WebCheckers Design Documentation

Modification Log

Date	Description	Author(s)
3/19/18	Initial Draft	Varnit Tewari (VT), Josh Abrams(JA)
3/27-29/18	Word Conversion, Complete for Sprint 2	JA

Team Information

- Team name: TheAdoptedFive
- Team members
  - Daniel Harrington
  - Josh Abrams
  - Thomas Morris
  - Varnit Tewari
  - Vincent Li

Executive Summary

The project is a multi-player web application of Checkers game. The application has a sign in page for every player after which the player can choose from a lobby who he or she wants to play the game of checkers against. After getting matched up with a player, the initial board of checkers is provided where players can play the game.

Purpose

This application allows you to play the game of American Checkers online so that the players don't have to physically meetup to play.

Glossary and Acronyms

Term	Definition
MVP	Minimum Viable Product
HTTP	HyperText Transfer Protocol
HTML	HyperText Markup Language

Requirements

This section describes the features of the application.

Nonfunctional

The project is required to be developed using Java 8 and Jetty (a Java-based server), which will allow it to be available at the URL, localhost:4567, when the server is running. Maven is a build tool for Java apps and is used to handle the dependencies for the project. The Spark Web micro framework and Free Marker template engine must be used to route the HTTP requests and create HTML responses.

Regarding extensibility, the project must be able to have possible enhancements added to it to improve the checkers players' user experience. In addition, the acceptance criteria must be able to be tested by other developers on the team and will only be tested by team members. It is not required to persist data across the web app shutdown and start.

Functional

- A player must be able to sign in with a unique player name and once signed in, is able to sign out from any page.
- Player/s must be able to play a game of checkers that follows the American rules using drag-and-drop actions.
- A player must be able to select which games they want to play according to their color preferences.
- A player must be able to resign from a checkers game at any time, which will cause the game to end.

Definition of MVP

A minimum viable product (MVP) is a development technique in which a new product or website is developed with sufficient features to satisfy early adopters. Therefore, the final, complete set of features for checkers game is only designed and developed after considering the customer requirements. The MVP for the product will allow two players to play a game of checkers with their own accounts.

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 of a game may choose to resign, at any point, which ends the game.

Roadmap of Enhancements

1. Spectator Mode: Other players may view an on-going game that they are not playing.
2. Player Help: Extend the Game View to support the ability to request help.

Application Domain

This section describes the application domain.

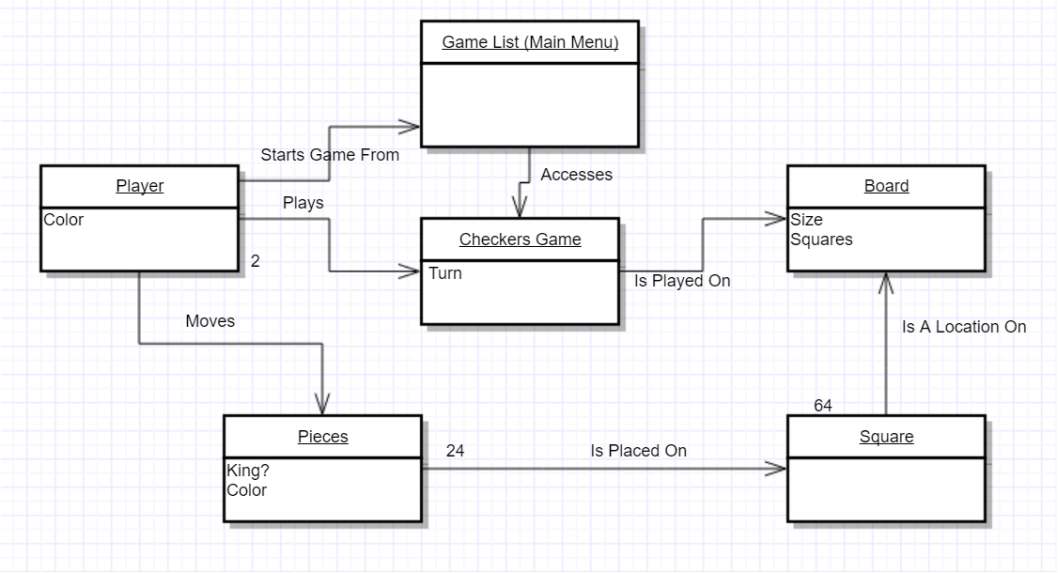


Figure 1: The WebCheckers Domain Model

The domain area is an American game of checkers. It consists of two players playing on an 8x8 board with 12 checkers pieces, either red or black, each. The objective of the game is to stop the opponent from moving or having them forfeit. Players can capture pieces by jumping over them and gain an advantage by capturing their pieces. They can choose which color they want to start with, since whoever is red makes the first move. Pieces only move diagonally on the black squares and must make a capture if it is possible. The game ends when one player has all their pieces captured, they cannot make any moves, or they choose to resign from the game.

Architecture

This section describes the application architecture.

Summary

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

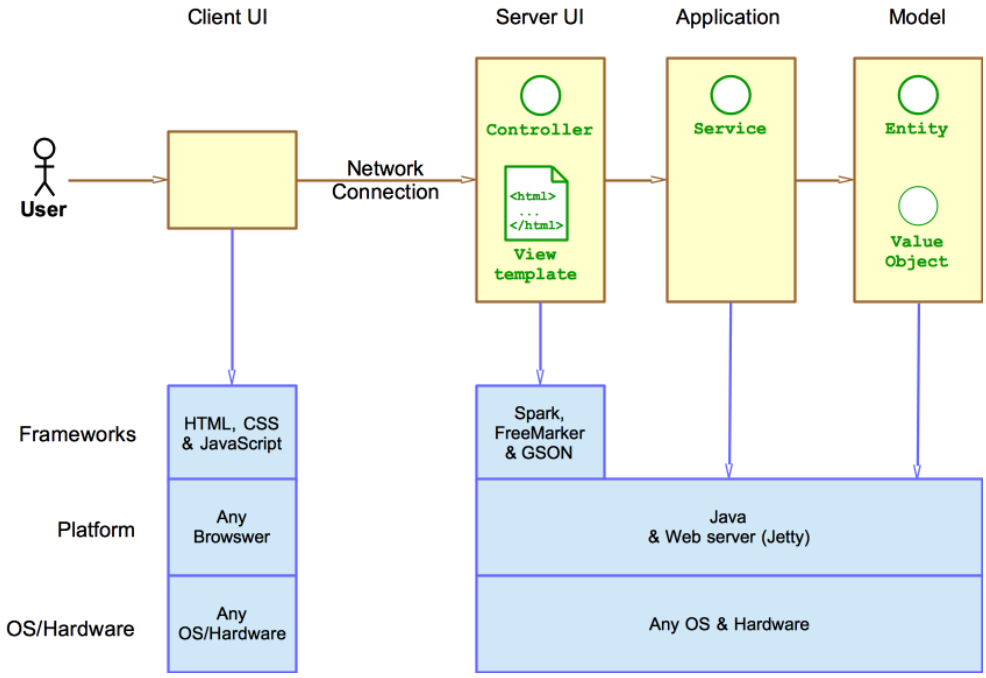


Figure 2: The Tiers & Layers of the Architecture

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.

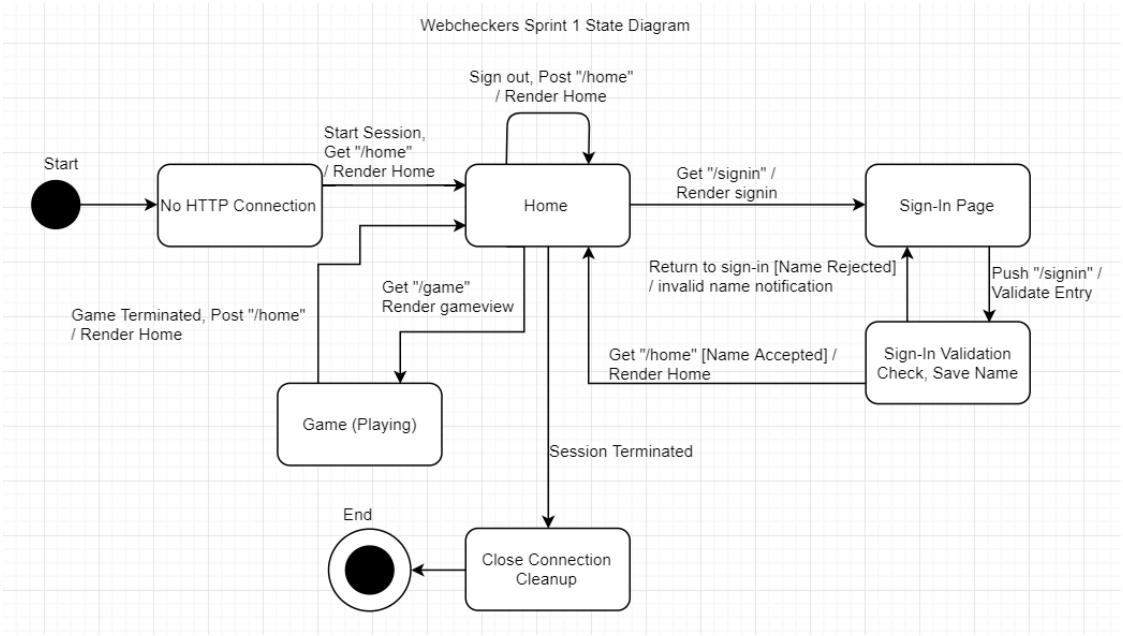


Figure 3: The WebCheckers Web Interface Statechart

As seen from the diagram, there are three rendered web pages: Home, Sign-in and Game. The Home page can be reached from all of the three above if a certain action is taken. A player can only access the Game and Sign-in pages once they have passed the Home page. The Home page's contents depend on the sign-in state of the player, with added functionality rendered when returning to home as a signed-in player. This includes the options to start games, which renders the gameview. The Home page represents the central location of user interaction, where all actions are ultimately being initiated from and returning to until the session is terminated.

UI Tier

The UI Tier's functionality was expanded into a Controller system, instead of the traditional Get/Post Routes. For this section, references to Controller systems are assumed to be under the UI Tier of responsibility.

This tier is responsible for user interface and client-side actions during the web app's operation. It handles the requests and rendering functions needed for displaying and transitioning between the various pages making up the application. Controllers in this tier coordinate with and access data structures and validators in the other tiers to properly gather and act on the data required for a robust user experience when accessing the WebCheckers application and playing games.

Dynamic models

A basic view of the process of starting a game is shown in Figure 4 below.

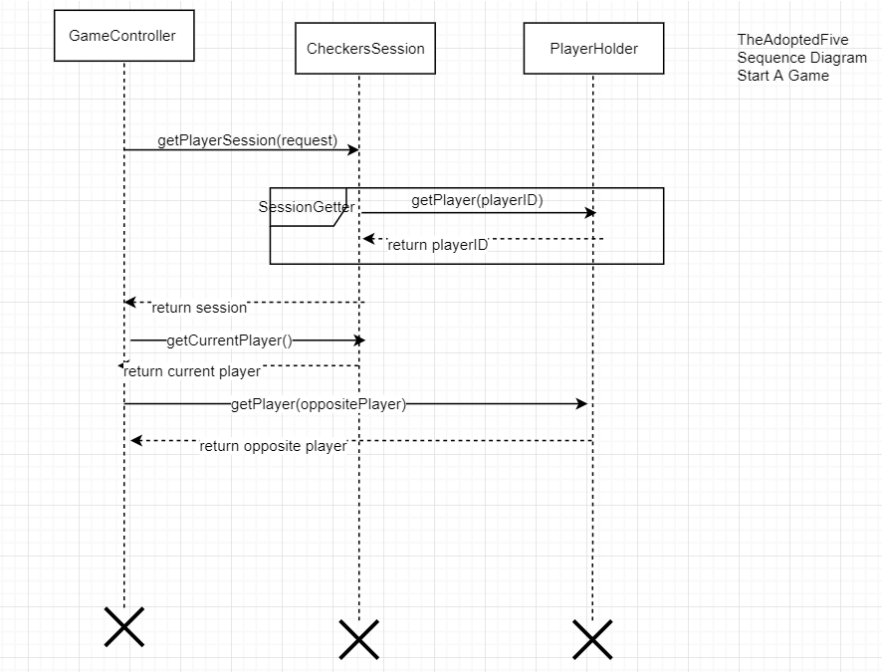


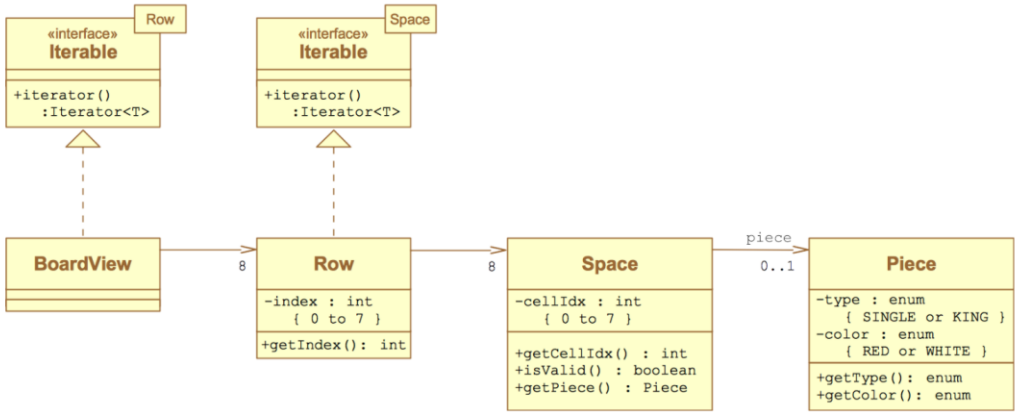
Figure 4: Sequence Diagram for Starting a Game

This sequence diagram depicts some simple interactions between components while a game is started. The CheckersSession handles information regarding the current user session, though game session information has since been expanded into a unique GameSession handler. This session is able to get the current player information for the GameController, using the PlayerHolder object that tracks all the users logged on to the application. The GameController also goes to the PlayerHolder object directly to retrieve the desired player in order to start a game with two players.

Application Tier

The Application Tier holds the basic structures and logic that the application is built upon. This includes the base board implementation, an iterable collection of squares in an iterable collection of rows, the Player representation, Piece representation, and other helpful classes used by top-level functions to complete their tasks.

Static models



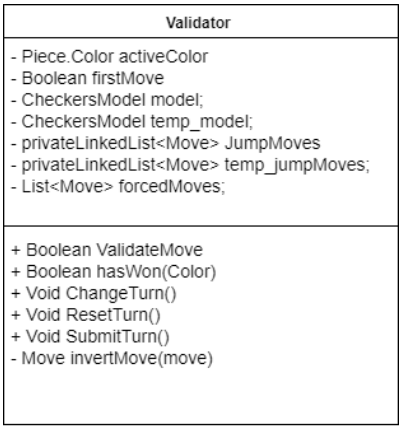
UML Diagram of Basic Application Classes

This diagram shows a section of the Application-tier classes. These classes represent the lowest-level components of the Board object, intended for building and rendering the board to the user. The most basic component is the Piece, which is the tool with which the game is played. It has two different characteristics, which are obtained by getter functions, in order to reflect a basic understanding of the game's state. Pieces occupy Spaces on the board, which are the individual squares making up each row. To account for Checkers rules about piece placement, a valid flag is included in Space to indicate whether or not the Space can legally hold a piece. Rows are iterable objects containing eight spaces, and a board, the BoardView, is an iterable object containing eight rows. The iterability is required for ease of rendering the board section by section. The interaction between these components is critical to the underlying representation of the board.

Model Tier

The Model Tier contains the internal board/game representation, interacting closely with the requests and updates coming from players as games progress. It uses a new board representation made up of Positions and Squares to track the game state. A Validator object was created here to be accessed when players attempt to make a move/jump, where the desired move is validated against internal piece movement algorithms that determine whether or not the player chose a valid move. This validation functionality works directly with game session handlers to update the internal and displayed board for both players on valid moves.

Static models



Validator Class Diagram

This class diagram for the Validator class provides a look at the data stored and functionality available through this model-tier element. It holds information about the state of the game, such as which player is currently active, and is able to track the state within a turn to reflect forced-jump rules. The methods implemented are used to validate a given move to update the board, observe winning conditions, and coordinate turns. A private helper function, invertMove, is used to reconcile the internal board's layout with the perspective of the opposing player, as the board is effectively inverted for the opposite player.

Sub-System: PlayerHolder

This section describes the PlayerHolder sub-system. PlayerHolder is a data storage class directed at holding user data for sign-in validation and game availability information. It holds a list of all the players that have logged on to the application. This allows the sign-in process to use PlayerHolder to check the desired name against those already entered, to see if the name is taken before creating a new player. It is also used when rendering the list of players available to play games.

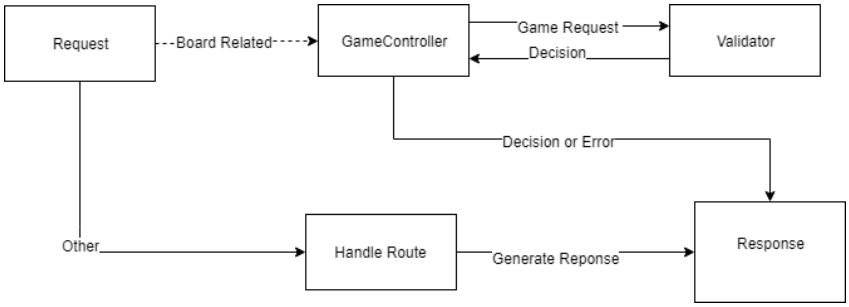
Dynamic Models

Figure 4 showed a selection of interaction between the PlayerHolder and the controller/sessions involved with starting a new game. The PlayerHolder object is very important for storing the list of players, which allows their sessions to be binded as both are pulled into the game page. HomeController also interacts heavily with PlayerHolder in order to obtain a list of the available players. This functionality is also extended to reflect players who are already in a game and cannot be called into a new game.

Sub-System: Piece Movement & Validator

The piece movement subsystem is designed to take in a desired move from the player and validate it. The move, or jump, is checked against an internal model of the board and game state in order to verify that it is a valid move. The primary interface for movement validation is a Validator object, instantiated on the start of a game. This object holds information about the state of the game, such as turns, pieces left, and internal board models. This information is passed to the validation function, which uses a

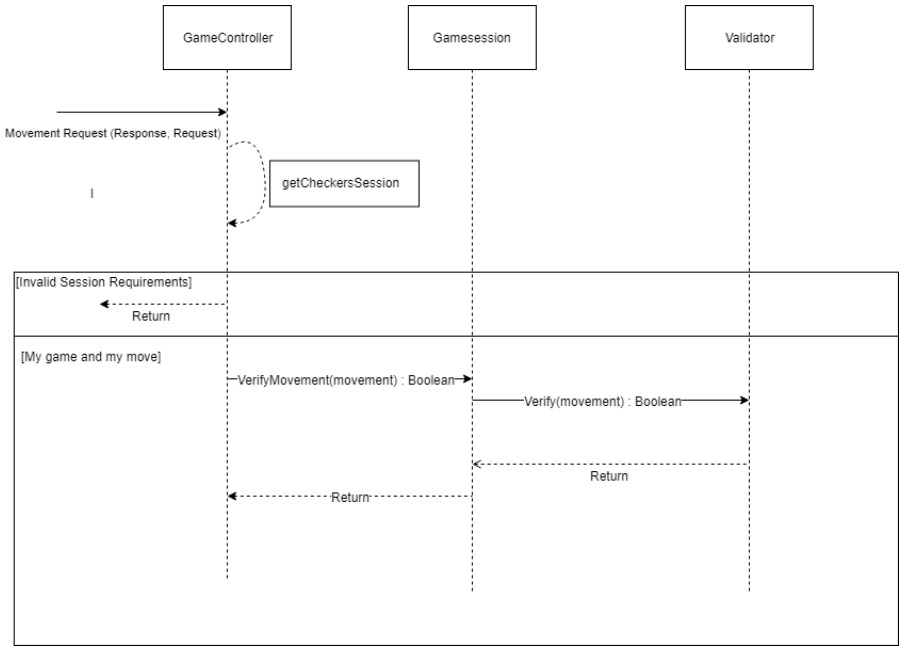
variety of piece movement algorithms to determine whether or not a given move is valid for a player. The validator interacts with the game session to account for multi-jump scenarios, where after a move the player must still have the Turn.



Flow of data starting with a Game Request

This figure models the interactions between the session routes/controller actions and the Validator in the process of determining move validity and building a response to update the board.

Dynamic Models



Simplified Sequence Model of Game-Validator Interaction

This sequence diagram shows how information stored and calculated in the Validator is passed to the GameSession and GameController, allowing it to render the game with updated information after a valid move, or reset the moved piece if the move was invalid. The GameSession acts as the representative for the game, providing access to the Validator and other internal data, and interacting directly with controllers to process and update the state of the game.