Webcheckers Design Documentation

Couch Coders

Modification Log

Date	Version	Description	Author(s)
03/19/19	1.0	Initial Draft	Patrick McCullough
03/23/19	1.1	Added Player Sign-in/out and Start a Game to **Requirements**. Added **Definition of MVP.**	Patrick McCullough
03/23/19	1.2	Completed **Definition of MVP, MVP Features,** and **Enhancements.** Inserted diagrams and wrote summaries for **Application Area Summary, Domain Area Detail,** and **Overview of Application Interface.**	Patrick McCullough
03/26/19	1.3	Updated **Application Interface** diagram	Patrick McCullough
03/27/19	1.3.2	Submitted to git	
03/31/19	1.5	Added diagrams to **Application Interface** and **Model Interface.**	Patrick McCullough
03/31/19	1.6	Reviewed document, fixed grammatical inconsistencies, and suggested changes to wording.	Adam Wolf
04/01/19	2.0	Finalized submission for Sprint 2. Added diagram to **Overview of User Interfaces** and **Overview of Model Interfaces.**	Patrick McCullough
04/13/19	2.1	Updated **Acceptance Criteria,** and **Requirements.**	Patrick McCullough
04/16/19	2.2	Added diagram and description to **Overview of Client UI Interface** and updated **Acceptance Criteria**	Patrick McCullough

Team Information

Team Name: Couch Coders

Team Members: Ayush Rout
Mike Hurlbutt
Taylor Higgins
Adam Wolf
Patrick McCullough

Executive Summary:

The Web Checkers application provides users with a fun and easy-to-use program that allows them to sign into an account and play a game of checkers against an opponent of their choosing.

The client program is written using HTML, CSS, and Javascript, while the server-side functionality utilizes the Spark and FreeMarker frameworks. Through these we have a clean, functional application that is able to run on any operating system through any browser.

Purpose:

This document outlines the functionality of the Web Checkers application, and how it operates on a high level. As the code nears completion, the relevant portions of the document are updated with detailed summaries of each component, as well as diagrams detailing the execution of the code and how the pieces interact with each other.

Glossary and Acronyms:

Term	Definition
UI	User Interface
HTML	Hypertext Markup Language (HTML) is a programming language used in the creation of web pages and web applications
CSS	Cascading Style Sheets (CSS) are used to dictate the formatting of a document written in HTML
Minimum Viable Product (MVP)	The base product being delivered. In this case, the MVP is the completed Web Checkers application.
User Story	One of the key elements in Agile development, user stories outline the components that go into the projects.
AI/Bot	An artificial intelligence designed to play a game of checkers.

Requirements:

Player Sign-In/Sign-Out

A player must log in with a valid name that is not already in use. If that name contains a non-alphanumeric character, then that name is not considered valid.

- Once a player signs in, they are shown other players currently logged in. They have the ability to choose an opponent from this list to play against.
- When a player signs into their account, a button labeled sign-out replaces the sign-in button. When clicked, this button signs the player out and frees their name.
- If a player is signed into the game, they are able to see a list of other players who are also logged into the client. However, if the player is not logged into the game the only thing they are shown is the number of players currently logged in.

Start a Game

- A player is able to challenge another player after logging into an account by clicking on their name.
- If the player being challenged (the opponent) is already in a game, the system will return the challenger to the home page and display an error message.
- If the player's challenge successfully goes through, they will be placed into a game with their opponent. They will be assigned the color red, and their opponent will be assigned the color white.
- If the player is challenged, they will be brought into a game with their opponent. The player that initiated the game (the opponent) will be assigned the color red, and the player will be assigned the color white.

Playing a Turn State

- Implement *POST /validateMove*, which triggers when a player drops a piece into an open space. This allows the game to tell whether a move made by the player is valid or not.
- Implement *POST /backupMove,* which triggers when the Backup button is clicked. This allows a player to "take back" their most recent move in the event they would like to rethink their decisions.
- Implement *POST /submitTurn,* which triggers when a move has been validated and the player wishes to end there turn. This allows the player to confirm their moves, and pass the turn over to their opponent.

Waiting for Turn State

- Implement *POST /checkTurn,* to see if the opponent has submitted their turn. Each time *POST /checkTurn* is sent to the server, it checks to see if the opponent has sent *POST /submitTurn.*
- Create a 5-second waiting period every time */checkTurn* alerts the player that the opponent's turn is ongoing.

Resigning From a Game

- Implement *POST /resignGame,* an action that tells the server that the player is resigning from the game.
- The Resign button should only allow a player to resign from a game if they are in the Empty Turn state.
- The Resign button should be disabled once a user makes a valid move. The only way to reactivate the button is to either submit the turn, or back up to the most recent Empty Turn state.

Promoting a Piece

- When a checker reaches the opposite end of the board, it should be marked as a king.
- When the checker is marked as a king, its appearance should change to that of a "kinged" checker.
- When a checker is marked as a king, its movement options should be expanded to include backwards movement.

Validating a Move

- Create a *Position Data Type,* which records the row and column of a space on the board as an integer from 0-7.
- Create a *Move Data Type* as outlined in section 2.2 of the Sprint 2 documentation. *Move* contains two position data types, recording the starting location and the ending location of the proposed move.
- Implement *POST /validateMove,* which is used by the server to keep track of each proposed move during a single turn of the user's game state.
- Implement *POST /checkTurn*, which is used by the server to check to see if the opponent has submitted their turn.

Jumping a Piece

- Implement *POST /submitTurn,* which submits the user's turn to the server.
- Update the model with the moved piece and the removed piece.

Definition of MVP:

The Minimum Viable Product of the Web Checkers application is a system that allows players to play a round of checkers against another human being. The application includes a matchmaking system that allows players to log in using a name of their choosing, and select who they wish to play against.

The gameplay adheres to the standard rules of checkers. Pieces are able to jump each other, and in some cases will make multiple jumps in a turn if possible. When a checker makes it to the end of the board opposite to the side it started on, it will be “kinged”. A “kinged” checker is able to make moves both forwards and backwards, as opposed to the standard checker which can only progress forwards from it’s starting location.

MVP Features:

- Every player must be able to sign-in before playing a game, and must be able to sign-out when they have finished playing.
- Two players must be able to play a game of checkers based on the American rules for the game.
- Each player may chose to resign at any point, ending the game.

Enhancements:

- **AI Player:** An autonomous opponent that the player can choose to challenge in place of another human.
- **Replay Mode:** A game can be stored, and replayed at a later date.

Application Domain:

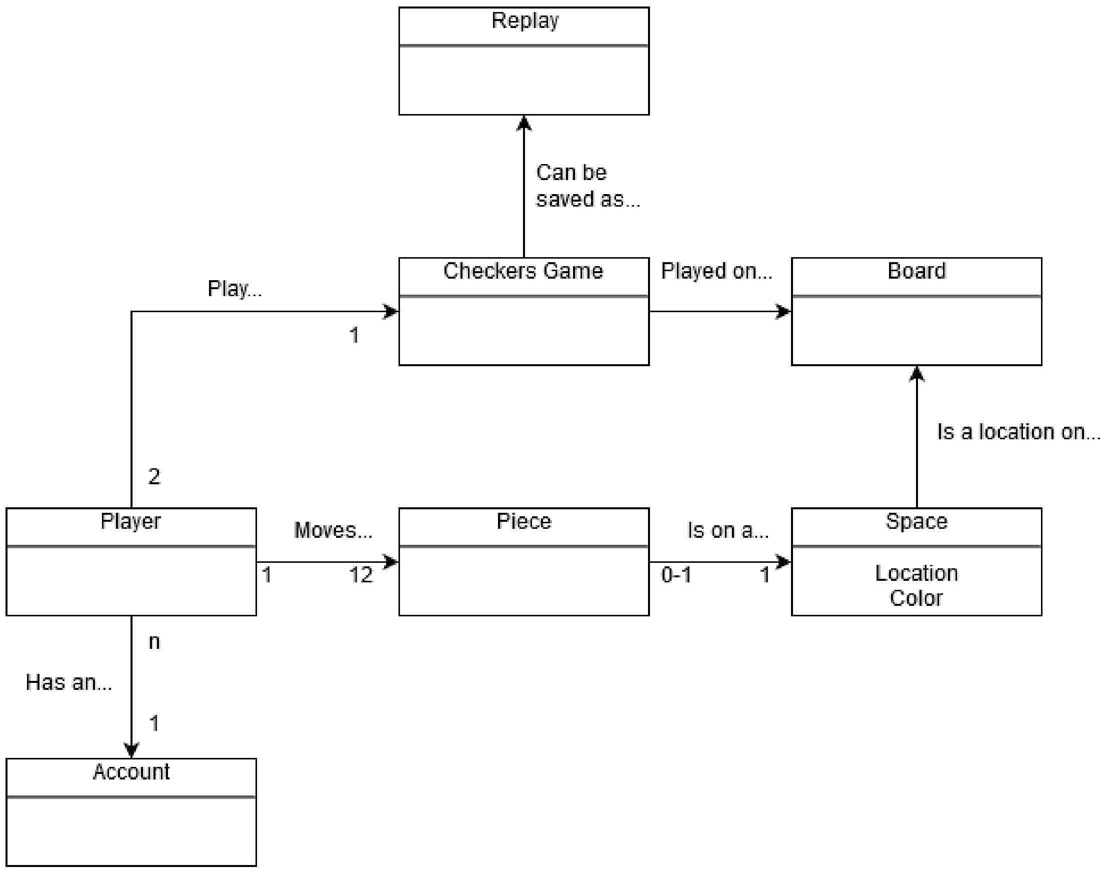
Overview of Major Domain Areas

The major domain areas of the project includes the classes representing the Players, the game Board, and the Checkers Game. Every Player class has an associated account, an object that registers a unique name for the human behind the screen. When two Players challenge each other, they are placed into a game which creates the associated board.

Domain Area Detail

The domain diagram details the high-level operations of the Web Checkers application. A game of checkers is played between two people on a single board. The board is an 8x8 grid of alternating white and black spaces, forming a checkerboard pattern. Each of the 64 spaces can hold a single checker (also known as a piece), and each player controls 12 pieces.

Each player has a single account that is identified by a unique name. Once two players have logged into their accounts, they are able to challenge each other to a game of checkers.

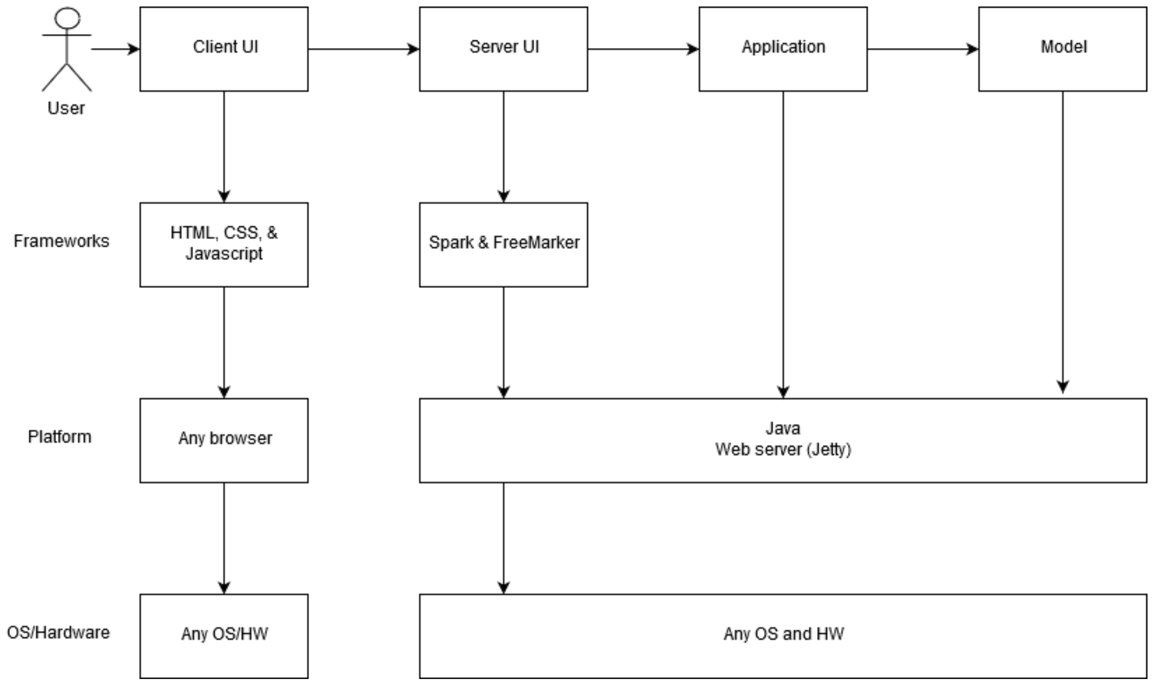


Application Architecture:

Summary:

The architecture of the application is a high-level breakdown of the system into its key pieces. The diagram is split into tiers, showing the flow of information between pieces. The User interacts with the Client UI, which in turn talks to the Server UI. The Server UI interacts with the Application tier, which holds the logic that controls the flow of the application. Finally, the Application tier passes information on to the Model, which holds the underlying logic that actually runs the Web Checkers application.

These tiers are further divided into layers that display the components of each tier. By following the arrows down, we can see what frameworks we are building each component of the project from.

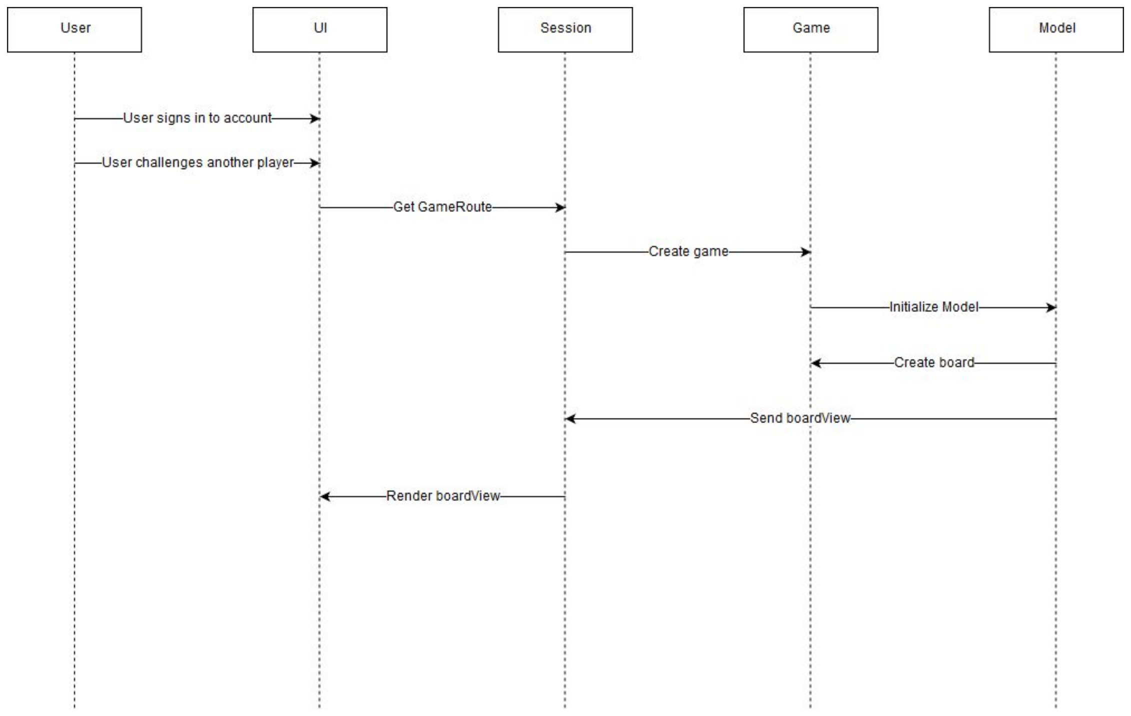


Overview of User Interfaces:

The User Interface of the Web Checkers application is the part of the program that allows the user to interact with the game. When a player first logs into their account, a player class is created to represent them.

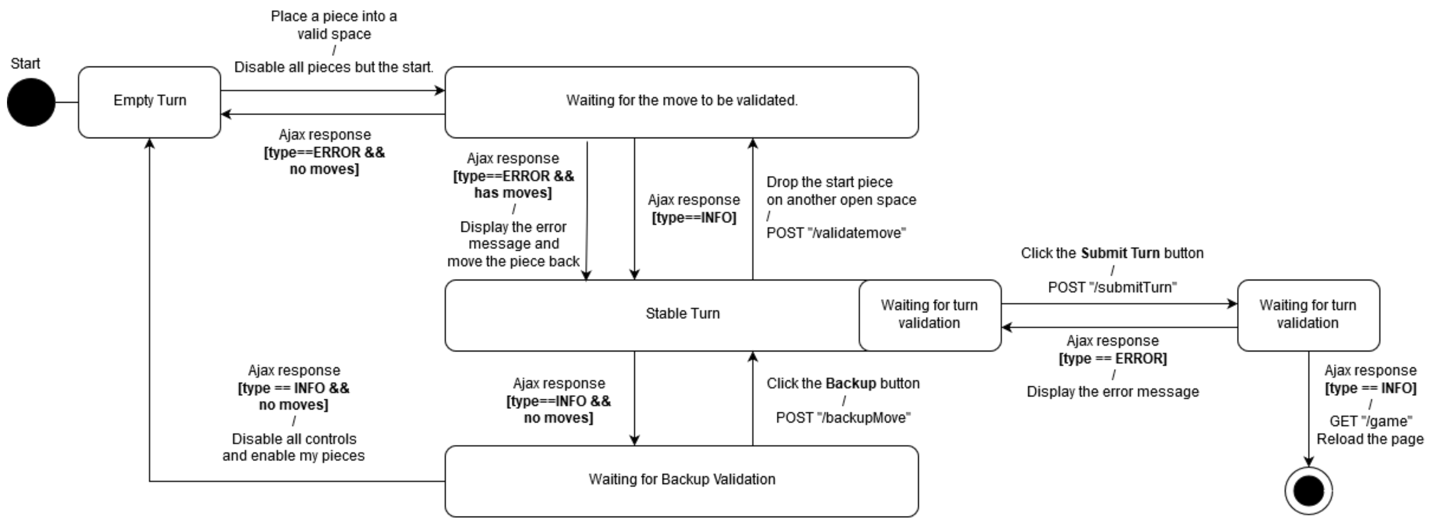
When one player challenges another, the program initializes the game by calling GetGameRoute. This creates a unique game object for the two players, and initializes a model that controls the logic underlying the checkers game. The model sends the game board up to the session. This allows the user interface to render the board, and display it to the players.

If the player decided to challenge a bot instead, the game would be instantiated using the unique



Overview of Client UI Interface:

The user interface for the client end of the Web Checkers application is responsible for allowing the Player to interact with the program. Once a game begins and it is the player's turn, they are able to interact with the board by either dragging and dropping pieces, clicking the Submit button, clicking the Resign button, or backing up a turn.

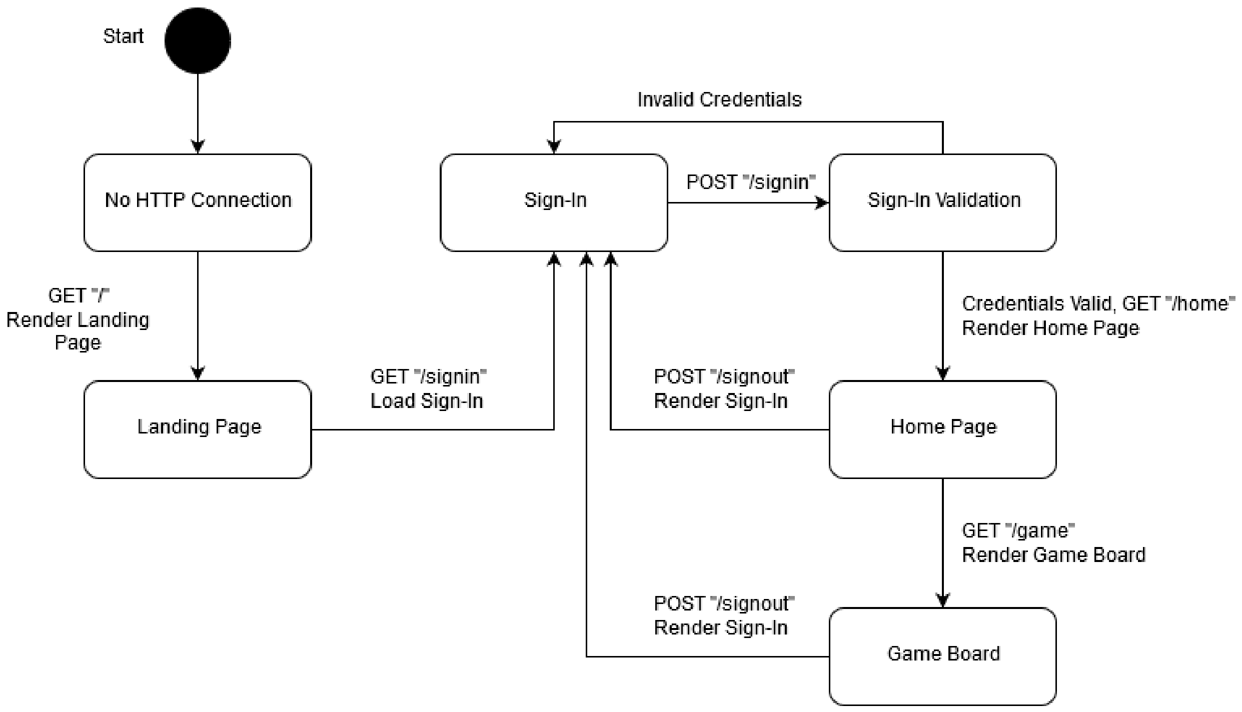

Start
Empty Turn
Place a piece into a valid space / Disable all pieces but the start.
Waiting for the move to be validated.
Ajax response [type==ERROR && no moves]
Ajax response [type==ERROR && has moves] / Display the error message and move the piece back
Ajax response [type==INFO]
Drop the start piece on another open space / POST "/validatemove"
Stable Turn
Waiting for turn validation
Click the Submit Turn button / POST "/submitTurn"
Waiting for turn validation
Ajax response [type == ERROR] / Display the error message
Ajax response [type == INFO] / GET "/game" Reload the page
Ajax response [type == INFO && no moves] / Disable all controls and enable my pieces
Ajax response [type==INFO && no moves]
Click the Backup button / POST "/backupMove"
Waiting for Backup Validation

Overview of Server UI Interface:

Overview of Application Interface:

The web application diagram details how the application segment of the program operates. Through a series of GET and POST commands, the application is able to load different pages of the site based on the user's interaction. When the application layer intercepts one of these commands, it renders the appropriate page of the website.

A player is disconnected from the application when they close their browser window, or redirect from the site hosting the Web Checkers application. When a player chooses to resign from a game they are returned to the home page, where they are able to challenge other players and begin the cycle anew.



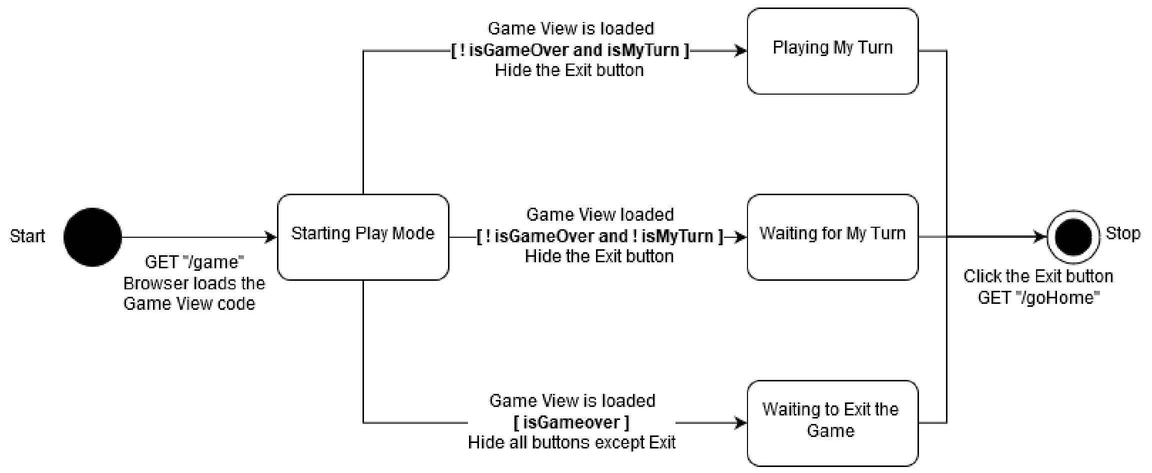
Overview of Model Interface:

Once two players begin a game of checkers, they enter what is known as the ‘turn sequence’. This is a series of states that the players are passed through that coordinate their alternating moves.

When the game begins, the first player (red) is placed into the ‘Playing My Turn’ state. When in this state, they are able to move pieces, resign from the game, and confirm their moves. Once the player has confirmed their move, the updated checkerboard is rendered on their opponent’s screen, and their turn begins.

While the opponent is making their move, the red player is transitioned to the ‘Waiting for my Turn’ state. Every five seconds, their client will ask the server if the opponent has finished their turn. Once they do, the players switch states once again.

When the game had ended, whether it is through a legitimate victory, or another player has resigned, both players transition to the ‘Waiting to Exit the Game’ state, where the Exit button is activated. Once they click the button, they are transitioned to the Home page, where they can begin another game.


Start
GET "/game" Browser loads the Game View code
Starting Play Mode
Game View is loaded [! isGameOver and isMyTurn] Hide the Exit button
Playing My Turn
Game View loaded [! isGameOver and ! isMyTurn] Hide the Exit button
Waiting for My Turn
Game View is loaded [isGameover] Hide all buttons except Exit
Waiting to Exit the Game
Click the Exit button GET "/goHome"
Stop

Testing:

This section will provide information about the testing performed and the results of the testing. Each story is given an “effort score” used to track the estimated complexity of each part of the project, and in turn plan for the next sprint.

Acceptance Testing:

Sprint I: This sprint covered the implementation of the Player Sign-In and Start a Game user stories. Each story is followed by the acceptance criteria said story had to pass before it could be considered complete.

- **Player Sign-In (8):** As a player I want to sign-in so that I can play a game of checkers.
    - Given that I have not yet signed in when I see the Home page then I must see a means to sign-in.
    - Given that I am not signed-in when I do click on the sign-in link then I expect to be taken to the Signin page, with a means to enter a player name.
    - Given that no one else is using my name when I enter my name in the sign-in form and click the Sign-in button then I expect system to reserve my name and navigate back to the Home page.
    - Given that no one else is using my name when I enter a name with a double quote (`"`) then I expect the system to reject this name and return the sign-in form. (Note: double quote is not a valid character in a Player's name.)
    - Given that someone else with my name has signed-in when I enter my name in the sign-in form then I expect the system to reject my sign-in and return the sign-in form. I can then choose a different user name.
    - Given that I have signed in when I see the Home page then I must see a means to sign-out. (such as a link or button) Thus the sign-in and sign-out links need to alternate and never been seen together or not at all.
    - Given that I am signed-in when I click on the sign-out link they I expect the system to sign me out, release my name for use by other users, and then return me to the Home page.
    - Given that I am signed-in when I navigate to the Home page then I expect to see a list of all other signed-in players.

- Given that I am not signed-in when I navigate to the Home page then I expect to see a message of how many players are signed-in, but not a list of them.

- **Start a Game (13):** As a player I want to start a game so that I can play checkers against an opponent.
  - Given that I am signed in when I view the Home page, then I can start a game by selecting a player listen on the Home page.
  - Given that the player I selected is not in a game when I select the player, then the system will begin a checkers game and assign me as the starting (Red) player and my opponent as the White player.
  - Give that the player I selected is already in a game when I select that player, then the system will return me to the Home page with an error message.
  - Given that I am waiting for a game when another player selects a game with me, then the system will transition me from the Home page to the Game View page.
  - Given a valid, initial game board when I drag a piece to a white space then the piece should not be droppable.
  - Given a valid, initial game board when I drag a piece to an occupied space then the piece should not be droppable..
  - Given a valid, initial game board when I drag a piece to an open space then the piece should not be droppable.
  - Given a valid, initial game board when I view the board in the browser then my pieces are oriented on the bottom of the board drid just like I could see the board if I were playing in the physical world.

Sprint II: This sprint covered the implementation of the base Web Checkers game, including stories that manage movement, resigning from a game, and the turn cycle.

- **Validate Move (8):** As a player, I want to be told if my move is valid or not so that I can play the game.
  - Given it is my turn when I move a piece then I expect the application to confirm that it is a valid move.
  - Given that my move is not valid when I confirm it, then I expect to be told that my move is invalid.
  - Given that my move is valid when they confirm it, then I expect it to complete my move.

- **Promote a Piece (1):** As a player, I want my checkers to be ‘kinged’ once they reach the end of the board so that I can move them both forward and back.
    - Given that when a peace makes it to my opponent’s end of the board, when I move the piece into that section of the board then I expect the application to king the piece.
    - Given one of my pieces is kinge, when I move that piece then I expect the piece to be able to make moves both forward and back.

- **Jump a Piece (8):** As a player, I want to be able to jump my opponent’s pieces so that I can capture them and remove them from the game.
    - Given I have an opposing piece diagonal to mine, when I jump it then I expect the opposing piece to be removed from the board.
    - Given I have chosen to make multiple jumps when I submit the move, then I expect it to complete the jumps and capture all pieces in between.

- **Make a Multi-Jump (7):** As a player, I want to be given the opportunity to make a multi-jump when it is possible so that I can capture more than one piece in a single jump.
    - Given there is the opportunity to ‘chain’ multiple jumps together into a single move, when I chose to jump the closes piece then I expect to jump along all the pieces available in the line.
    - Given I have made a jump, and a subsequent jump is available, when I try to submit my turn then it should still count as a valid move.

- **Play a Turn State (8):** As a player, I want to make a single move so that the game can progress.
    - Given it is my turn when I drop a piece into an open space then all other pieces should be disabled except for the start piece.
    - Given I have dropped a piece into an invalid space when the server recognizes that move as invalid then I expect to see an error message.
    - Given I have made a move with the start piece when I drop the start piece on another open space then I expect the turn to validate that move.
    - Given I have made an invalid move when the error message appears then my pieces should be re-enabled, and any piece that had been moved is placed back into its starting location.
    - Given I have made a valid move when I click the Backup button then I expect my most recent move to be undone.
    - Given I click the Backup button when I have made no more moves then all buttons should be disabled and my checkers should be re-enabled.

    - Given I have made a valid turn when I click the Submit button, then I expect the page to display my move.
    - Given I have made an invalid turn when I click the submit button then I expect to be shown an error message referring to the invalid turn.
    - Given I have submitted a valid turn when the server confirms that the turn is valid then I expect the page to reload for both myself and my opponent.

- **Wait for the Player Turn State (5):** As a player, I want to wait while my opponent makes a move so that we can play a game of checkers together.
    - Given I have ended my turn, when my opponent begins their turn then I expect to have to wait until the other player had finished their turn before my own can begin.
    - Given my opponent has finished their turn, when my turn begins then I expect to see an updated game board.
    - Given I am waiting for my opponent to finish their turn, when I resign then I expect the game to end.

- **Resign from a Game (8):** As a player I want to forfeit a game so that I can end a round of checkers prematurely.
    - Given it is my turn when I have not made any moves then the Resign button should be enabled.
    - Given the Resign button is enabled when I click it then I expect the game to end and my opponent to be awarded the victory.
    - Given it is my turn when I make a valid move then the Resign button should be disabled.

Sprint III: This sprint covered the implementation of the enhancements that were to be added to the Web Checkers game.

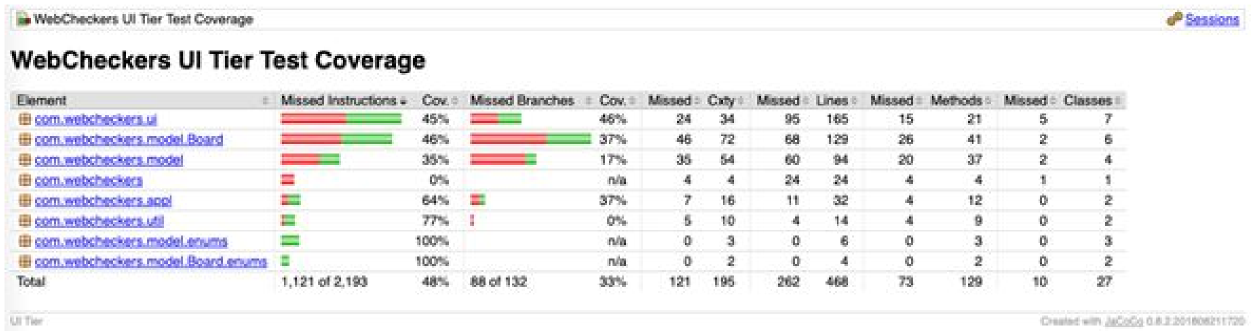
- **Build AI Player (13):** As a player I want to be able to challenge an artificial intelligence so that I can play a game without having to challenge another living person.
    - Given that I select a game with an AI player, when the game begins then I expect to be placed into a game with an AI.
    - Given I am placed into a game with an artificial player, when it is the AI's turn then I expect them to make a legal move.
- **View a Replay (8):** As a player, I want to be able to view replays of my most recent game so that I can review my gameplay.
    - Given that I have logged into an account, when I look at the display then I expect to see a place where the replay of the most recent game is stored.

    - Given I have played a game of checkers when I navigate back to the home tab then I expect to see a place where the most recent game I have played is stored.
    - Given I play a game when a recording is already stored, when I view the record of the last game I played then I expect it to have overwritten the previous game.

Unit Testing and Code Coverage:

Unit testing is a design technique that allows developers to test individual sections of the code on their own without having to worry about other classes they may depend on. This means that, in the event something goes wrong with a portion of the project, that specific piece can be examined on its own. Code coverage reports let developers know how well the unit tests represent the component they were designed for.

UI Tier: The coverage report for the UI Tier of the Web Checkers application


WebCheckers UI Tier Test Coverage

Sessions

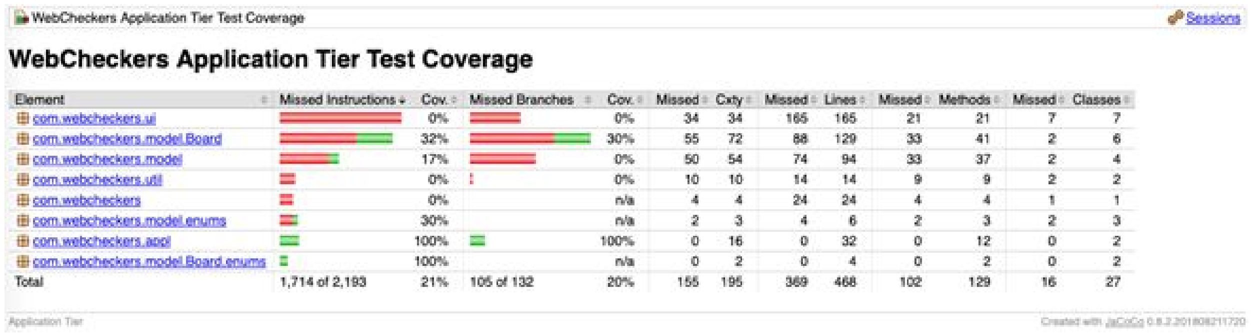
WebCheckers UI Tier Test Coverage

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.webcheckers.ui		45%		46%	24	34	95	165	15	21	5	7
com.webcheckers.model.Board		46%		37%	46	72	68	129	26	41	2	6
com.webcheckers.model		35%		17%	35	54	60	94	20	37	2	4
com.webcheckers		0%		n/a	4	4	24	24	4	4	1	1
com.webcheckers.appl		64%		37%	7	16	11	32	4	12	0	2
com.webcheckers.util		77%		0%	5	10	4	14	4	9	0	2
com.webcheckers.model.enums		100%		n/a	0	3	0	6	0	3	0	3
com.webcheckers.model.Board.enums		100%		n/a	0	2	0	4	0	2	0	2
Total	1,121 of 2,193	48%	88 of 132	33%	121	195	262	468	73	129	10	27

UI Tier

Created with JaCoCo 0.8.2.201808211720

Application Tier: The coverage report for the Application Tier of the Web Checkers application


WebCheckers Application Tier Test Coverage

Sessions

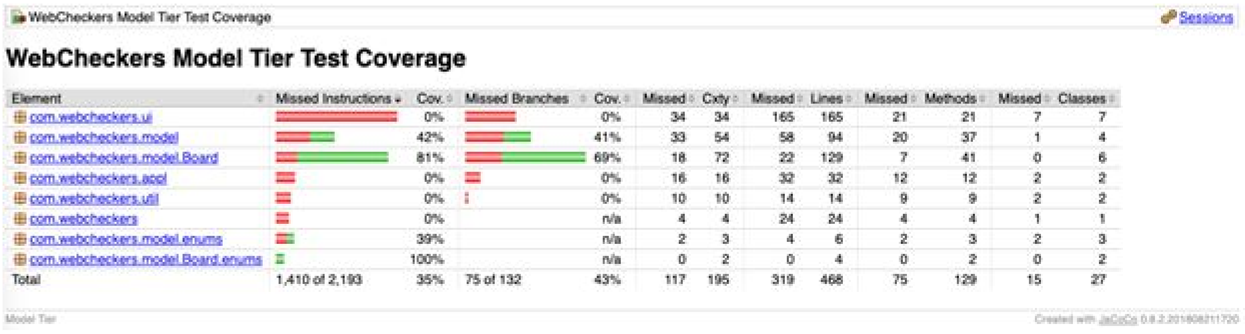
WebCheckers Application Tier Test Coverage

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.webcheckers.ui		0%		0%	34	34	165	165	21	21	7	7
com.webcheckers.model.Board		32%		30%	55	72	88	129	33	41	2	6
com.webcheckers.model		17%		0%	50	54	74	94	33	37	2	4
com.webcheckers.util		0%		0%	10	10	14	14	9	9	2	2
com.webcheckers		0%		n/a	4	4	24	24	4	4	1	1
com.webcheckers.model.enums		30%		n/a	2	3	4	6	2	3	2	3
com.webcheckers.appl		100%		100%	0	16	0	32	0	12	0	2
com.webcheckers.model.Board.enums		100%		n/a	0	2	0	4	0	2	0	2
Total	1,714 of 2,193	21%	105 of 132	20%	155	195	369	468	102	129	16	27

Application Tier

Created with JaCoCo 0.8.2.201808211720

Model Tier: The coverage report for the Model Tier of the Web Checkers application



WebCheckers Model Tier Test Coverage　Sessions

WebCheckers Model Tier Test Coverage

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.webcheckers.ui		0%		0%	34	34	165	165	21	21	7	7
com.webcheckers.model		42%		41%	33	54	58	94	20	37	1	4
com.webcheckers.model.Board		81%		69%	18	72	22	129	7	41	0	6
com.webcheckers.appl		0%		0%	16	16	32	32	12	12	2	2
com.webcheckers.util		0%		0%	10	10	14	14	9	9	2	2
com.webcheckers		0%		n/a	4	4	24	24	4	4	1	1
com.webcheckers.model.enums		39%		n/a	2	3	4	6	2	3	2	3
com.webcheckers.model.Board.enums		100%		n/a	0	2	0	4	0	2	0	2
Total	1,410 of 2,193	35%	75 of 132	43%	117	195	319	468	75	129	15	27

Model Tier　Created with JaCoCo 0.8.2.201808211720

Code Metrics:

Not required until Sprint 4

[

Insert a description of what Code Metrics were completed and INSERT the results of your Code Metrics AND an analysis of the results.

]

Design Quality and Possible Revisions:

Not required until Sprint 4

[

Insert a summary of the application highlighting what your team feel are the highlights. Also, include a description of possible revisions or inclusions that the team would like to have seen if they were to continue updating this application.

]

THREE IMPROVEMENTS IN PICTURES