geometry: margin=1in

PROJECT Design Documentation

Team Information

- Team name: Team Rhamz
- Team members
  - Anh Nguyen
  - Rhamsez Thevenin
  - Sierra Tran
  - Phil Ganem
  - Mohammed Alam

Executive Summary

WebCheckers is an application that users can sign in to play online Checkers with other players. Using the Spark framework for Java, WebCheckers provides users with a realtime experience challenging an opponent.

Purpose

> This project is a web-application of checkers. Users should be able to log-in, and play against other players.

Glossary and Acronyms

> *Provide a table of terms and acronyms.*

Term	Definition
UI	User Interface
POJOs	Plain Old Java Objects
MVP	Minimal Viable Product

Requirements

This section describes the features of the application.

> *In this section you do not need to be exhaustive and list every story. Focus on top-level features from the Vision document and maybe Epics and critical Stories.*

Definition of MVP

> *Provide a simple description of the Minimum Viable Product.*

MVP Features

> *Provide a list of top-level Epics and/or Stories of the MVP.*

Roadmap of Enhancements

> *Provide a list of top-level features in the order you plan to consider them.*

Application Domain

This section describes the application domain.

The WebCheckers Domain Model

The domain highlights the general flow of the application:

- The user signs in to be at the home page (menu)
- The user can then start a game (becoming a Player) or check player stats for competitive mode
- The player can make a move to play the game
- The move communicates with the model which includes the Board, Row, Space, and Pieces
- The player plays the game

The most important entities of the domain model are the Player, Board, Row, Space, and Piece entities.

- Each set of 24 checker Pieces on the board represents a Player in the game. Players take a turn when they make a move which is played with Checkers which are on the board.
- Regular or King Pieces are on specific Spaces which belong to Rows which also builds the entire Board. In short, there can be zero or one Piece on each Space, there are 8 Spaces on each Row, and

there are 8 Rows on a board.

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.

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.

The WebCheckers Web Interface Statechart

> *Provide a summary of the application's user interface. Describe, from the user's perspective, the flow of the pages in the web application.*

So we start with our initial condition if the GET request doesn't render the page correctly. We then load on the sign-in page with GET "/" render sign-in. The sign-in page state was initially made with a piece of account information in mind, so if the user doesn't enter the right password, we would get the sign-in page again for them to input their information once again.

Once the player successfully signs in, they are redirected to home, with the Post sending their info to check if sign-in is correct, We will be getting home after that. Once in the home page state, the User will have options such as a button that will make a get request for a page that will display the user's stats.

On the Home page, there will be an option to click the other player's button and be redirected into a game using the GET "/game request". In the game state, You will have several requests and options.

Here in the Game state You can make Post requests trying to move checker pieces, and you will either have a GET request to update the board on your UI based on the board class, and we will be posting information such as the space you moved to. Based on whether or not it's your turn, your opponent will still be playing and modifying the board so we will have GET requests to update the Board as such from the Opponent's turn state.

There is also a Help Menu state on the game, where you can pause your game and go into the Help Menu state which you access with a GET request. In that state you can open another state called Pause game which will allow you to pause the game if you ever need to take a break This is done by a GET request to the pause state and another back to the help menu state. Another option from the Help Menu is the Hint state, so if you're stuck on Checkers a Hint will be shown and will help you decide your next move. We will be sending a GET request for the Hint state, and from the Hint state we will be Posting information, or in our case, the Hint the player can do. After that, we can just close Hint by doing another Get request of the Help Menu, and another Get request to go back to our game will be played.

So now we are playing the game and repeating the aforementioned requests. We now end up to where the game is finished. When the Game is finished we GET the Game Finished State, basically a screen showing the game has stopped and whatever the result is, then we POST information about the players into the Competitive stats page state that was mentioned before. This will update their wins/losses and other details. This can always be accessed back on the Home page. In the Finished game screen state, you have the option of wanting to go back home or directly to the user stats page.

UI Tier

> *Provide a summary of the Server-side UI tier of your architecture. Describe the types of components in the tier and describe their responsibilities. This should be a narrative description, i.e. it has a flow or "story line" that the reader can follow.*

> *At appropriate places as part of this narrative provide one or more static models (UML class structure or object diagrams) with some details such as critical attributes and methods.*

> *You must also provide any dynamic models, such as statechart and sequence diagrams, as is relevant to a particular aspect of the design that you are describing. For example, in WebCheckers you might create a sequence diagram of the `POST /validateMove` HTTP request processing or you might show a statechart diagram if the Game component uses a state machine to manage the game.*

> *If a dynamic model, such as a statechart describes a feature that is not mostly in this tier and cuts across multiple tiers, you can consider placing the narrative description of that feature in a separate section for describing significant features. Place this after you describe the design of the three tiers.*

Application Tier

> *Provide a summary of the Application tier of your architecture. This section will follow the same instructions that are given for the UI Tier above.*

Model Tier

> *Provide a summary of the Application tier of your architecture. This section will follow the same instructions that are given for the UI Tier above.*

Design Improvements

> *Discuss design improvements that you would make if the project were to continue. These improvement should be based on your direct analysis of where there are problems in the code base which could be addressed with design changes, and describe those suggested design improvements. After completion of the Code metrics exercise, you will also discuss the resutling metric measurements. Indicate the hot spots the metrics identified in your code base, and your suggested design improvements to address those hot spots.*

Testing

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

Acceptance Testing

> *Report on the number of user stories that have passed all their acceptance criteria tests, the number that have some acceptance criteria tests failing, and the number of user stories that have not had any testing yet. Highlight the issues found during acceptance testing and if there are any concerns.*

Unit Testing and Code Coverage

> *Discuss your unit testing strategy. Report on the code coverage achieved from unit testing of the code base. Discuss the team's coverage targets, why you selected those values, and how well your code coverage met your targets. If there are any anomalies, discuss those.*