Crispy Cantaloupes

A functional Specifications Report

Last Updated: 29th March 29, 2018

Modification log

Version	Date	Description	Author
1.0	March 20, 2018	Initial Release	Karthik and Emily
2.0	April 1, 2018	Sprint 2 revision	Gaurav

Team Information

- **Team Name:**
  Crispy Cantaloupes

- **Team Members:**
  Karthik Iyer (KL) - Lead coder
  Emily Wesson (EW) - Program manager
  Eugene Chang (EC) - Systems coder + Software tester
  Hongda Lin (HL) - Unit tester
  Gaurav Dev Pant (GP) - Software tester + Documentations incharge

Executive Summary

This project is a multi-player, online web application to play the game of checkers with American rules as enforced by the American Checkers Federation. The application's PlayerLobby allows players to log-in and matches them with an opponent if another player selected them. Once in the game, players take turns moving pieces with the goal to capture opponent pieces and upgrade their own pieces. The game ends when a player resigns or takes all their opponent's pieces.

Purpose

This checkers application follows American rules of play and allows the first player to choose their opponent. The most important user group is checkers enthusiasts who want to play a game online without meeting in person with an opponent.

Glossary and Acronyms

Term	Definition
HTML	Hypertext Markup Language
MVP	Minimum Viable Product
HTTP	Hypertext Transfer Protocol
App	Application
Web	World Wide Web

Requirements

Non-Functional Requirements

This project requires several specific technologies to be used in its implementation. Java 8 and Jetty are used for development, allowing access to the project at the localhost:4567 URL when the server is running. Maven is used to handle the project dependencies. Spark, a web micro framework, and the FreeMarker template engine are required to route the HTTP requests and create HTML responses. The project must be extendable to include various enhancements added in to improve the user experience for players. The acceptance criteria for each part must be testable by other developers on the team, and only tested by team members. Data does not have to persist across the web app after shutdown and restart.

Functional Requirements

- A player must be able to sign into the game with a unique username.
- Once signed in, the player must be able to sign out from any page.
- A player must be able to play a game of checkers following American rules of play using drag-and-drop actions to make their moves.
- A player must be able to select their opponent from the home page, making the selecting player RED/first player, and the selected player WHITE/second player.
- A player must be able to resign from a checkers game at any time, ending the game and forfeiting to their opponent.

Definition of MVP

The Minimum Viable Product is the minimum features needed to make the checkers web app meet customer requirements. The MVP is defined by the Product Owner. The MVP will let two players sign in and play a game of checkers together.

MVP Features

1. Every player must sign-in before playing a game and be able to sign-out when finished playing.
2. Two players must be able to play a game of checkers based upon the American rules.
3. Either player in a game may choose to resign, at any point, which ends the game.

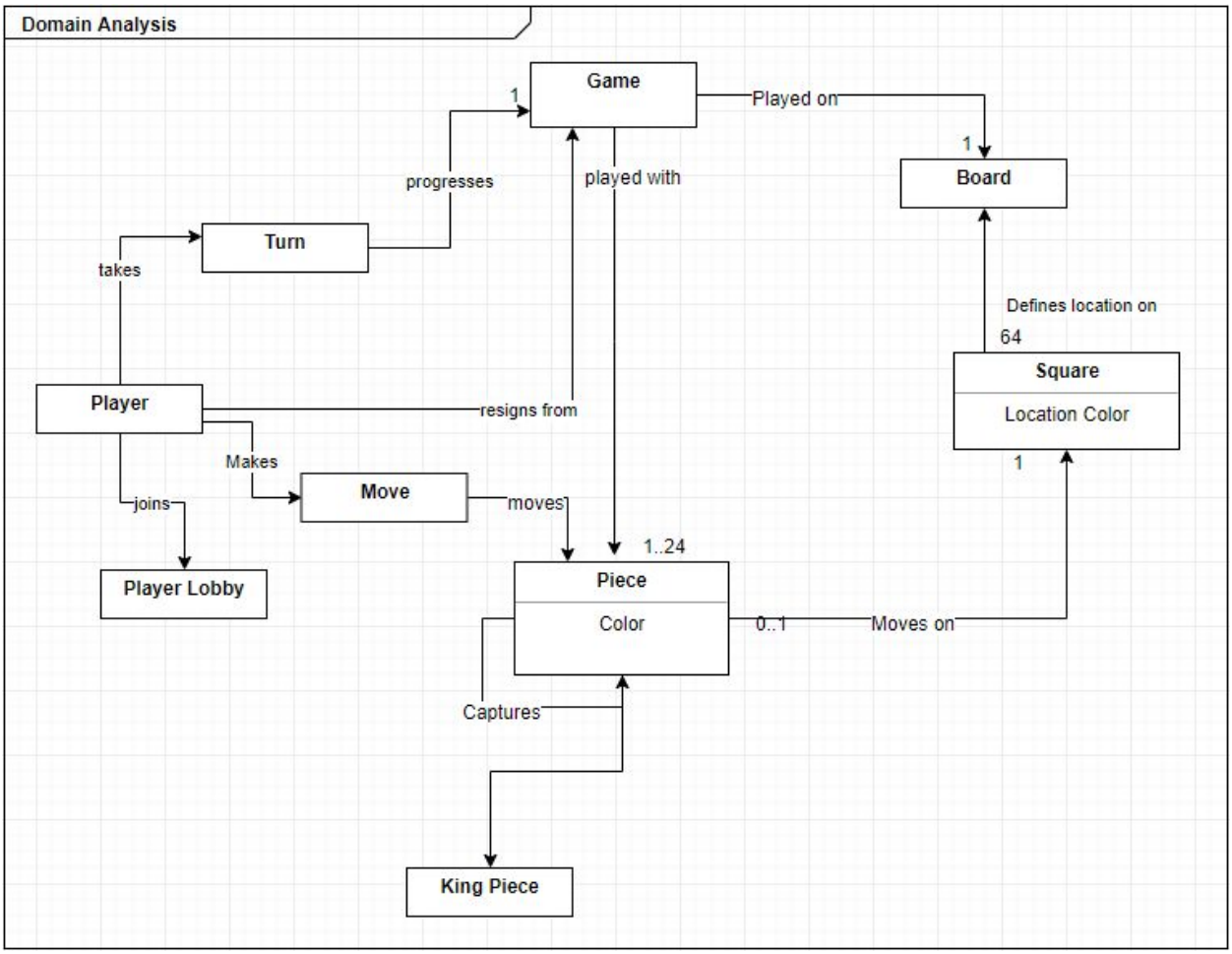
Roadmap of Enhancements

The Product Owner also requests the development team enhance the MVP with two enhancement features from a list of seven. The team selected:

1. Spectator mode: Other players may view an on-going game that they are not playing.
2. Replay Mode: Games can be stored and then replayed at a later date.

Application Domain

This section describes the application domain.



Overview of Major Domain Areas

The domain area is a game of American checkers. Two players play the game on an 8x8 board with alternating black and white pieces. Each player has 12 pieces, either red or black. The objective of the game is to capture the opponent's pieces. The following are the major domain areas in this application:

Player: The person who wants to sign in to play a game of checkers
Web Checkers: The website where players go to play checkers
Board: The game board on which the players play
Piece: The pieces that the players control.

Details of Domain Areas

A player captures an opponent's piece(s) by jumping diagonally over them. If a player's piece reaches the opposite side of the board, it becomes a king, and has greater movement options. In our model, the player to select their opponent goes first, and therefore is given the red pieces. A piece may only move diagonally forward unless it is a king (may move backward). If a piece can capture an opponent piece, that player must make the jump. When one player has lost all pieces, cannot make any moves, or forfeits, the game ends and their opponent wins.

Player

The players are people that want to sign into the Web Checkers website. After signing in, they can log out, wait for a player to want to play a game with them, or initialize a game with another player. The players can resign from the game at any time as they please.

Web Checkers

The web checkers application provides many useful features for its users. It allows them to sign into a player lobby to find or wait for opponents. The application also provides in game features for the players such as resigning from a game at any point. The application also controls the way players must make moves as players.

Board

The board is an eight by eight square of squares, which are alternatingly black and white. Checkers are initially arranged on both players' ends of the board but can only exist and move along the board's black squares. The board is reversed for the second player's view.

Piece

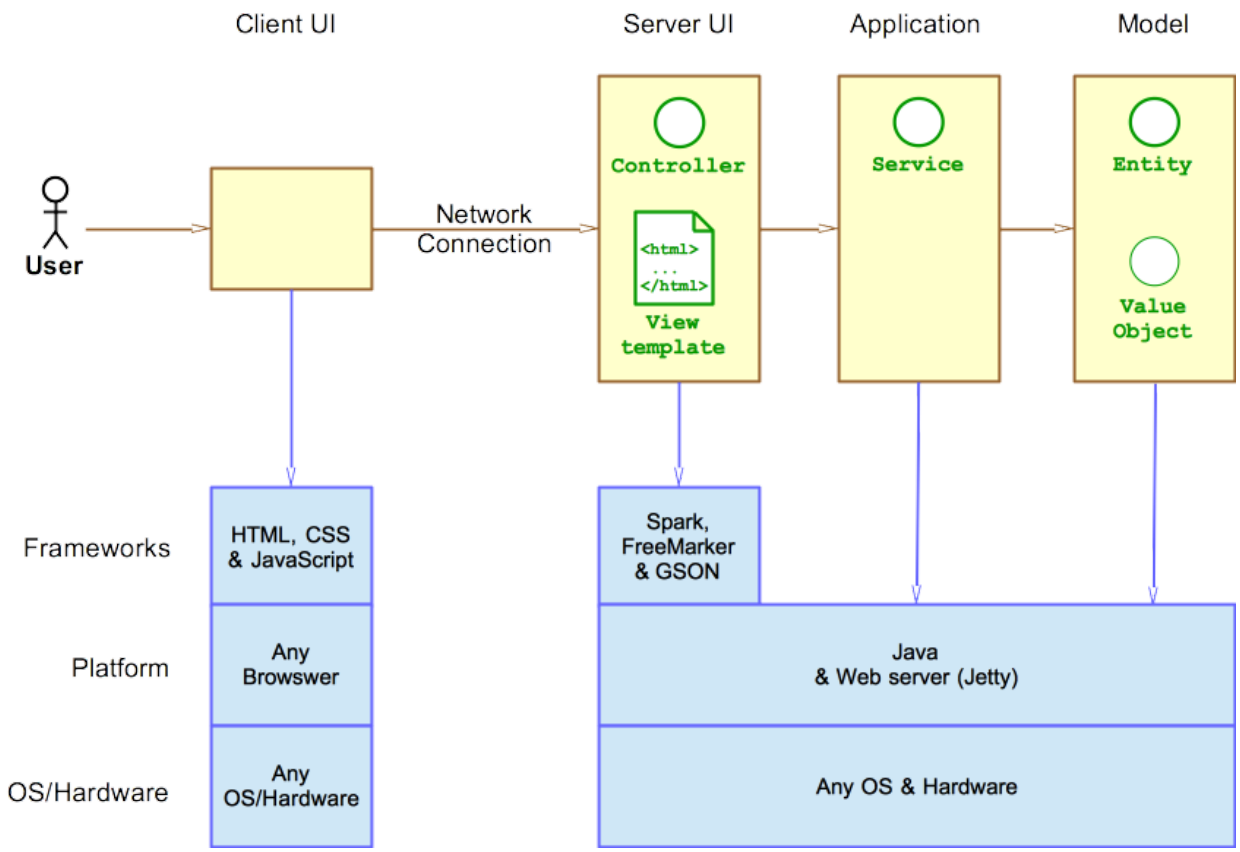
Each piece is either red or white. Pieces that share a color are controlled by the same player. Pieces are placed on the black squares of the board and can only move on black squares. A normal piece can move one square diagonally toward the other player's side of the board, or it can jump over an opponent's piece toward the other player's side of the board. A piece can be crowned (converted into a king piece) by moving it to the opponent's edge of the board. King pieces are not restricted in their direction of movement but are still confined to single and jump moves.

Architecture

This section describes the application architecture.

Summary

The architecture has three tiers: UI, Application and Model.



As seen above in the diagram, the Server UI uses Spark as the microframework and FreeMarker as the template engine. The Application uses Jetty for the web server that runs on Java. Because Java runs on all major operating systems, system calls are abstracted away from the responsibility of the system.

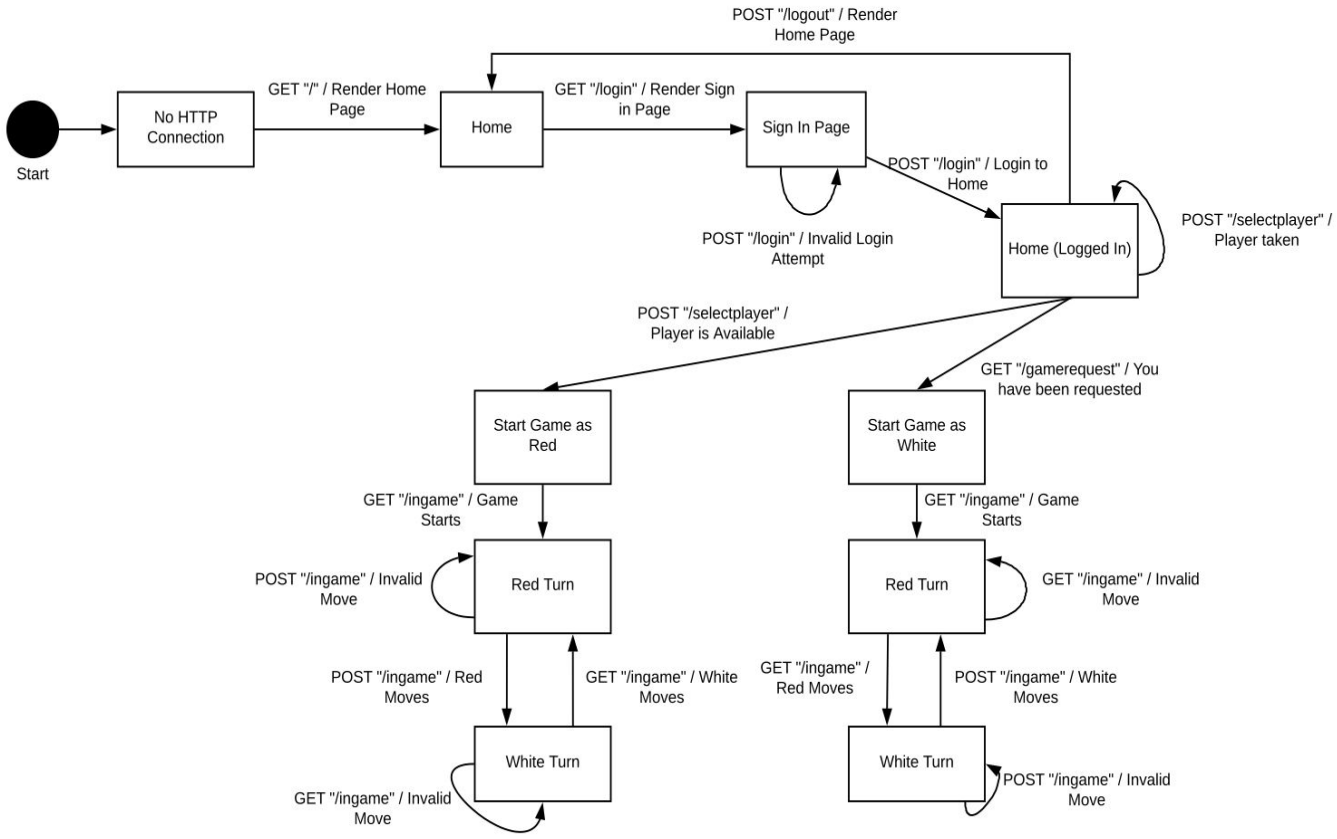
As a web application, the user interacts with the system using a browser. The client-side of the UI is composed of HTML pages with some minimal CSS for styling the page. There is also some JavaScript that has been provided to the team by the architect.

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.

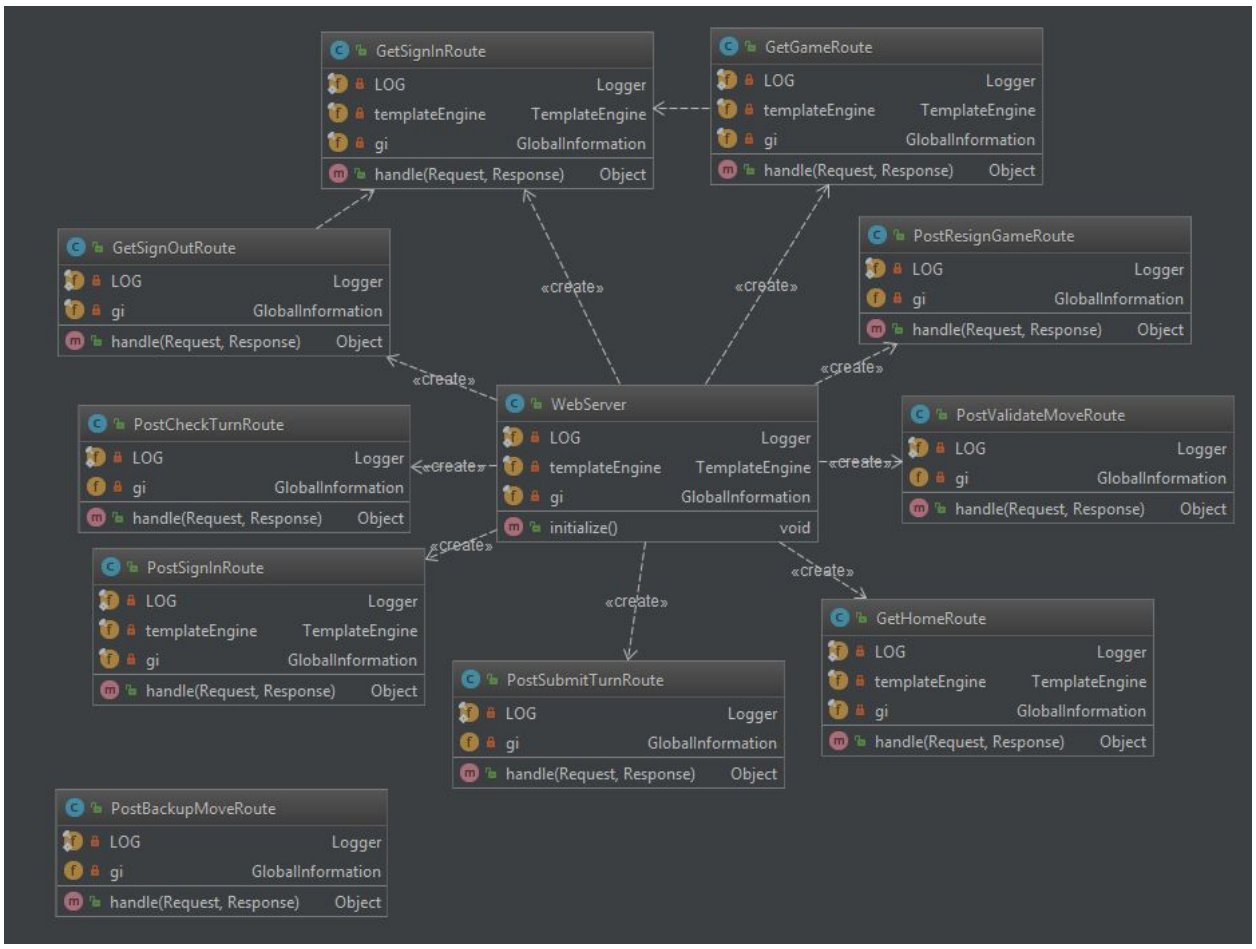
Overview of User Interface

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



UI Tier

This tier handles the user interface and client-side responsibilities of the web app. It consists of the game board display, the configuration file for Spark, and the controllers for the view-models. The game board display allows the user to have a familiar interface to play a game of Checkers. The following is a UML diagram that describes the relationships among the classes in the UI Tier.

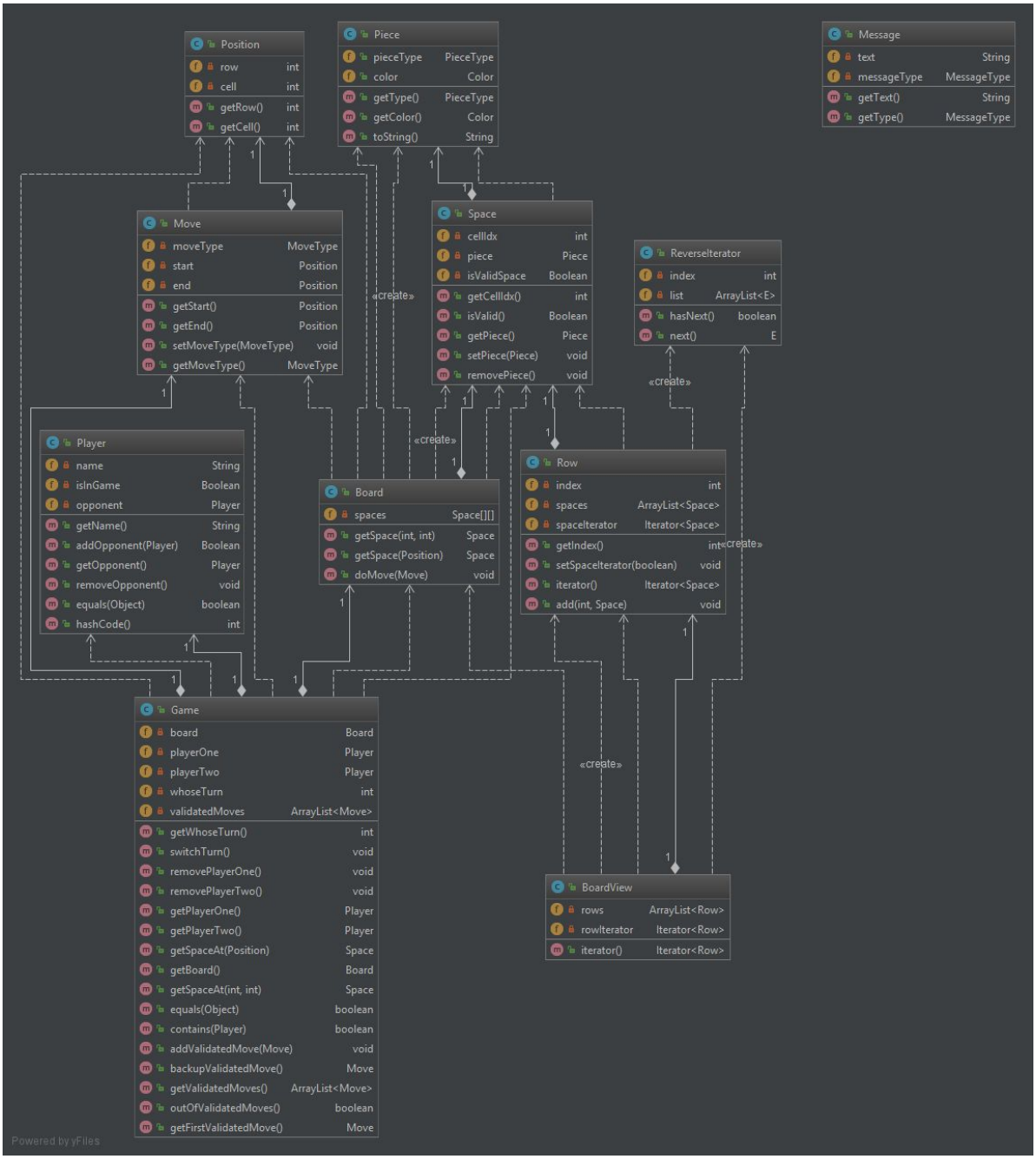


Application Tier

This tier currently holds the player logic. It contains the PlayerLobby. This tier handles server wide tracking of the users signed in, the games being played and has functionality for singing in players, signing our players, creating games, etc.

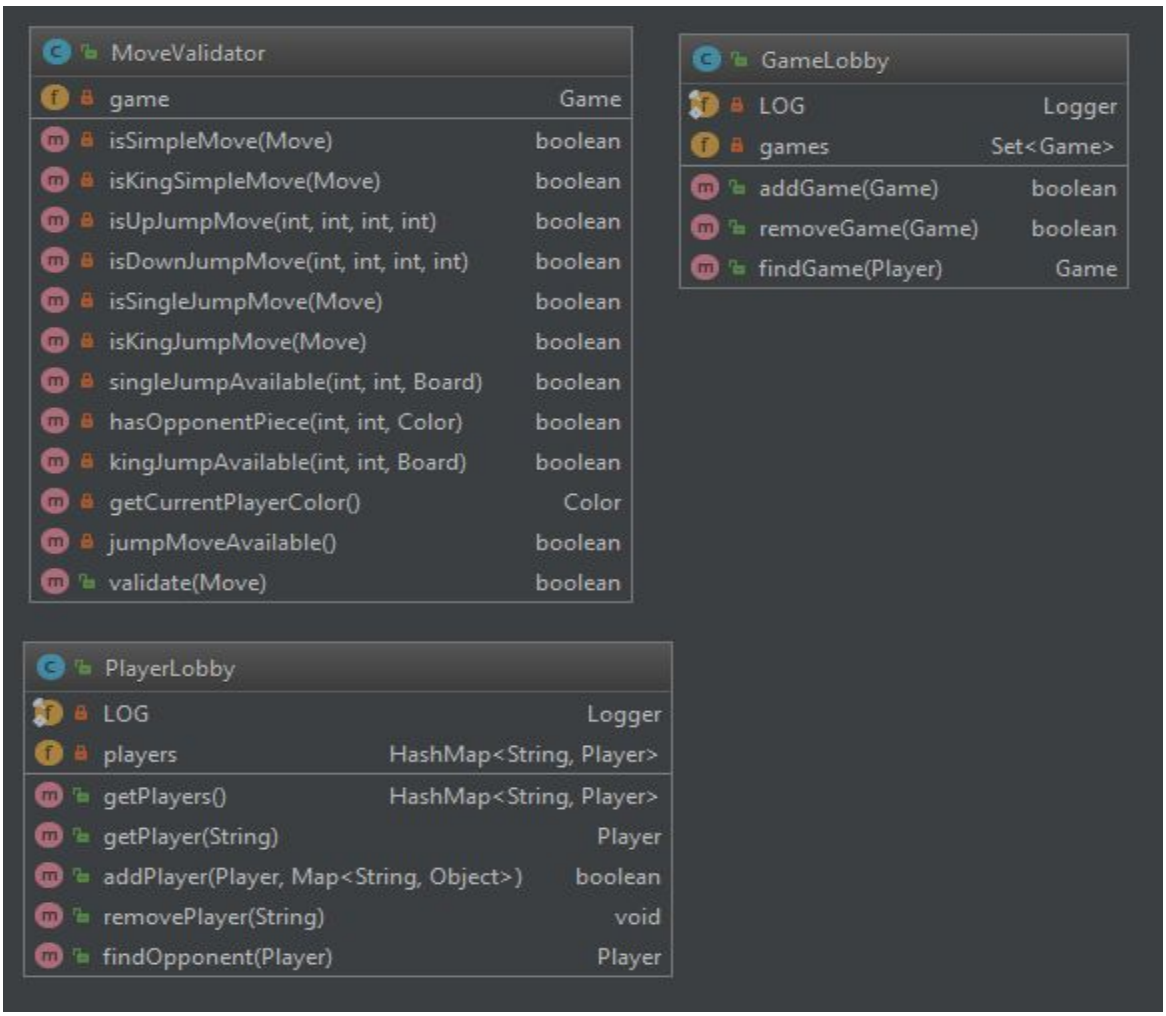
Model Tier

This tier holds the internal representation of the board, lobby and player. The model is responsible for representing the domain of a checkers game. This tier contains a class for the player, which is responsible for associating a name with a Player object. The model's board, row, square, piece, move, and position classes store and manipulate the internal representation of the play area. It also contains the game class, which is responsible for associating two players with a board object and maintaining that board. Additionally, it tracks which player is taking their turn and what color their pieces are. The turn class validates, and stores moves made during a player's turn.



Controller Subsystems

A sub-system would exist within one of the application tiers and is a group of components cooperating on a significant purpose within the application. For example, in WebCheckers all of the UI Controller components for the Game view would be its own sub-system. We have the PlayerLobby class that controls what is displayed on the homepage and controls signing in a player. The GameLobby class controls which games are displayed on the home page and controls whether a game is in session or not



Game Subsystem

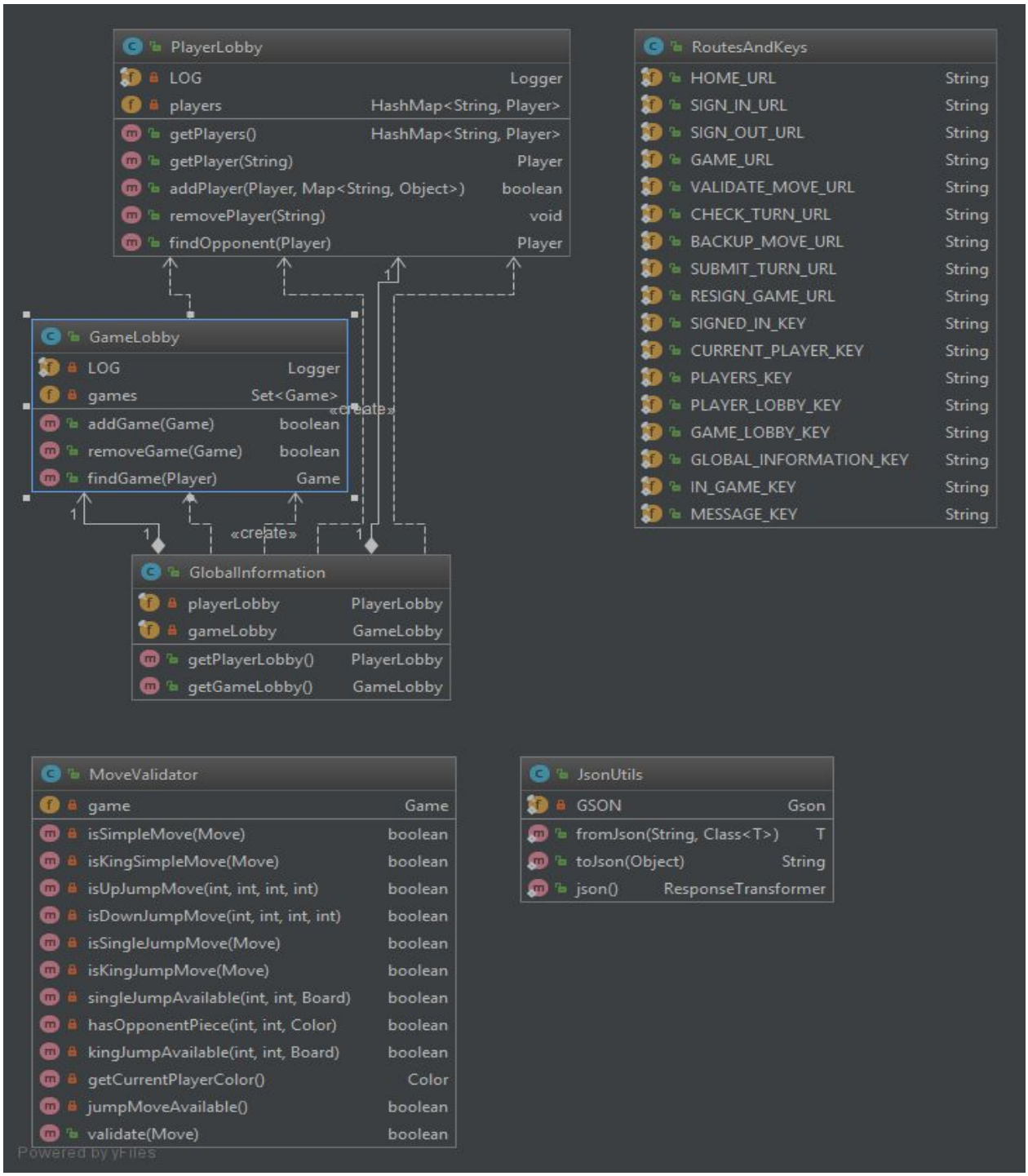
The game class stores two players, the colors assigned to those players, and a board, as well as information pertaining to the turn. Games are created by the WebServer and interact with board and turn objects to continuously progress the state of the game.

Purpose of the Sub-system

When a game is created, it initializes a board on which the two players can play and assigns a color to each player. The game is then responsible for starting and ending players' turns, as well as updating the state of the board after each turn ends. Finally, the game is also responsible for

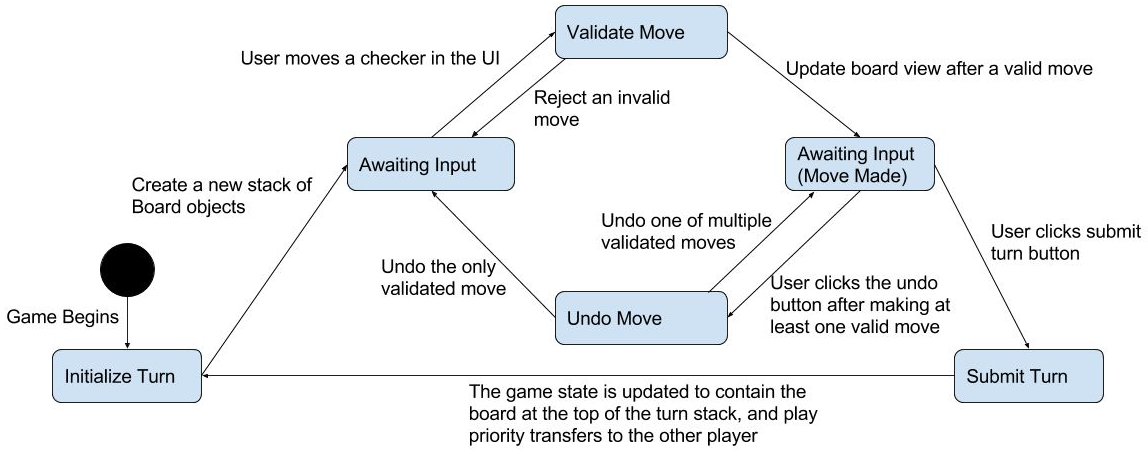
checking the state of the board and determining if the game is over.

Static models

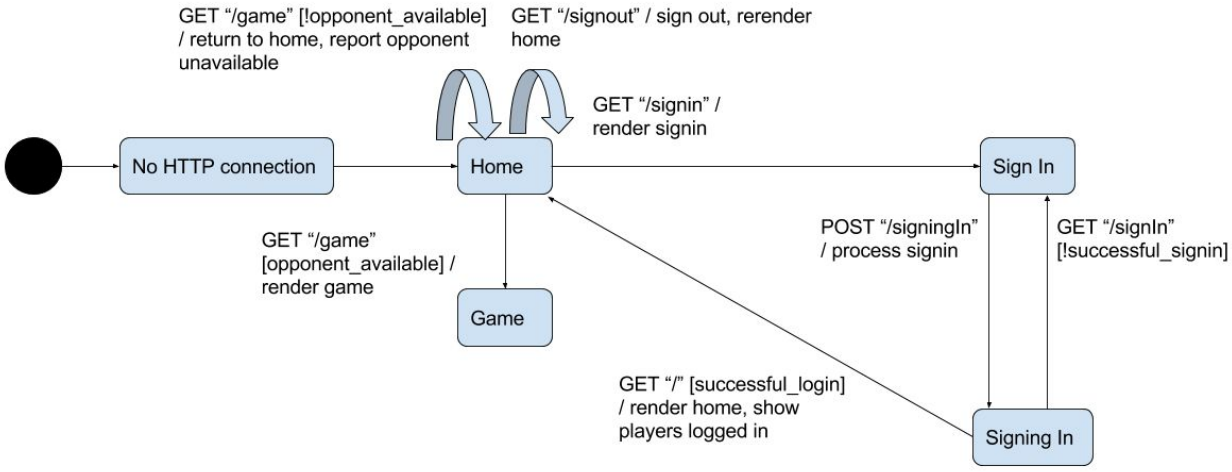


Dynamic Model

The turn state chart waits for the input of a player and to then validate the move if the input was a move or react to the button inputs to initiate an undo, resign, or submit turn.



The application state diagram is initiated when a player connects to the web checkers server. When the user calls different pages, the route classes reroute the user to those pages.



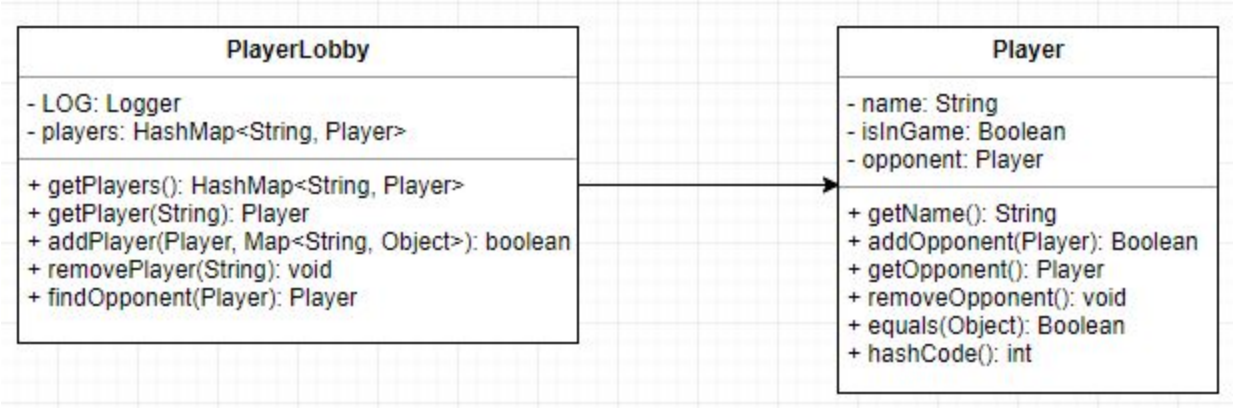
Sub-system Player Sign In/ Sign Out

Purpose of the Sub-system

The purpose of the Player Sign-in/Sign-out sub-system is to allow players to join a Checkers game and play against other players. Signing in will allow players to keep track of their game statistics, mathc records, and other data; signing out will help protect their accounts.

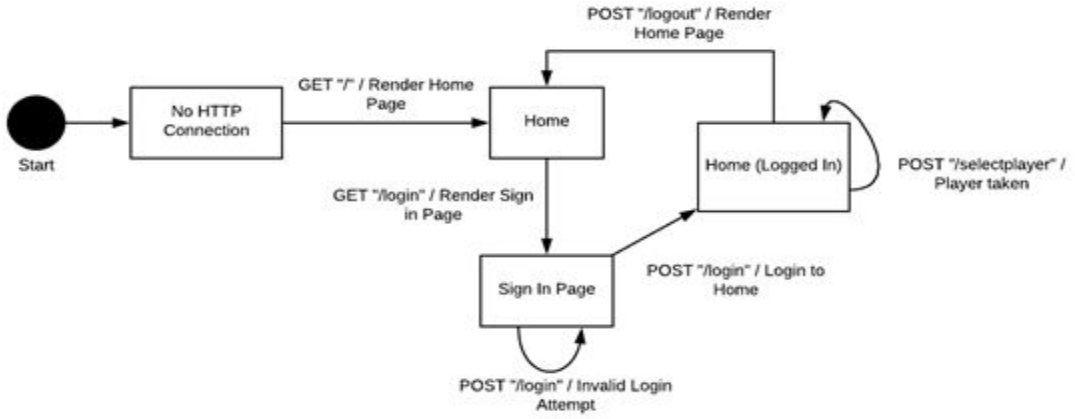
Static Model

The Sign in /sign out system interacts with the Player and PlayerLobby classes to store the session and validate the sign-in/ out. While the player is in the model tier to hold data of the actual player, the PlayerLobby is on the application tier because the whole application needs to know if a player is signed in or not and what the player's name is.



Dynamic Model

As part of the UI state diagram, this diagram shows how a player can easily sign in through the signin page and they will be logged in the home page as well. They can easily sign out from the homepage clicking on the signout button.



Design Comments

We are satisfied with our front-end design. All the spark and Ajax routes do what they need to do and follow the design principles. We were able to incorporate information hiding and information expert throughout the project by assigning data and functionality to the proper classes. We did not do any sort of inheritance in our project, expect for the spark routes but that was because we never needed to.

We observed the single responsibility principle very closely and had very high cohesion, but we are trying our best to enforce low coupling. We have realized that our move validator has way too many dependencies and are looking to start the next sprint by figuring a way around it.

We have improved our design from spring 1 trying to align ourselves to the design principles but this meant that we had to spend a significant time refactoring code. But we all agree that the time spent doing so and establishing a good foundation really helped everyone on track.

We used the controller principle in the PlayerLobby and GameLobby classes and pure fabrication in the roots and keys class.