**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 |
|  |  |  |  |

**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 be visually match our contagion theme.***
* ***Spectator*: User 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 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:**

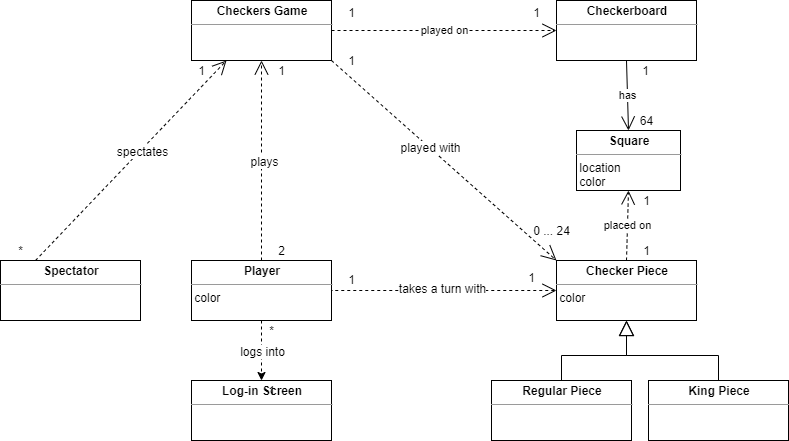
* **• Sign-In  
  • Game Creation  
  • Game Resignation**
* **• Move Making  
  • King Crowning**
* **• King Movement**
* **• Win Condition**

**Enhancements:**

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

**Application Domain:**

**Overview of Major Domain Areas**

****

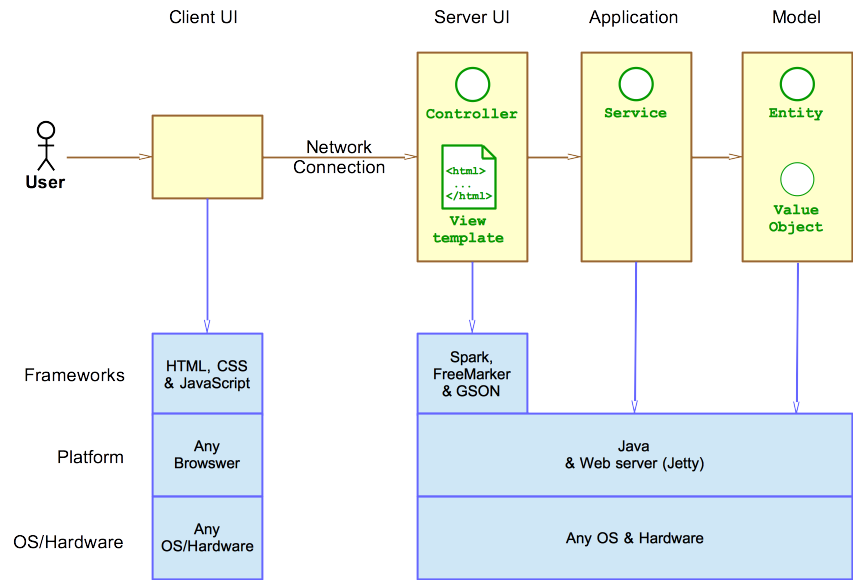
* **A *checker piece* can either be a regular piece or a king piece, and has a color attribute. A single checker piece lies on a *square*.**
* **A *player* can either be a human or an AI player, and has a color attribute. Human players can log into the application via the *Log-In Screen*. A single player 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.

****

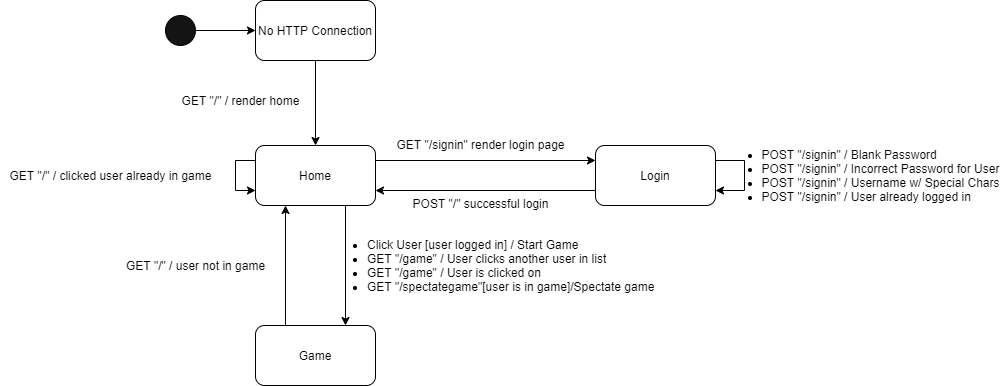
***As a web application, the user interacts with the system using a browser. The client-side of the UI is composed of HTML pages with some minimal CSS for styling the page. There is also some JavaScript that has been provided to the team by the architect.***

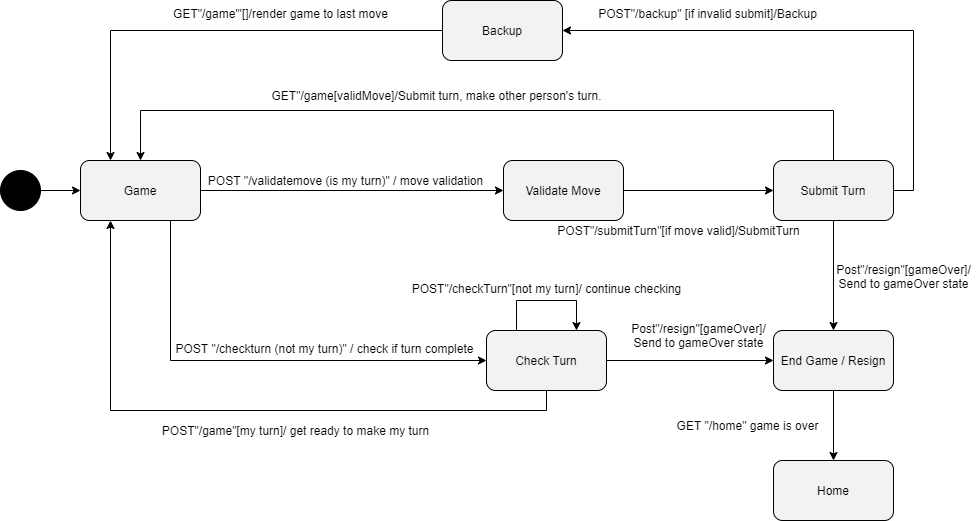
***The server-side tiers include the UI Tier that is composed of UI Controllers and Views. Controllers are built using the Spark framework and View are built using the FreeMarker framework. The Application and Model tiers are built using plain-old Java objects (POJOs).***

**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.

****

****

**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.**

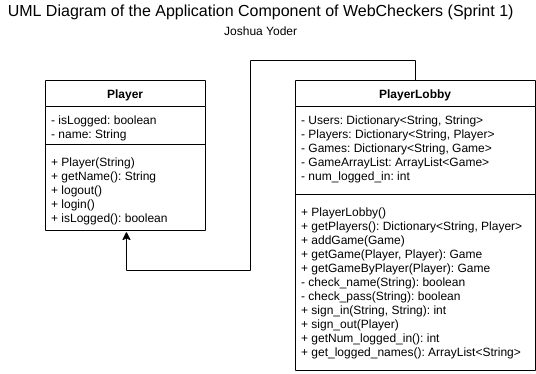
**The User starting the game is the red player and goes first. The Game Page has a message box notifying that either it is your turn and the instructions to move a piece OR its your opponent's turn. When it's your turn you have the option to back-up one move, play a valid move, resign, or submit your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

**Overview of Application Interface:**

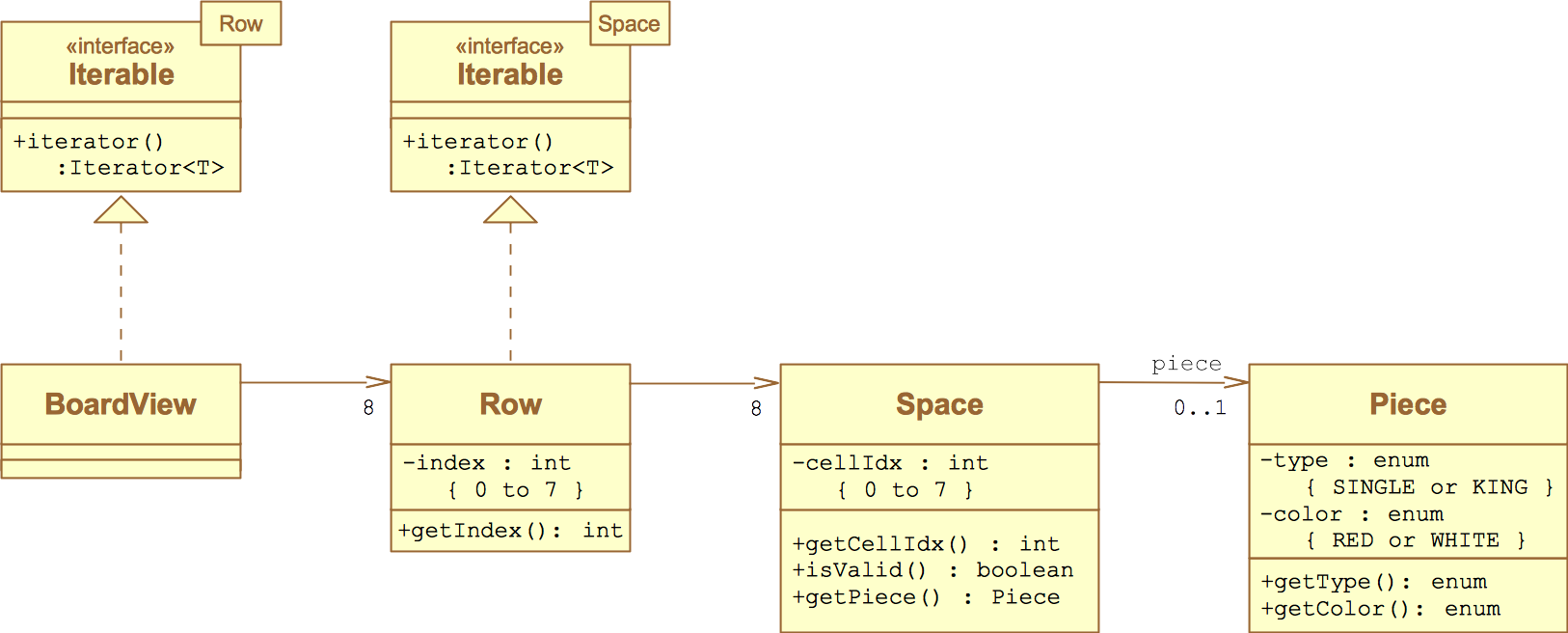
All Players that have an account are stored in the PlayerLobby. All games of players in the lobby are stored in the PlayerLobby, but in future releases a GameCenter will be taken over this functionality.

The Player and PlayerLobby classes has the following structure:

****

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.

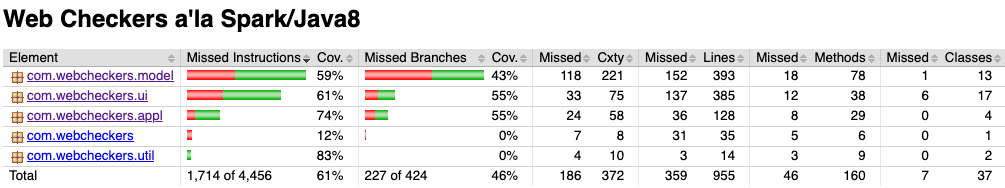
**Testing:**

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:**



* Currently we have overall code coverage of 61%
  + 59% Model Layer
  + 61% UI Layer
  + 74% Application Layer
* All sprint 1 classes and methods have unit tests written for them.
* Remaining tests will be implemented soon. 90% code coverage is possible within the next sprint.

**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. Currently the biggest revisions we want to make is in our testing. Next sprint we plan on tackling code coverage to attempt to go from 61% code coverage up to at least 80% but preferably 90% code coverage. If there are any issues we find in testing with our design quality we will be sure to work on it in the next sprint.