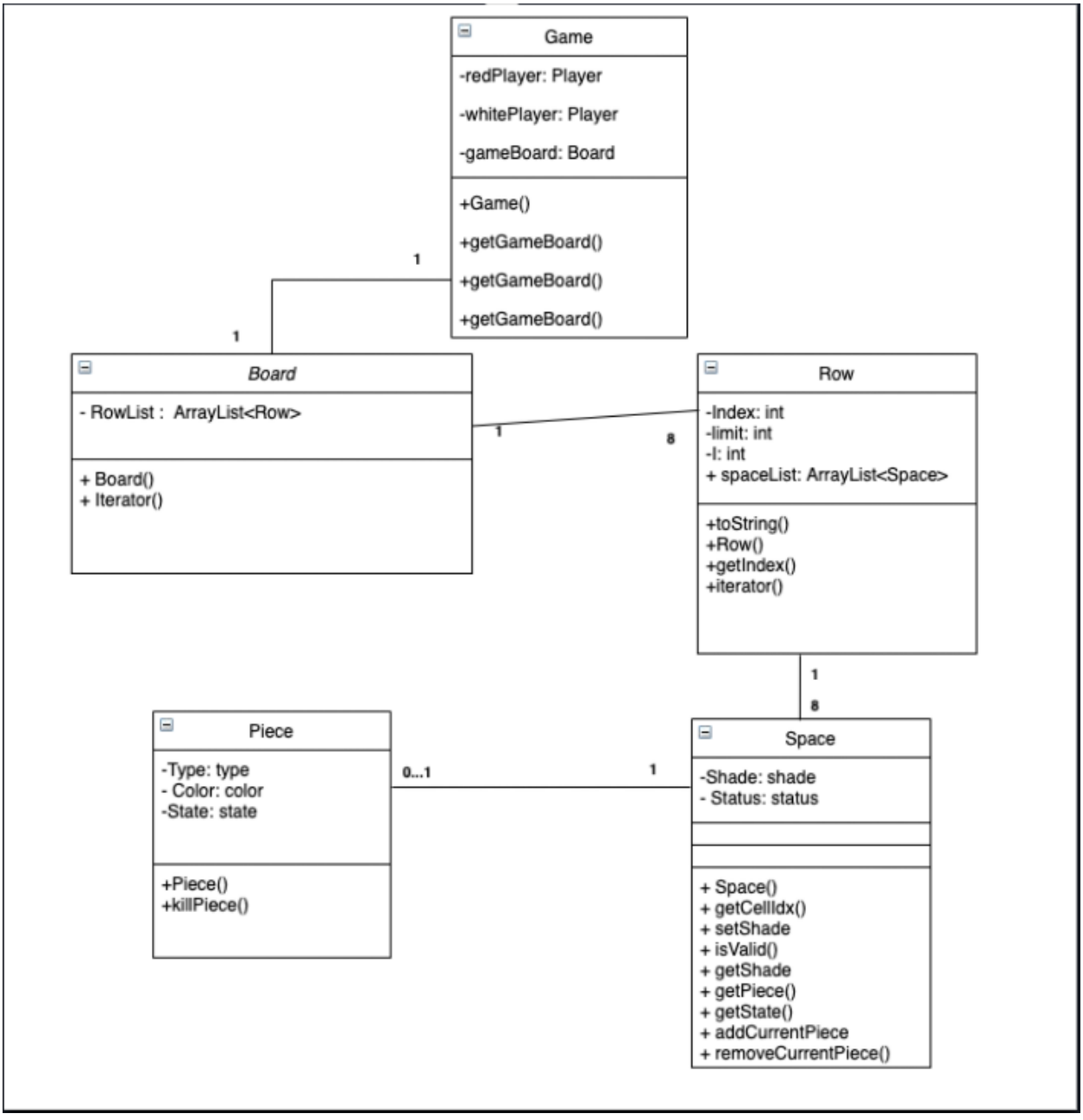


Figure 5: Model class diagram

Unit Testing and Code Coverage

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Space		0%		0%	35	35	50	50	14	14	1	1
Row		0%		0%	5	5	11	11	4	4	1	1
Board		0%		0%	4	4	9	9	3	3	1	1
PlayerLobby		79%		66%	11	25	7	36	3	10	0	1
Row.new Iterator(){...}		0%		0%	4	4	4	4	3	3	1	1
Board.new Iterator(){...}		0%		0%	4	4	4	4	3	3	1	1
Game		0%		n/a	4	4	9	9	4	4	1	1
ViewMode		0%		n/a	1	1	4	4	1	1	1	1
Space.Status		0%		n/a	1	1	1	1	1	1	1	1
ActiveColor		0%		n/a	1	1	3	3	1	1	1	1
Space.Shade		0%		n/a	1	1	1	1	1	1	1	1
Player		100%		n/a	0	6	0	10	0	6	0	1
Piece		100%		n/a	0	5	0	9	0	5	0	1
Piece.State		100%		n/a	0	1	0	2	0	1	0	1
Player.Color		100%		n/a	0	1	0	1	0	1	0	1
Piece.type		100%		n/a	0	1	0	2	0	1	0	1
Piece.color		100%		n/a	0	1	0	2	0	1	0	1
Total	504 of 761	33%	60 of 80	25%	71	100	101	156	38	60	10	17

com.webcheckers.ui

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
GetGameRoute		0%		n/a	3	3	33	33	3	3	1	1
WebServer		0%		n/a	3	3	17	17	3	3	1	1
PostSignOutRoute		0%		n/a	3	3	23	23	3	3	1	1
GetHomeRoute		60%		50%	1	4	9	29	0	3	0	1
GetSignInRoute		55%		50%	1	4	7	17	0	3	0	1
PostSignInRoute		93%		75%	3	8	3	35	0	2	0	1
PostHomeRoute		100%		n/a	0	3	0	13	0	3	0	1
Total	388 of 694	44%	5 of 16	68%	14	28	92	167	9	20	3	7

com.webcheckers.util

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Message		83%		50%	7	16	2	17	2	10	0	1
PlayerException		0%		n/a	1	1	2	2	1	1	1	1
Message.Type		100%		n/a	0	1	0	2	0	1	0	1
Total	18 of 104	82%	6 of 12	50%	8	18	4	21	3	12	1	3

Our overall coverage for testing was fairly low due to a last minute rush of development with which testing was not able to keep pace with. This sort of rapid development can create issues later down the line when untested methods are relied upon in further development. In the next sprint of development the team will address these issues to give us a solid base for final development.

#Metrics ###Chidamber-Kemerer Metrics These Metrics showed no outliers in the analysis. All values were in the target ranges. ###Complexity Metrics The `MoveValidator`, `PostSignInRoute`, and `Space` classes had average operational Complexity values higher than the target range. In addition, the `Space`, `MoveValidator`, and `Game` classes had weighted method complexity values higher than the target range. The cognitive complexity values for the Space.Space(), MoveValidator.simpleJumpMoves(), Game.takeTurn(), Game.isOver(), and MoveValidator.kingJumpMoves() methods were higher than the target range. The Game.takeTurn(), PlayerLobby.checkAndAddName(), MoveValidator.isOneDiagonal(), Game.isOver(), PostSpectatorUpdator.handle(), Game.submitMove(), Space.moveTo(), GetGameRoute.handle(), and GetSpectatorRoute.handle() methods have essential cyclomatic complexity values higher than the target range. The design complexity values for the Space.Space(), MoveValidator.simpleJumpMoves(), MoveValidator.kingJumpMoves(), MoveValidator.kingDiagonalMoves(), and Game.isOver() methods are higher than the target range. The cyclomatic complexity values for the Space.Space(), MoveValidator.simpleJumpMoves(), MoveValidator.kingJumpMoves(), Game.isOver(), and other Space.Space() methods are higher than the target range. ###Javadoc Coverage Metrics No values in this analysis were outside of their target ranges. ###Lines of Code Metrics No values were outside a target range. The `MoveValidator` class had the most comment lines of code, and the most total lines of code. The test classes had the least comment lines of code. ###Martin Package metrics No values were outside a target range. Every package had 0 abstractness. The model package had the highest

value for afferent couplings. The application package had the highest value for efferent couplings.

##Recommendations For the SignInRoute, no changes should be made as the route is unstable by default and does its main purposes properly. The Space class definitely needs design changes. It should be split into other classes which have higher cohesion, such as a single board populator, rather than having board population be done in Space. The MoveValidator and Game classes could also be split into other classes with more focused purposes.

##Future Enhancements Firstly, since the implementation process was rushed, an enhancement that was never reached, was the Replay mode for WebCheckers. The Replay Mode would have allowed players to watch previously played games on the website using a unique game-id.

Optional enhancements that can be added in the future can be AI (with varying difficulties), leaderboards, tournaments, and access to multiple games being played by a player. These additions are minor, yet provide so much more to the experience with WebCheckers and can definitely be looked at in the future.