DesignDoc.md

geometry: margin=1in

PROJECT Design Documentation

Team Information

- Team name: 2C
- Team members:
  - Julio Cuello
  - Alanna Luce
  - Benson Yan
  - Joshua Shaffer
  - Alex Iacob

Executive Summary

A web checkers application is being created. This implementation contains all of the rules and methodology to run a game of checkers.

Purpose

This is a web checkers application that uses freemarker and Maven to create a playerLobby, gameCenter and the respective routes to handle the users signing in, creating games and playing checkers online

Glossary and Acronyms

Term	Definition
VO	Value Object
MVP	Minimum Viable Product
UI	User Interface

Requirements

This section describes the features of the application.

Definition of MVP

The MVP contains all of the basic rules of a classic game of checkers. This implies:

- Having a 8x8 checkered board with 12 red and 12 white pieces.
- Being able to capture any piece given there is space.
- Being able to king a checker piece once you get a piece in your opponent's closest row.
- Being able to capture multiple pieces in one single turn with the same checker piece.
- Being able to win the game via captures or opponent resignation.

MVP Features

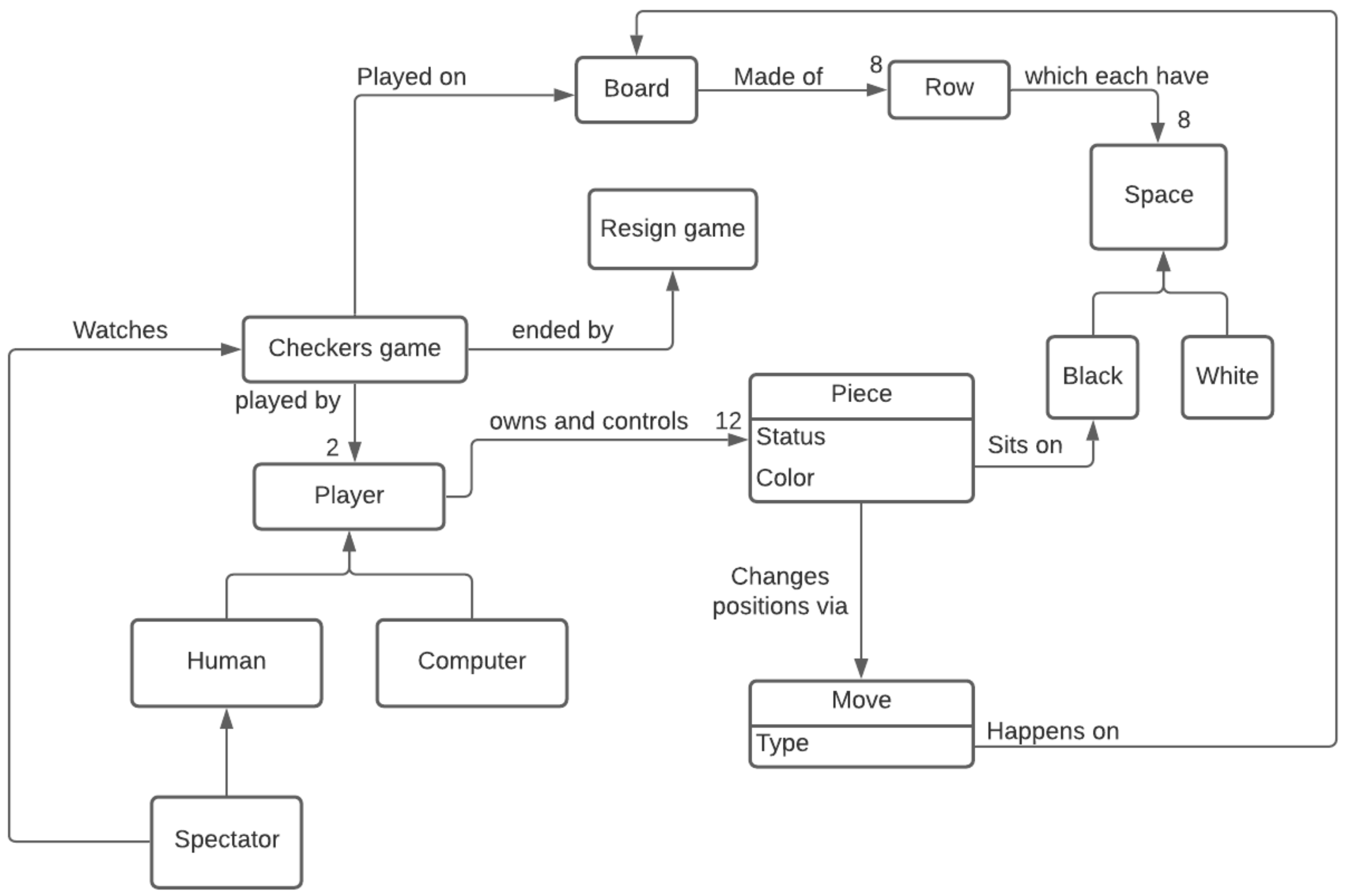
- Pre-game
  - Create a board
  - Player Sign-in
  - Player Sign-Out
- Player Movement
  - Single Move
  - Single Jump
  - Multiple Jumps
  - King a Piece
- End a Game
  - Resign from Game
  - Win a Game

Road-map of Enhancements

- Play against the AI user that allows anyone to start a game with a CPU.
- Spectator feature that allows others to view an ongoing game of checkers.

Domain Model

This section describes the application domain.



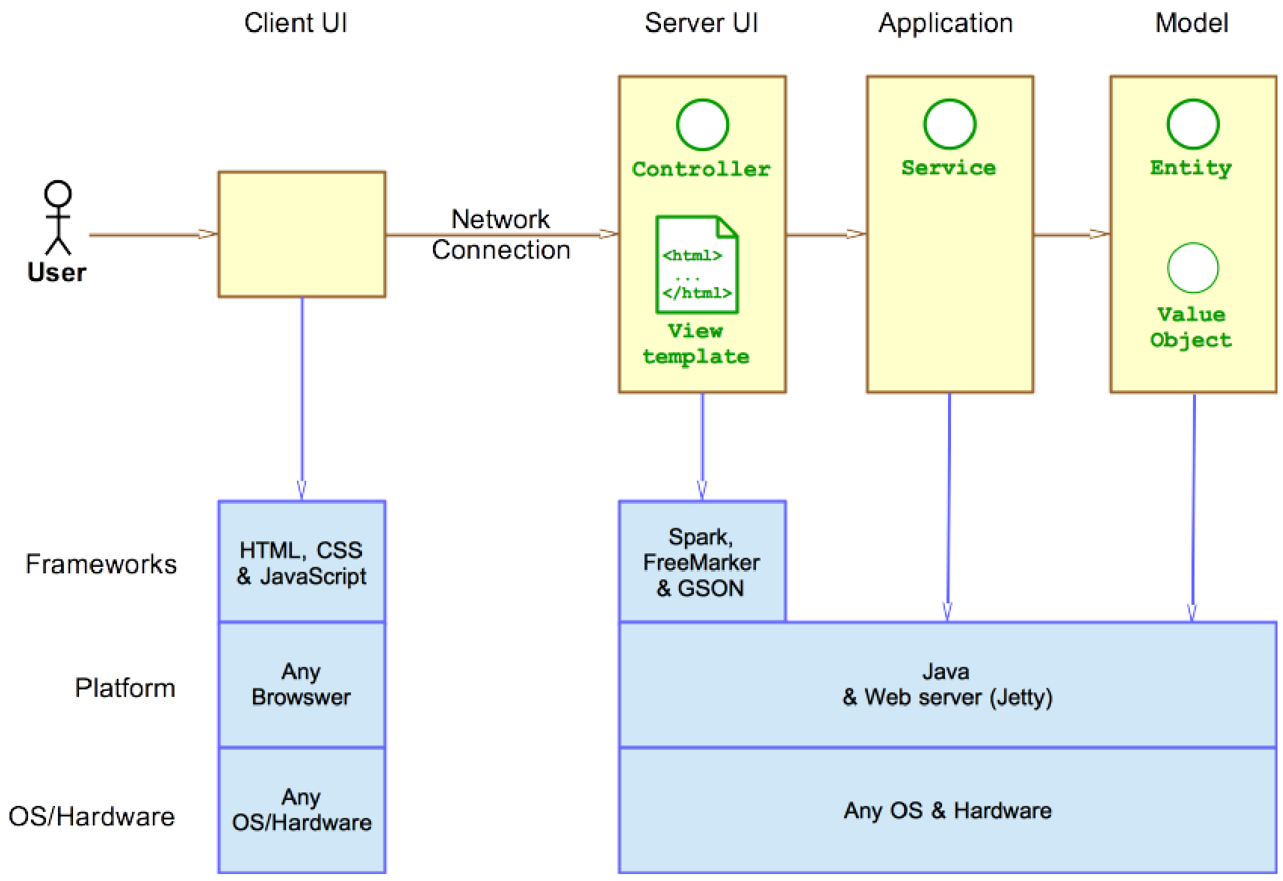
The domain shows a checkers game which is played by two players or one player and an AI user on a 8x8 checkered board. Each player has 12 checker pieces; one player has red pieces and the other white. Each piece is placed on a dark square and can perform a variety of moves. These include, a single move, a single jump, and a multi jump. A spectator can also join to watch an ongoing match of checkers. This spectator can stop watching at any time and has no power over what happens on the 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.



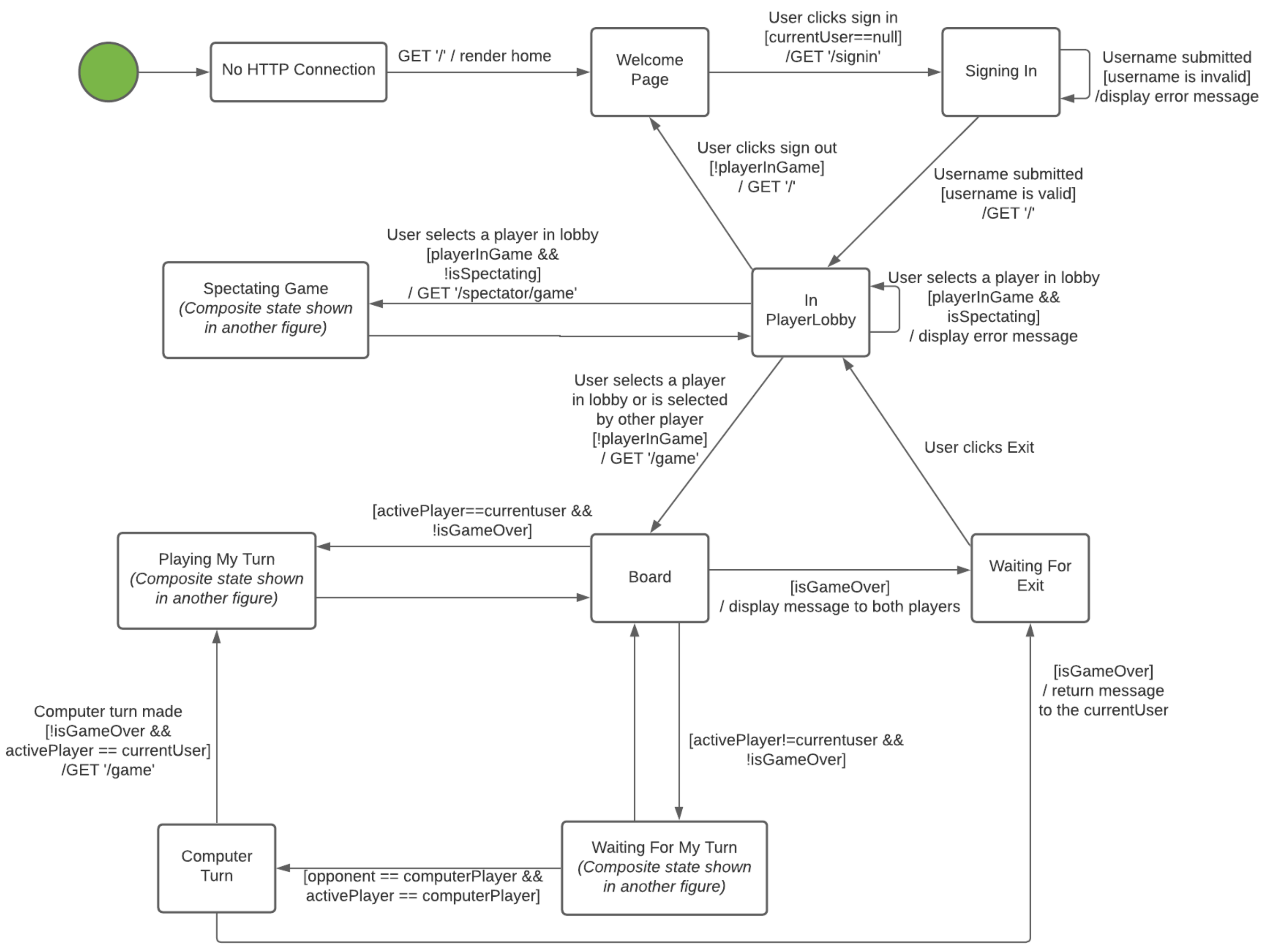
As a web application, the user interacts with the system using a browser. The client-side of the UI is composed of HTML pages with some minimal CSS for styling the page. There is also some JavaScript that has been provided to the team by the architect.

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

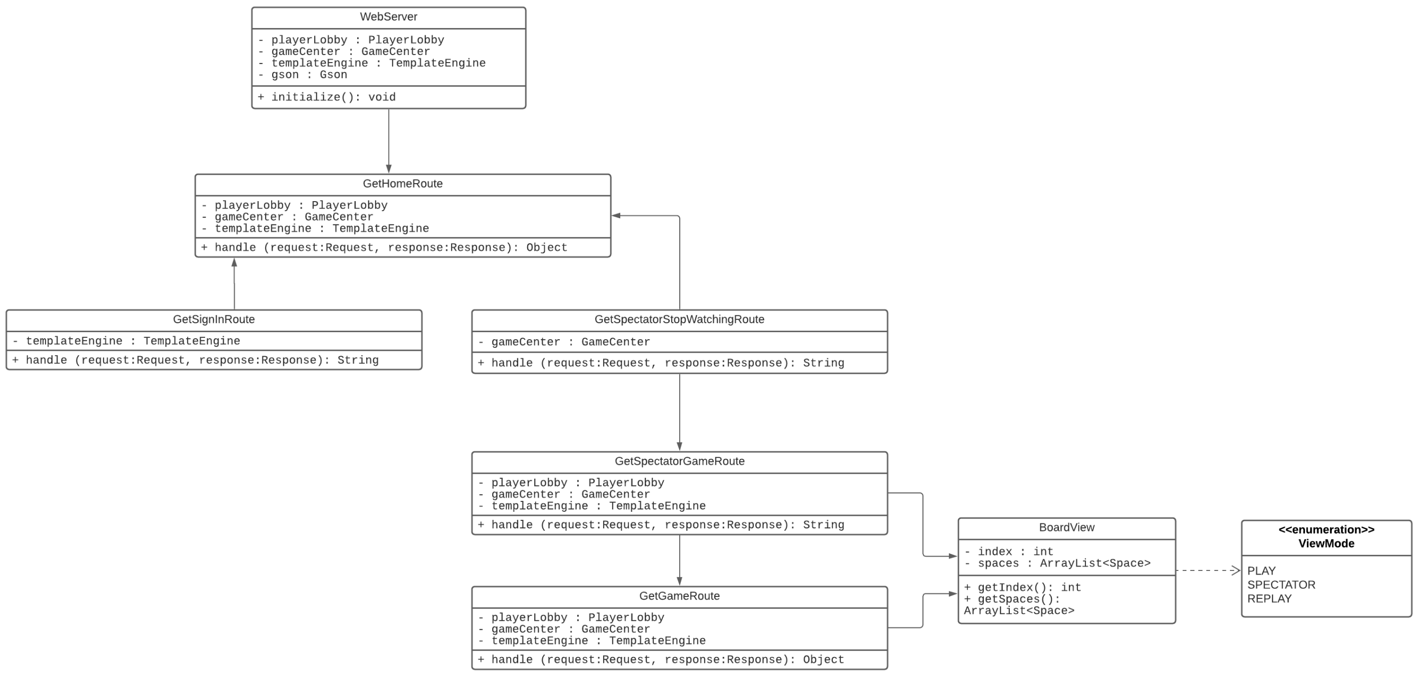
This section describes the web interface flow; this is how the user views and interacts with the WebCheckers application.


No HTTP Connection
GET '/' / render home
Welcome Page
User clicks sign in [currentUser==null] /GET '/signin'
Signing In
Username submitted [username is invalid] /display error message
User clicks sign out [!playerInGame] / GET '/'
Username submitted [username is valid] /GET '/'
User selects a player in lobby [playerInGame && !isSpectating] / GET '/spectator/game'
Spectating Game (Composite state shown in another figure)
In PlayerLobby
User selects a player in lobby [playerInGame && isSpectating] / display error message
User selects a player in lobby or is selected by other player [!playerInGame] / GET '/game'
User clicks Exit
[activePlayer==currentuser && !isGameOver]
Playing My Turn (Composite state shown in another figure)
Board
[isGameOver] / display message to both players
Waiting For Exit
[isGameOver] / return message to the currentUser
Computer turn made [!isGameOver && activePlayer == currentUser] /GET '/game'
[activePlayer!=currentuser && !isGameOver]
Computer Turn
[opponent == computerPlayer && activePlayer == computerPlayer]
Waiting For My Turn (Composite state shown in another figure)

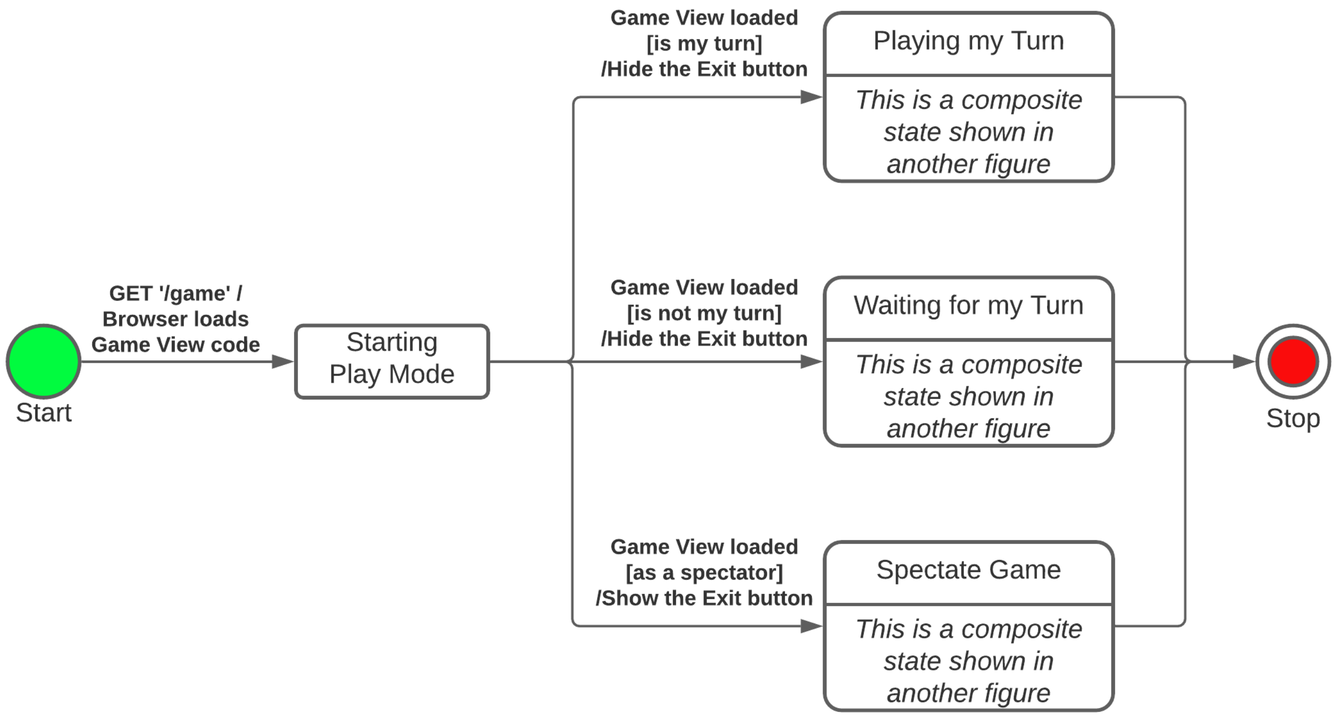
When the user is connected to the checkers application, they are displayed the home page where the number of players online and a hyperlink to redirect to the sign-in page is shown. When the user attempts to sign-in to the application with an invalid username, they will stay on the sign-in page. If a valid username is used, they will redirect to the home page which now contains information about the other users that are online. The user now is able to challenge other players who currently are not in a game. When the user attempts to challenge someone that is in a game, they will begin spectating the game that user is playing. When the user attempts to challenge someone that is spectating a game, they will be given an error message. When the challenged player is not in a game, both players will be redirected to a Board and begin a new game.

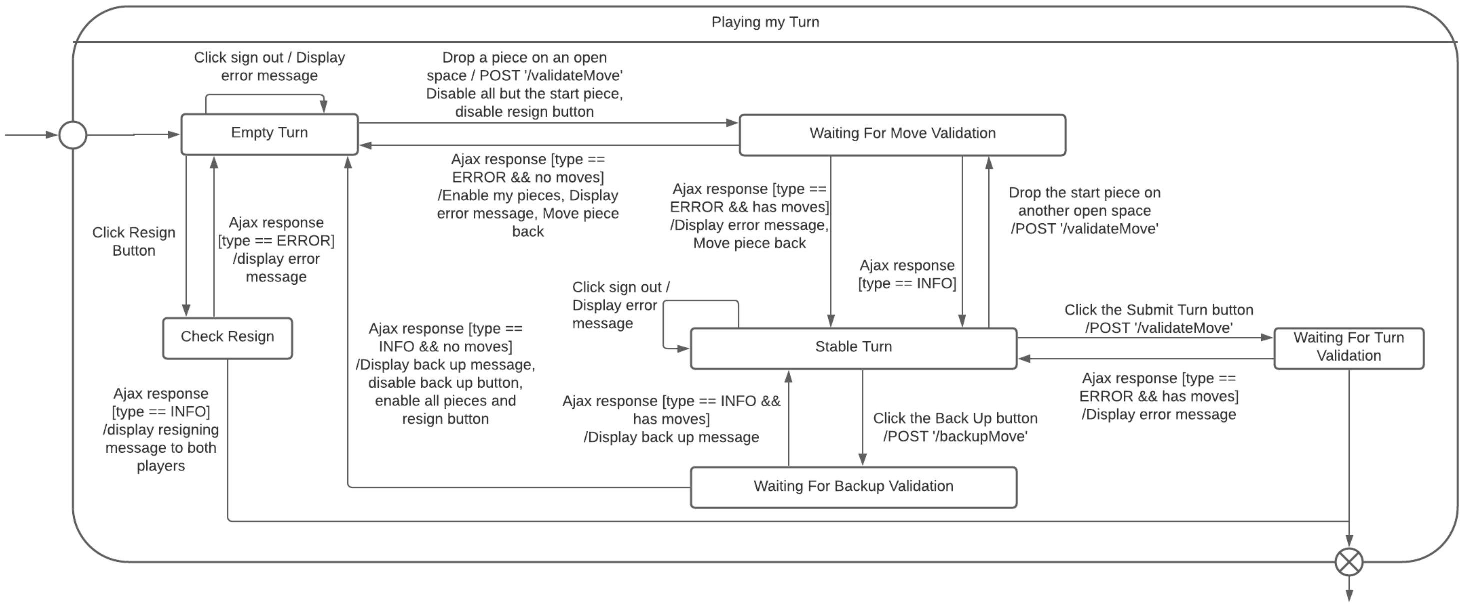
UI Tier

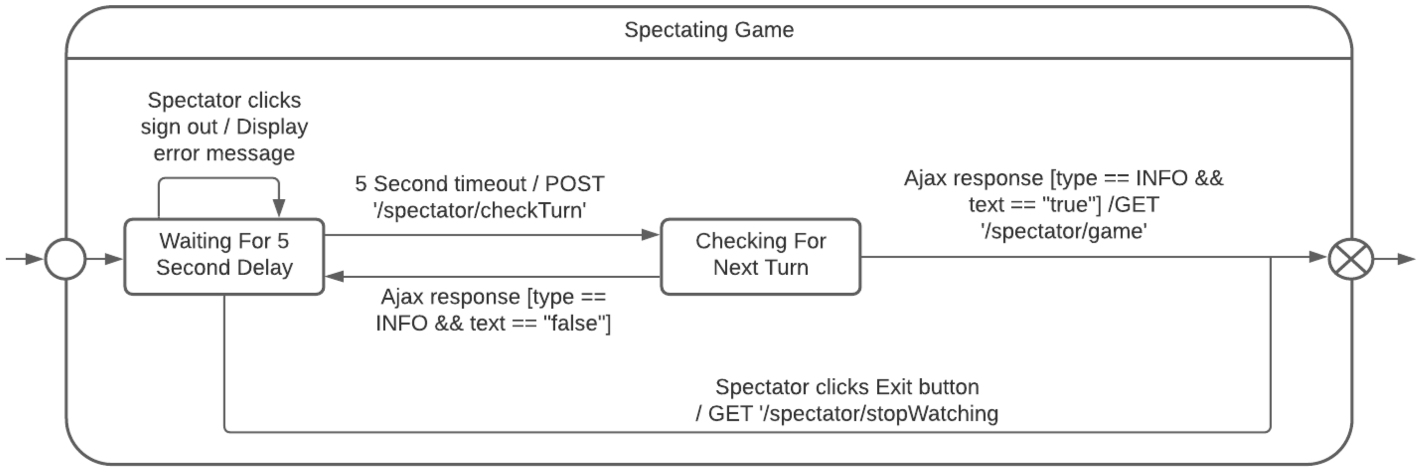
The UI Tier mainly consists of all of the 'GET' Routes which are responsible for creating the view-model and displaying a page for the user to see. When the user initially connects, GetHomeRoute renders a home screen for the user to navigate through. Once the user(s) sign in via a sign-in screen rendered by GetSignInRoute, they are directed to a page where they can choose whether they would want to play a game or spectate an ongoing game. If they choose to start a game, GetGameRoute renders a playable game for two players and GetSpectatorGameRoute renders an ongoing game. The user's choice is then sent to other routes in the form of a ViewMode, which states whether that user is a player or spectator. When the spectator decides to stop watching, they can exit, and GetSpectatorStopWatchingRoute redirects them back to the home page. As for the other classes in this tier, WebServer allows the game to function the way it is meant to. Any time a player makes a move, that information is sent to BoardView, which is then sent to the model tier to display to the user.

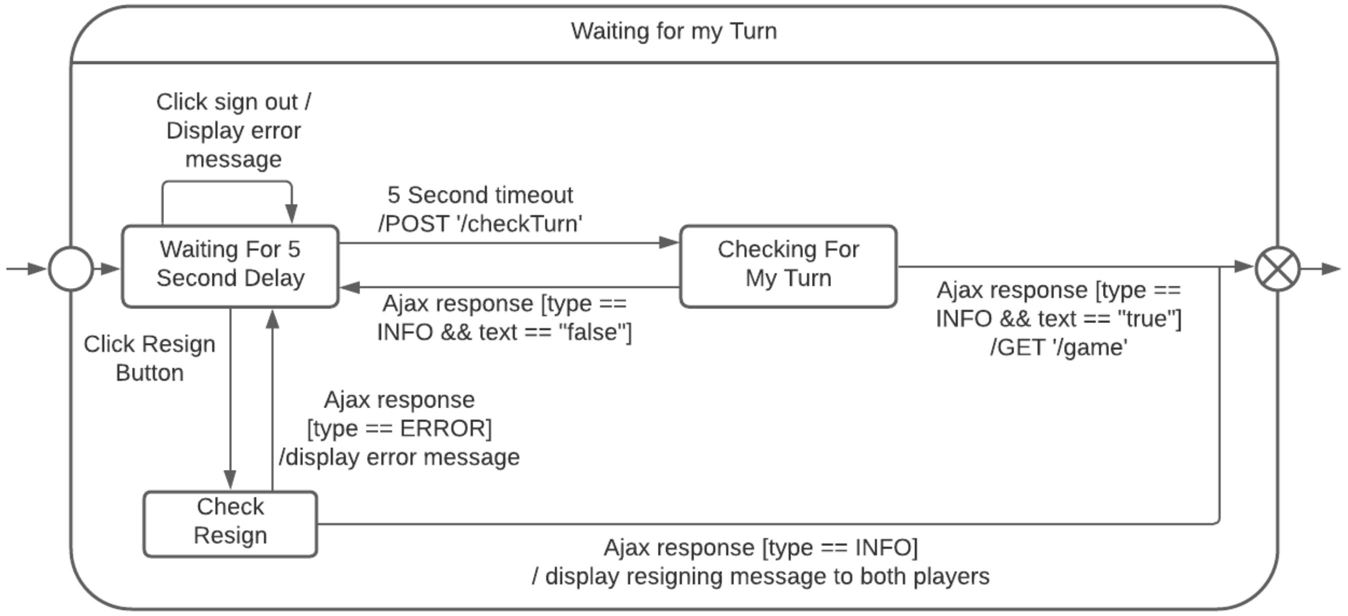

WebServer
- playerLobby : PlayerLobby
- gameCenter : GameCenter
- templateEngine : TemplateEngine
- gson : Gson
+ initialize(): void
GetHomeRoute
- playerLobby : PlayerLobby
- gameCenter : GameCenter
- templateEngine : TemplateEngine
+ handle (request:Request, response:Response): Object
GetSignInRoute
- templateEngine : TemplateEngine
+ handle (request:Request, response:Response): String
GetSpectatorStopWatchingRoute
- gameCenter : GameCenter
+ handle (request:Request, response:Response): String
GetSpectatorGameRoute
- playerLobby : PlayerLobby
- gameCenter : GameCenter
- templateEngine : TemplateEngine
+ handle (request:Request, response:Response): String
GetGameRoute
- playerLobby : PlayerLobby
- gameCenter : GameCenter
- templateEngine : TemplateEngine
+ handle (request:Request, response:Response): Object
BoardView
- index : int
- spaces : ArrayList<Space>
+ getIndex(): int
+ getSpaces(): ArrayList<Space>
<<enumeration>>
ViewMode
PLAY
SPECTATOR
REPLAY

The WebCheckers In-Game State Diagram


Start
GET '/game' / Browser loads Game View code
Starting Play Mode
Game View loaded [is my turn] /Hide the Exit button
Playing my Turn
This is a composite state shown in another figure
Game View loaded [is not my turn] /Hide the Exit button
Waiting for my Turn
This is a composite state shown in another figure
Game View loaded [as a spectator] /Show the Exit button
Spectate Game
This is a composite state shown in another figure
Stop

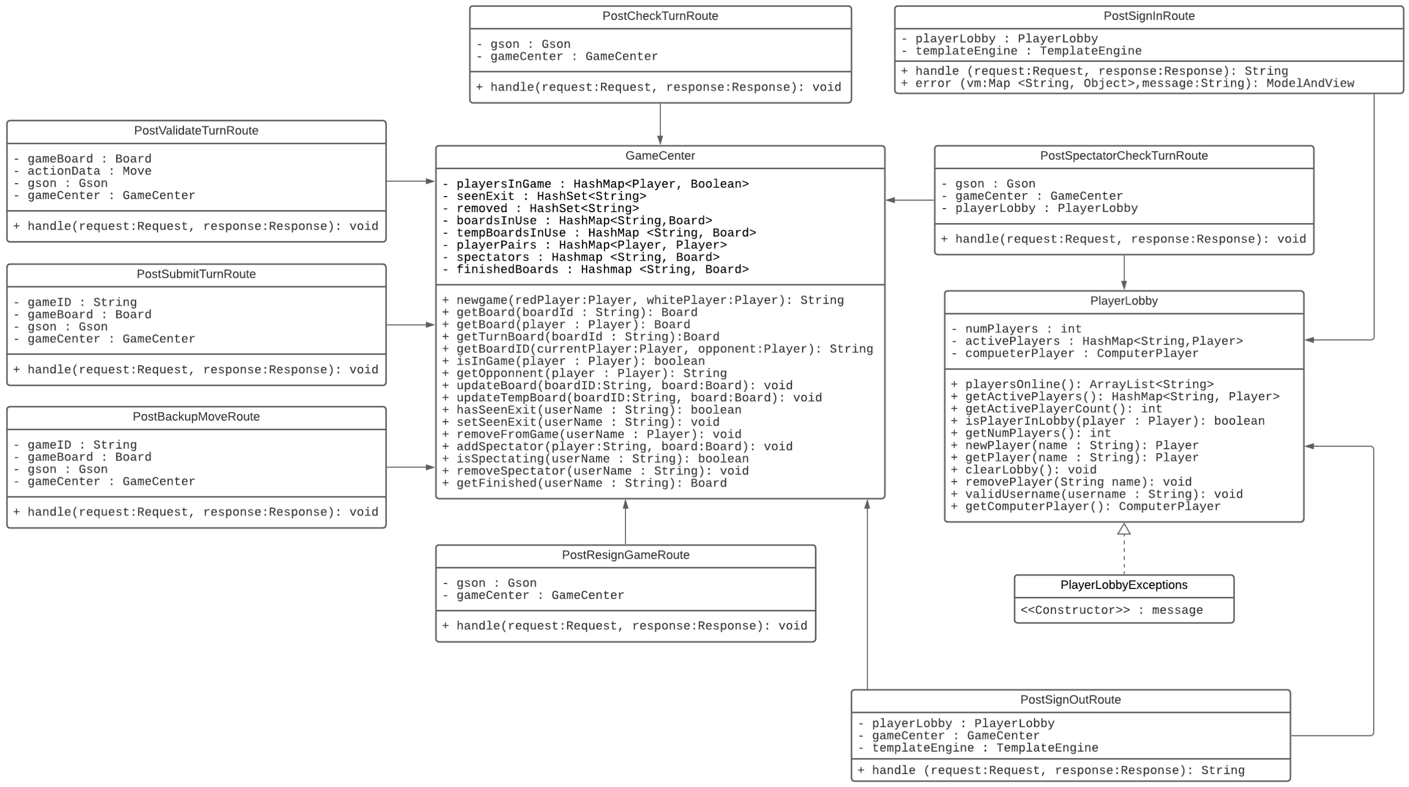

Playing my Turn
Click sign out / Display error message
Empty Turn
Drop a piece on an open space / POST '/validateMove' Disable all but the start piece, disable resign button
Waiting For Move Validation
Ajax response [type == ERROR && no moves] /Enable my pieces, Display error message, Move piece back
Ajax response [type == ERROR && has moves] /Display error message, Move piece back
Drop the start piece on another open space /POST '/validateMove'
Click Resign Button
Ajax response [type == ERROR] /display error message
Ajax response [type == INFO]
Check Resign
Click sign out / Display error message
Stable Turn
Click the Submit Turn button /POST '/validateMove'
Waiting For Turn Validation
Ajax response [type == ERROR && has moves] /Display error message
Ajax response [type == INFO && no moves] /Display back up message, disable back up button, enable all pieces and resign button
Ajax response [type == INFO && has moves] /Display back up message
Click the Back Up button /POST '/backupMove'
Ajax response [type == INFO] /display resigning message to both players
Waiting For Backup Validation


Spectating Game
Spectator clicks sign out / Display error message
5 Second timeout / POST '/spectator/checkTurn'
Ajax response [type == INFO && text == "true"] /GET '/spectator/game'
Waiting For 5 Second Delay
Checking For Next Turn
Ajax response [type == INFO && text == "false"]
Spectator clicks Exit button / GET '/spectator/stopWatching'

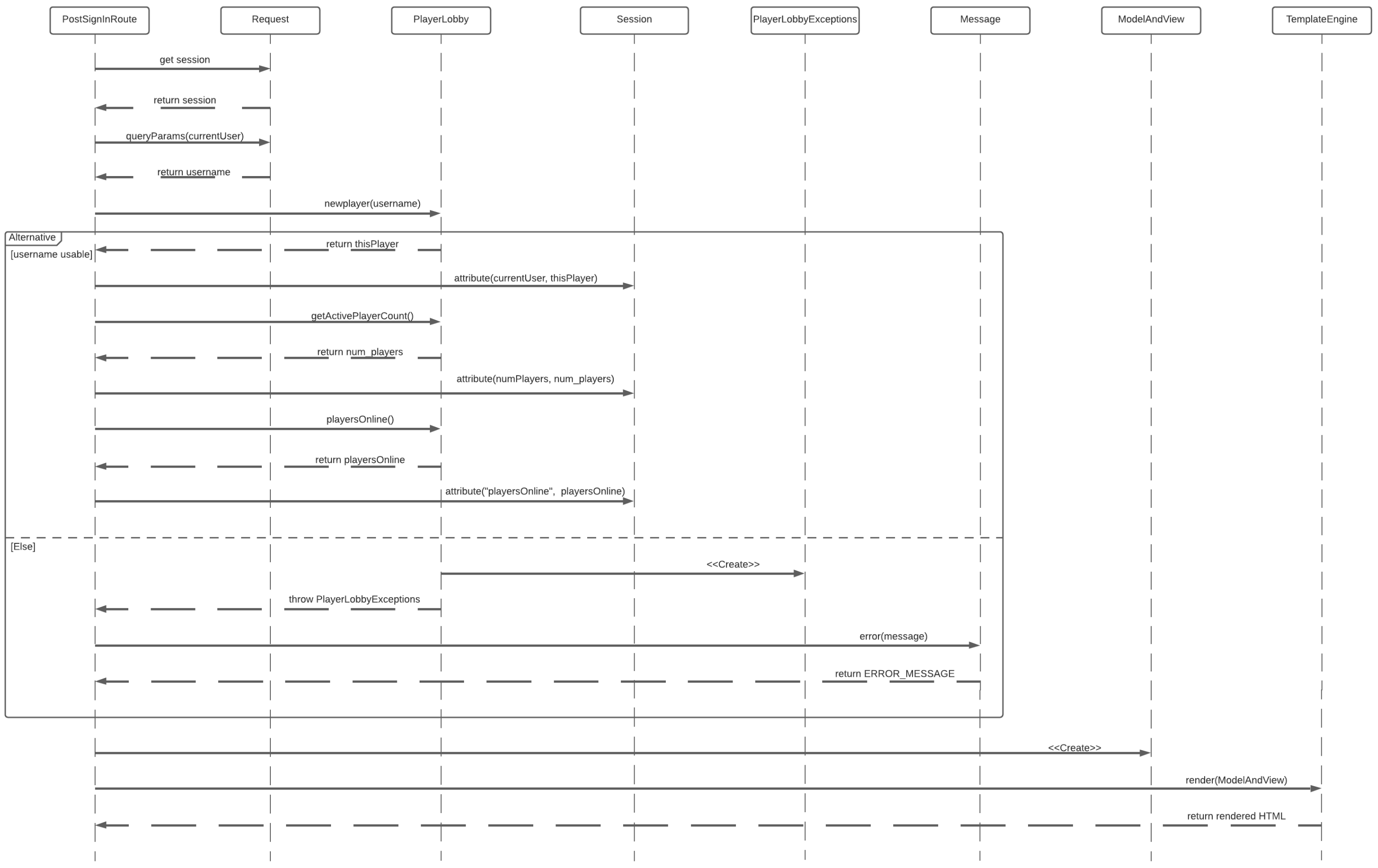

Waiting for my Turn
Click sign out / Display error message
5 Second timeout /POST '/checkTurn'
Waiting For 5 Second Delay
Checking For My Turn
Ajax response [type == INFO && text == "false"]
Ajax response [type == INFO && text == "true"] /GET '/game'
Click Resign Button
Ajax response [type == ERROR] /display error message
Check Resign
Ajax response [type == INFO] / display resigning message to both players

Application Tier

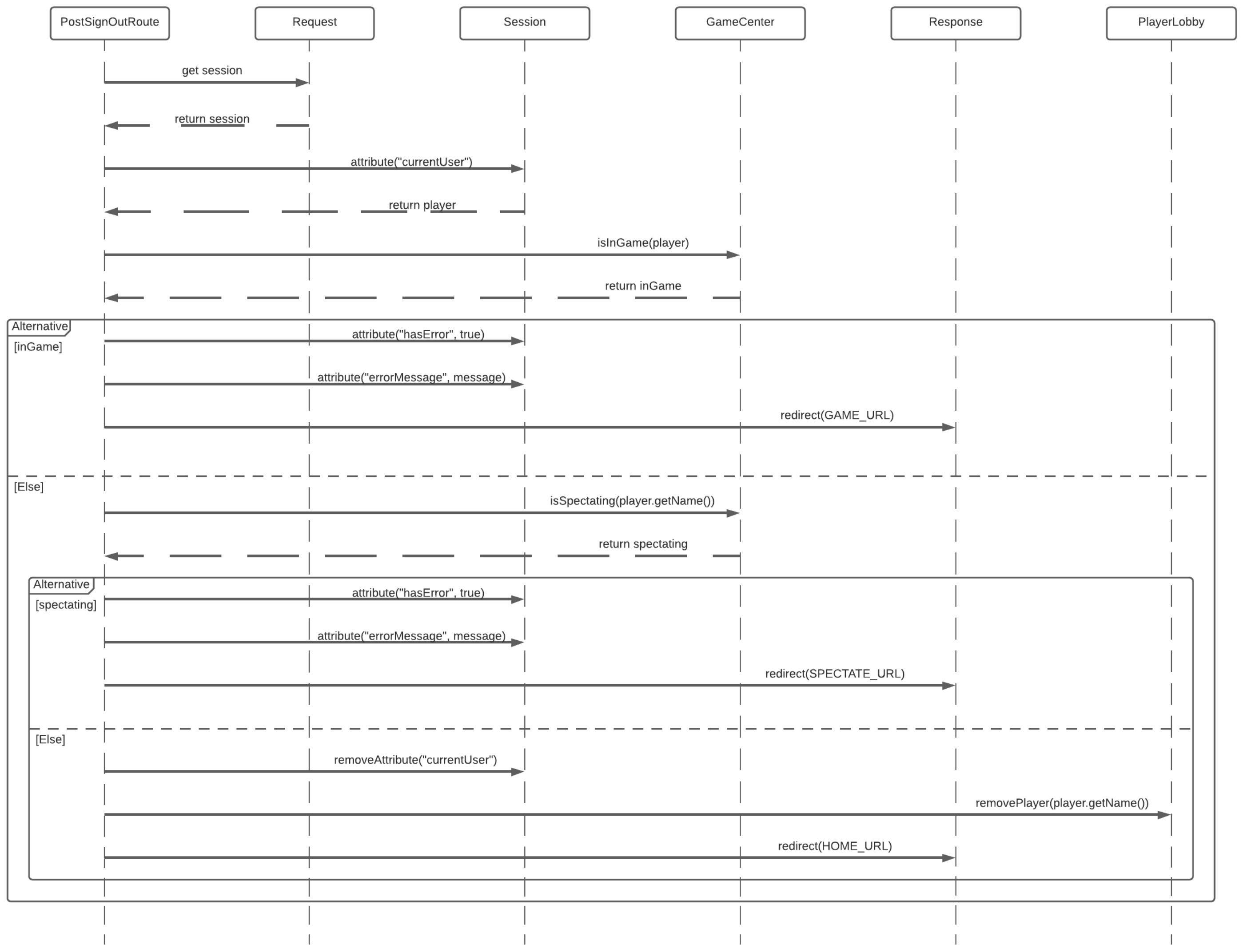
The Application Tier mainly consists of all the ‘POST’ Routes which are also responsible for the business logic of the application. The Application Tier begins with an instance of class PlayerLobby which contains the information about players signed in to the application. Through this class, the PostSignInRoute is able to check if the sign-in attempt is valid. Each time a player signs in with a valid username, they are added to the list of online players and can begin playing games. When the users in PlayerLobby choose to sign out, PostSignOutRoute will remove their username from the list in PlayerLobby which clears it up for use by a different user. The GameCenter class keeps track of the information related to active games, including when a new game is created. As such, the routes that relate to the playing of a game, such as submitting a turn, backing up, or resigning are dependent on this class. The PostSignOut route is also dependant on the GameCenter as it needs to know that a user is not in a game when they are trying to sign out. The PostSpectatorCheckTurnRoute is also dependent on both the PlayerLobby and the GameCenter as the spectator needs knowledge of the player that they are spectating and the game that player is playing.


PostCheckTurnRoute
- gson : Gson
- gameCenter : GameCenter
+ handle(request:Request, response:Response): void
PostSignInRoute
- playerLobby : PlayerLobby
- templateEngine : TemplateEngine
+ handle (request:Request, response:Response): String
+ error (vm:Map <String, Object>,message:String): ModelAndView
PostValidateTurnRoute
- gameBoard : Board
- actionData : Move
- gson : Gson
- gameCenter : GameCenter
+ handle(request:Request, response:Response): void
GameCenter
- playersInGame : HashMap<Player, Boolean>
- seenExit : HashSet<String>
- removed : HashSet<String>
- boardsInUse : HashMap<String,Board>
- tempBoardsInUse : HashMap <String, Board>
- playerPairs : HashMap<Player, Player>
- spectators : Hashmap <String, Board>
- finishedBoards : Hashmap <String, Board>
+ newgame(redPlayer:Player, whitePlayer:Player): String
+ getBoard(boardId : String): Board
+ getBoard(player : Player): Board
+ getTurnBoard(boardId : String):Board
+ getBoardID(currentPlayer:Player, opponent:Player): String
+ isInGame(player : Player): boolean
+ getOpponnent(player : Player): String
+ updateBoard(boardID:String, board:Board): void
+ updateTempBoard(boardID:String, board:Board): void
+ hasSeenExit(userName : String): boolean
+ setSeenExit(userName : String): void
+ removeFromGame(userName : Player): void
+ addSpectator(player:String, board:Board): void
+ isSpectating(userName : String): boolean
+ removeSpectator(userName : String): void
+ getFinished(userName : String): Board
PostSpectatorCheckTurnRoute
- gson : Gson
- gameCenter : GameCenter
- playerLobby : PlayerLobby
+ handle(request:Request, response:Response): void
PostSubmitTurnRoute
- gameID : String
- gameBoard : Board
- gson : Gson
- gameCenter : GameCenter
+ handle(request:Request, response:Response): void
PlayerLobby
- numPlayers : int
- activePlayers : HashMap<String,Player>
- compueterPlayer : ComputerPlayer
+ playersOnline(): ArrayList<String>
+ getActivePlayers(): HashMap<String, Player>
+ getActivePlayerCount(): int
+ isPlayerInLobby(player : Player): boolean
+ getNumPlayers(): int
+ newPlayer(name : String): Player
+ getPlayer(name : String): Player
+ clearLobby(): void
+ removePlayer(String name): void
+ validUsername(username : String): void
+ getComputerPlayer(): ComputerPlayer
PostBackupMoveRoute
- gameID : String
- gameBoard : Board
- gson : Gson
- gameCenter : GameCenter
+ handle(request:Request, response:Response): void
PostResignGameRoute
- gson : Gson
- gameCenter : GameCenter
+ handle(request:Request, response:Response): void
PlayerLobbyExceptions
<<Constructor>> : message
PostSignOutRoute
- playerLobby : PlayerLobby
- gameCenter : GameCenter
- templateEngine : TemplateEngine
+ handle (request:Request, response:Response): String

The WebCheckers Sign In Sequence Diagram

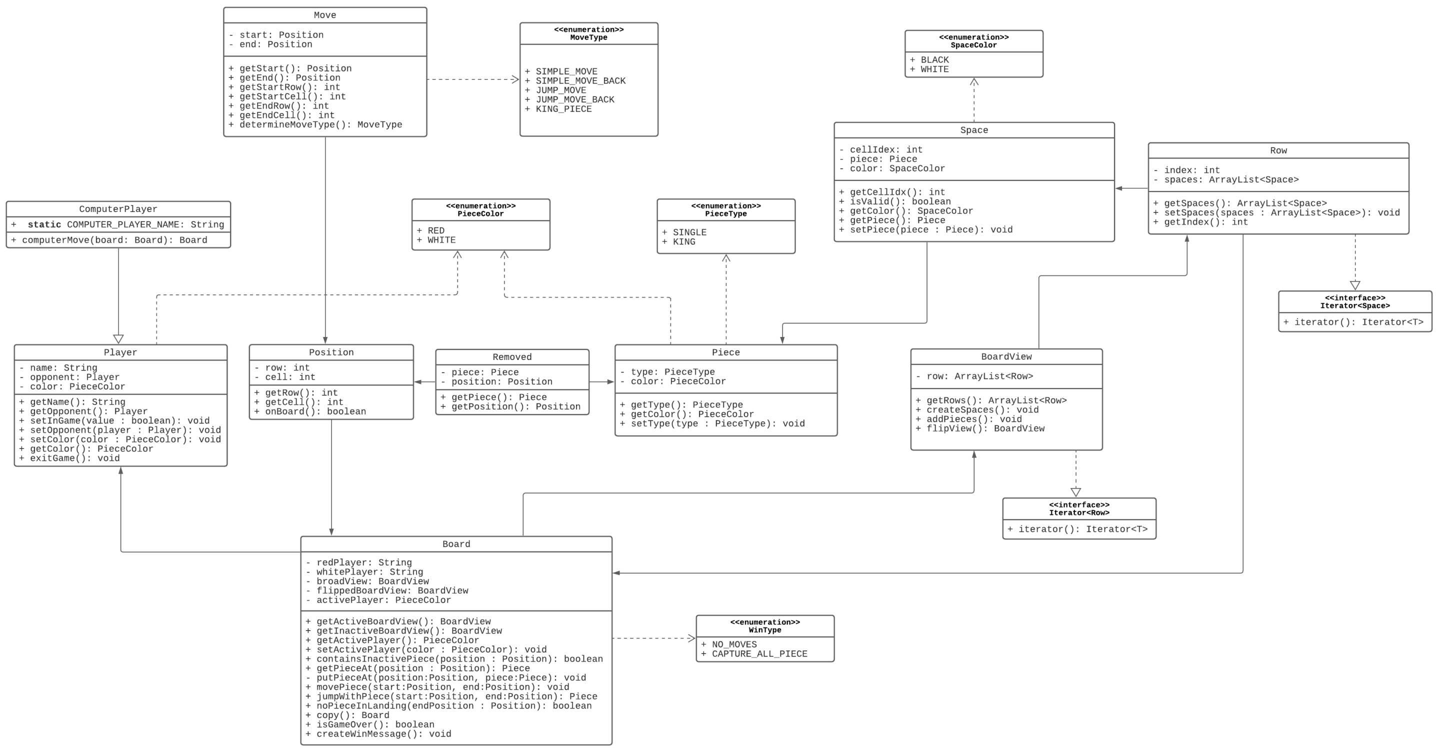


The WebCheckers Sign Out Sequence Diagram



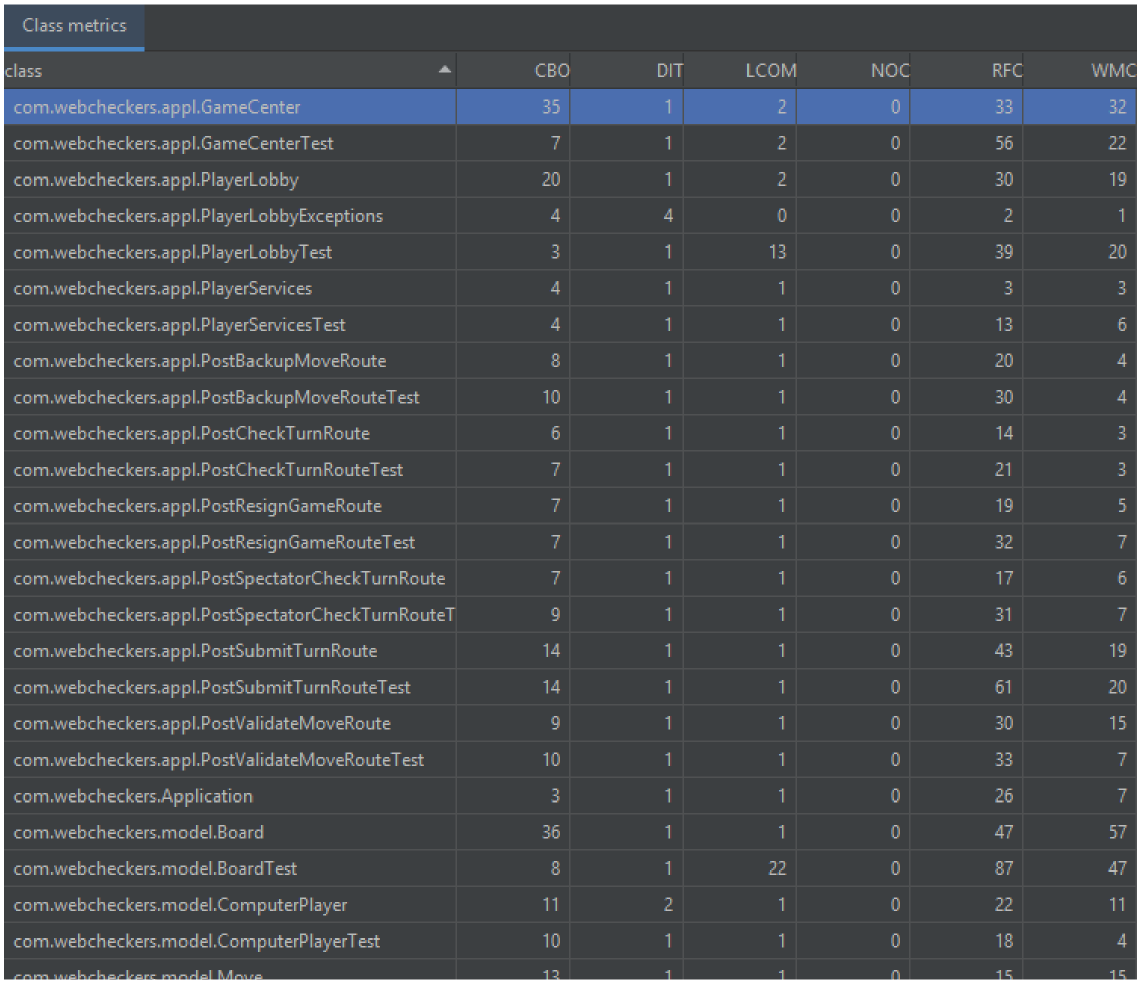
Model Tier

The model tier is in charge of everything related to the logic and the rules of the game and it makes sure that everything works correctly in terms of what can and can't be done in the game. We start off with the board class that takes care of everything that is in the board and moving the components in it. There are several types of moves that the players can make in the board either a jump move, a simple move or a multiple jump move. The board has pieces which can be of different colors, either red or white, and these can be updated to new positions, or even different types of piece like the king piece once they reach the end of the board from their perspective. A removed object is used for a jump move is made so that if that move is backed up, the piece can be replaced on the board. We then have the player which plays the game and can either be a human player or a computer player, the latter of which looks at the board and makes a random move from the ones that are available. We also have the wintypes which are used for determining how the user won or lost the game.


Move
- start: Position
- end: Position
+ getStart(): Position
+ getEnd(): Position
+ getStartRow(): int
+ getStartCell(): int
+ getEndRow(): int
+ getEndCell(): int
+ determineMoveType(): MoveType
<<enumeration>> MoveType
+ SIMPLE_MOVE
+ SIMPLE_MOVE_BACK
+ JUMP_MOVE
+ JUMP_MOVE_BACK
+ KING_PIECE
<<enumeration>> SpaceColor
+ BLACK
+ WHITE
Space
- cellIdex: int
- piece: Piece
- color: SpaceColor
+ getCellIdx(): int
+ isValid(): boolean
+ getColor(): SpaceColor
+ getPiece(): Piece
+ setPiece(piece : Piece): void
Row
- index: int
- spaces: ArrayList<Space>
+ getSpaces(): ArrayList<Space>
+ setSpaces(spaces : ArrayList<Space>): void
+ getIndex(): int
ComputerPlayer
+ static COMPUTER_PLAYER_NAME: String
+ computerMove(board: Board): Board
<<enumeration>> PieceColor
+ RED
+ WHITE
<<enumeration>> PieceType
+ SINGLE
+ KING
<<interface>> Iterator<Space>
+ iterator(): Iterator<T>
Player
- name: String
- opponent: Player
- color: PieceColor
+ getName(): String
+ getOpponent(): Player
+ setInGame(value : boolean): void
+ setOpponent(player : Player): void
+ setColor(color : PieceColor): void
+ getColor(): PieceColor
+ exitGame(): void
Position
- row: int
- cell: int
+ getRow(): int
+ getCell(): int
+ onBoard(): boolean
Removed
- piece: Piece
- position: Position
+ getPiece(): Piece
+ getPosition(): Position
Piece
- type: PieceType
- color: PieceColor
+ getType(): PieceType
+ getColor(): PieceColor
+ setType(type : PieceType): void
BoardView
- row: ArrayList<Row>
+ getRows(): ArrayList<Row>
+ createSpaces(): void
+ addPieces(): void
+ flipView(): BoardView
<<interface>> Iterator<Row>
+ iterator(): Iterator<T>
Board
- redPlayer: String
- whitePlayer: String
- broadView: BoardView
- flippedBoardView: BoardView
- activePlayer: PieceColor
+ getActiveBoardView(): BoardView
+ getInactiveBoardView(): BoardView
+ getActivePlayer(): PieceColor
+ setActivePlayer(color : PieceColor): void
+ containsInactivePiece(position : Position): boolean
+ getPieceAt(position : Position): Piece
- putPieceAt(position:Position, piece:Piece): void
+ movePiece(start:Position, end:Position): void
+ jumpWithPiece(start:Position, end:Position): Piece
+ noPieceInLanding(endPosition : Position): boolean
+ copy(): Board
+ isGameOver(): boolean
+ createWinMessage(): void
<<enumeration>> WinType
+ NO_MOVES
+ CAPTURE_ALL_PIECE

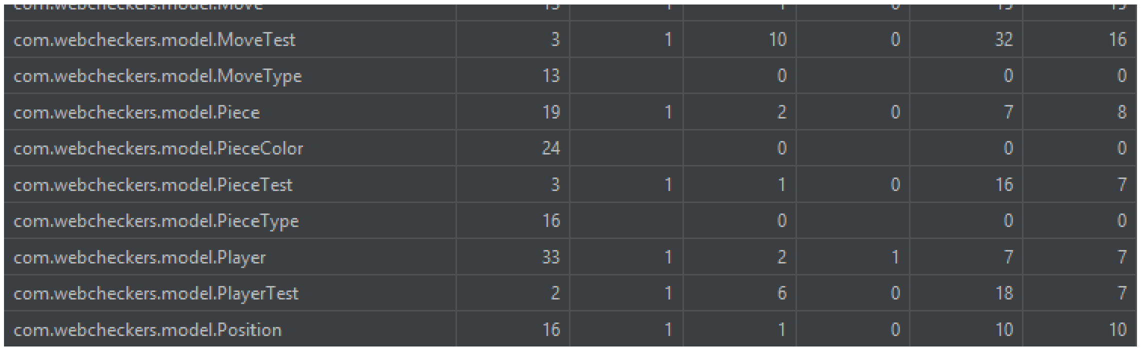
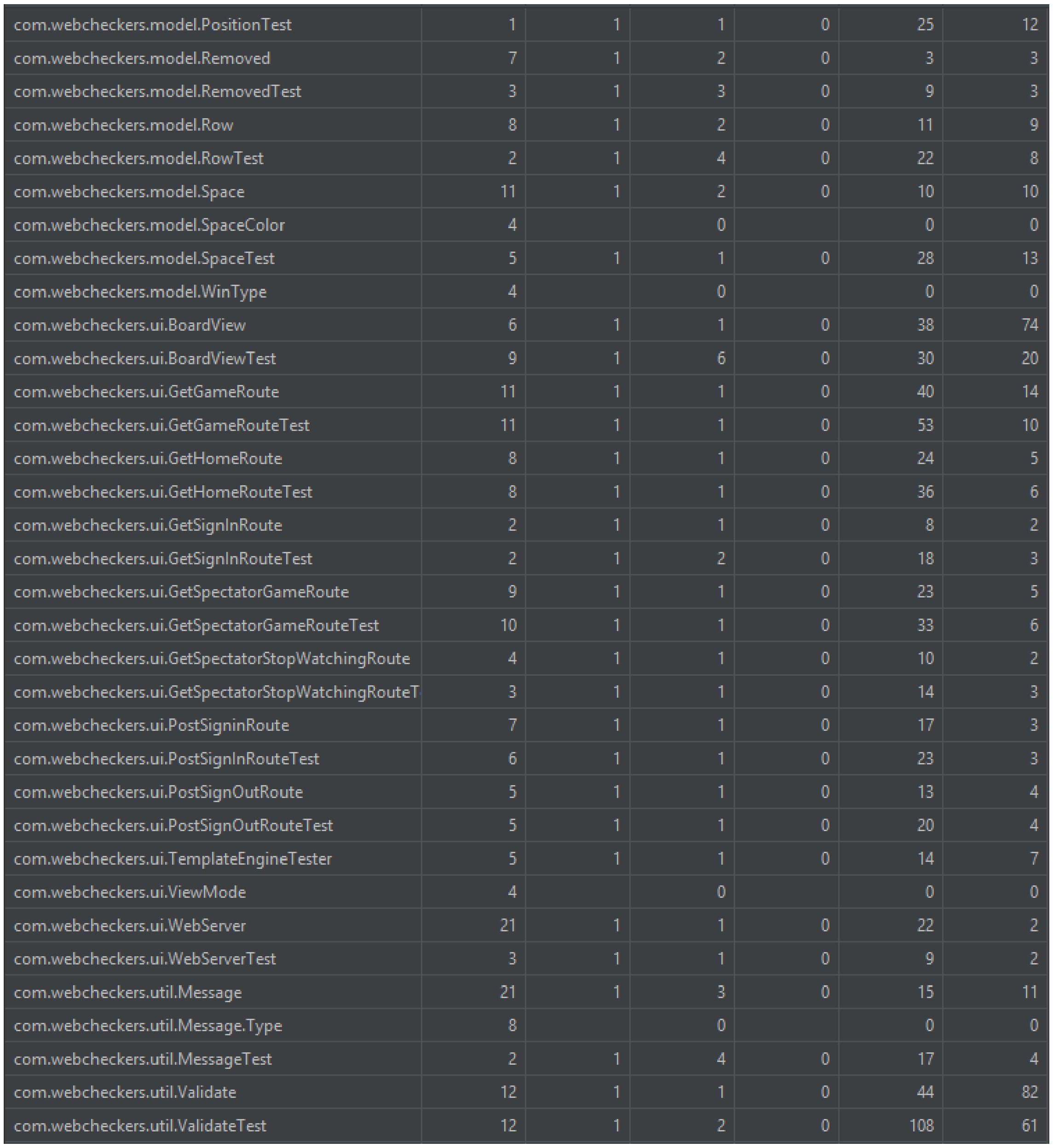
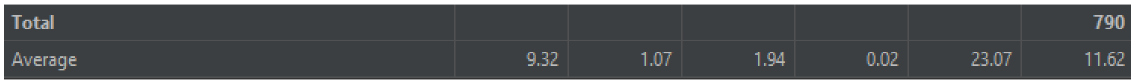
Code Metrics

Chidamber-Kemerer Metrics



Class metrics

class	CBO	DIT	LCOM	NOC	RFC	WMC
com.webcheckers.appl.GameCenter	35	1	2	0	33	32
com.webcheckers.appl.GameCenterTest	7	1	2	0	56	22
com.webcheckers.appl.PlayerLobby	20	1	2	0	30	19
com.webcheckers.appl.PlayerLobbyExceptions	4	4	0	0	2	1
com.webcheckers.appl.PlayerLobbyTest	3	1	13	0	39	20
com.webcheckers.appl.PlayerServices	4	1	1	0	3	3
com.webcheckers.appl.PlayerServicesTest	4	1	1	0	13	6
com.webcheckers.appl.PostBackupMoveRoute	8	1	1	0	20	4
com.webcheckers.appl.PostBackupMoveRouteTest	10	1	1	0	30	4
com.webcheckers.appl.PostCheckTurnRoute	6	1	1	0	14	3
com.webcheckers.appl.PostCheckTurnRouteTest	7	1	1	0	21	3
com.webcheckers.appl.PostResignGameRoute	7	1	1	0	19	5
com.webcheckers.appl.PostResignGameRouteTest	7	1	1	0	32	7
com.webcheckers.appl.PostSpectatorCheckTurnRoute	7	1	1	0	17	6
com.webcheckers.appl.PostSpectatorCheckTurnRouteT	9	1	1	0	31	7
com.webcheckers.appl.PostSubmitTurnRoute	14	1	1	0	43	19
com.webcheckers.appl.PostSubmitTurnRouteTest	14	1	1	0	61	20
com.webcheckers.appl.PostValidateMoveRoute	9	1	1	0	30	15
com.webcheckers.appl.PostValidateMoveRouteTest	10	1	1	0	33	7
com.webcheckers.Application	3	1	1	0	26	7
com.webcheckers.model.Board	36	1	1	0	47	57
com.webcheckers.model.BoardTest	8	1	22	0	87	47
com.webcheckers.model.ComputerPlayer	11	2	1	0	22	11
com.webcheckers.model.ComputerPlayerTest	10	1	1	0	18	4
com.webcheckers.model.Move	13	1	1	0	15	15

com.webcheckers.model.MoveTest	3	1	10	0	32	16
com.webcheckers.model.MoveType	13		0		0	0
com.webcheckers.model.Piece	19	1	2	0	7	8
com.webcheckers.model.PieceColor	24		0		0	0
com.webcheckers.model.PieceTest	3	1	1	0	16	7
com.webcheckers.model.PieceType	16		0		0	0
com.webcheckers.model.Player	33	1	2	1	7	7
com.webcheckers.model.PlayerTest	2	1	6	0	18	7
com.webcheckers.model.Position	16	1	1	0	10	10
com.webcheckers.model.PositionTest	1	1	1	0	25	12
com.webcheckers.model.Removed	7	1	2	0	3	3
com.webcheckers.model.RemovedTest	3	1	3	0	9	3
com.webcheckers.model.Row	8	1	2	0	11	9
com.webcheckers.model.RowTest	2	1	4	0	22	8
com.webcheckers.model.Space	11	1	2	0	10	10
com.webcheckers.model.SpaceColor	4		0		0	0
com.webcheckers.model.SpaceTest	5	1	1	0	28	13
com.webcheckers.model.WinType	4		0		0	0
com.webcheckers.ui.BoardView	6	1	1	0	38	74
com.webcheckers.ui.BoardViewTest	9	1	6	0	30	20
com.webcheckers.ui.GetGameRoute	11	1	1	0	40	14
com.webcheckers.ui.GetGameRouteTest	11	1	1	0	53	10
com.webcheckers.ui.GetHomeRoute	8	1	1	0	24	5
com.webcheckers.ui.GetHomeRouteTest	8	1	1	0	36	6
com.webcheckers.ui.GetSignInRoute	2	1	1	0	8	2
com.webcheckers.ui.GetSignInRouteTest	2	1	2	0	18	3
com.webcheckers.ui.GetSpectatorGameRoute	9	1	1	0	23	5
com.webcheckers.ui.GetSpectatorGameRouteTest	10	1	1	0	33	6
com.webcheckers.ui.GetSpectatorStopWatchingRoute	4	1	1	0	10	2
com.webcheckers.ui.GetSpectatorStopWatchingRouteT	3	1	1	0	14	3
com.webcheckers.ui.PostSigninRoute	7	1	1	0	17	3
com.webcheckers.ui.PostSignInRouteTest	6	1	1	0	23	3
com.webcheckers.ui.PostSignOutRoute	5	1	1	0	13	4
com.webcheckers.ui.PostSignOutRouteTest	5	1	1	0	20	4
com.webcheckers.ui.TemplateEngineTester	5	1	1	0	14	7
com.webcheckers.ui.ViewMode	4		0		0	0
com.webcheckers.ui.WebServer	21	1	1	0	22	2
com.webcheckers.ui.WebServerTest	3	1	1	0	9	2
com.webcheckers.util.Message	21	1	3	0	15	11
com.webcheckers.util.Message.Type	8		0		0	0
com.webcheckers.util.MessageTest	2	1	4	0	17	4
com.webcheckers.util.Validate	12	1	1	0	44	82
com.webcheckers.util.ValidateTest	12	1	2	0	108	61
Total						**790**
Average	9.32	1.07	1.94	0.02	23.07	11.62

This metric covers the code via the measurement theory. Each of the columns show a measurement that the metrics have taken into account. These are Coupling Between Objects, Depth of Inheritance Tree, Lack of Cohesion Methods, Number of Children, Response for a Class, and Weighted Method Complexity.

Starting from the Coupling Between Object measurement, a majority of the high coupling classes are testing classes and classes that we did not have control over, such as Message.java and WebServer.java. The other methods that have high coupling are routes and the board creation classes. These make sense because they are the backbone of the functionality. An action to take to lower this CBO measurement is to use more dependency inversion.

Next is the Depth of Inheritance Tree measurement. This shows the maximum length of a path from a class to a root class in the inheritance structure. Generally, having a higher DIT value represents having high degrees of code reusability. In this case, mostly everything had a value of 1, which means that the code was not very reusable in the scope of the project. To improve upon this, more abstract classes should be made.

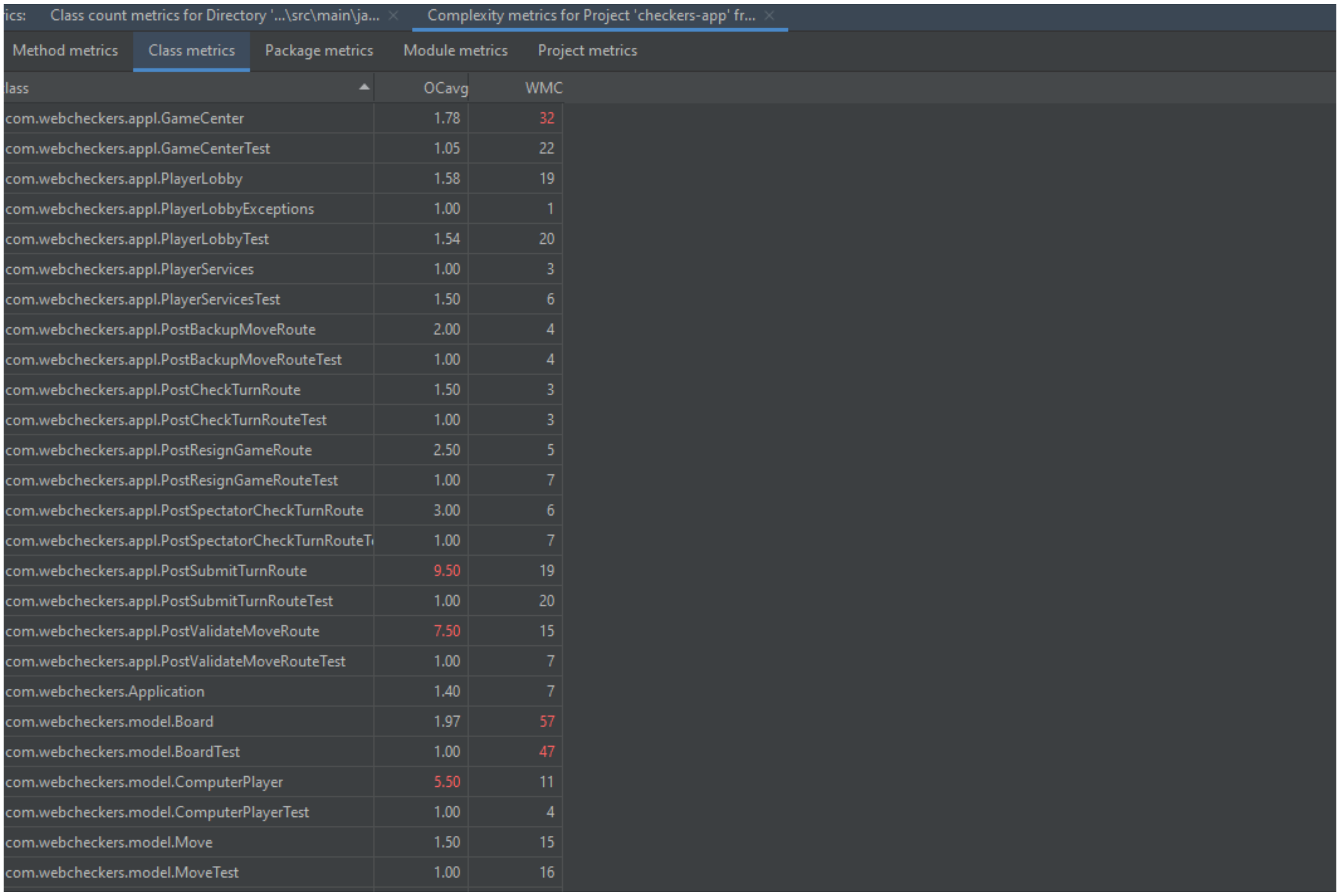
Next is the Lack of Cohesion Methods measurement. This just shows the lack of cohesions between the classes and the lower this is, the higher the cohesion. In our case, the only methods that had low cohesion were testing methods. There are not many adjustments to make for this specific metric.

Next is the Number of Children measurement. This shows the amount of created dependencies in the code. Generally, a lower number is better because this also leads to lower coupling. In our case there was only one class that had one child and the rest had none. Nothing substantial can be improved for this particular method.

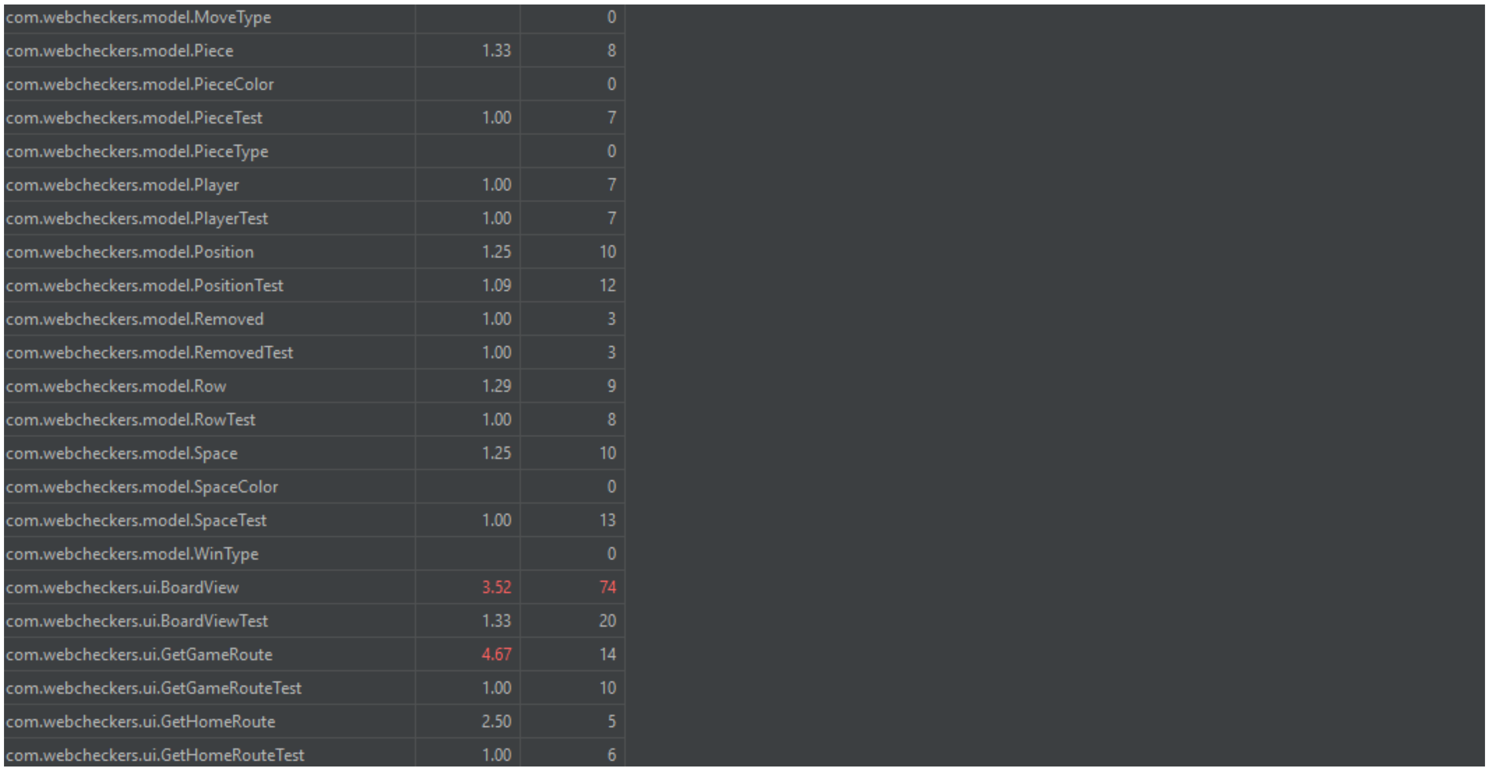
Next is the Response for a Class method measurement. This is just the amount of unique different methods that are called in the class. There is no particular “good measurement” for this metric, however making sure that in the code that one method is not called multiple times for the same value should be prioritized.

Lastly is the Weighted Method Complexity metric. This is the weighted sum of methods implemented in a class. Like the previous measurement, there is no particular “good measurement”, however there are few outliers to the general pool. The larger metrics include BoardView, Validate, and Board. These classes hold a bulk of the application’s functionality. There is not necessarily any specific improvement that can be made, however making a few of these classes less complex would assist with method readability.

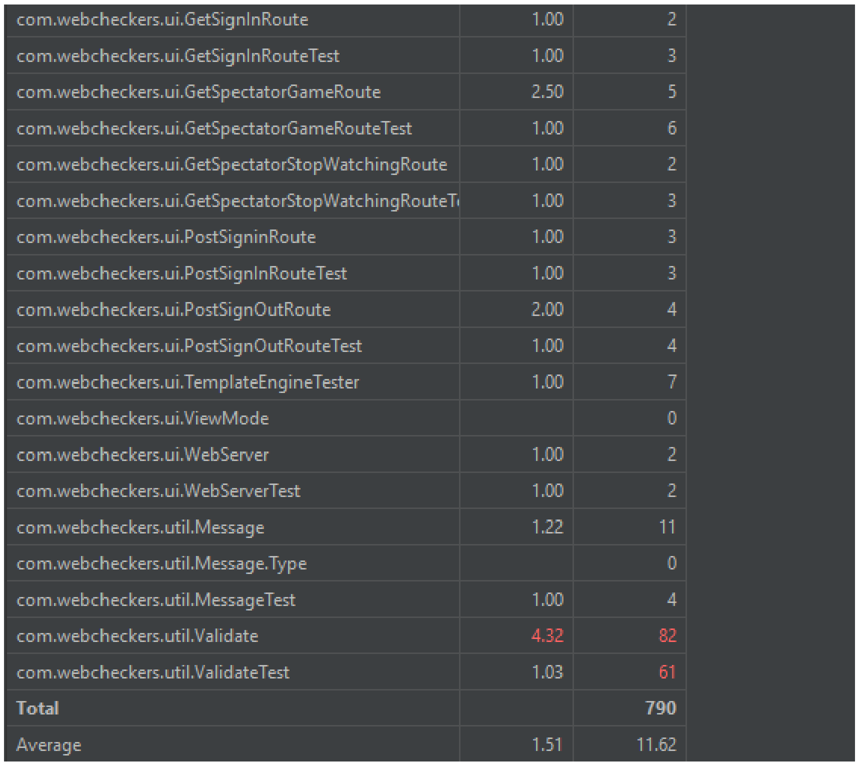
Cyclomatic Complexity Metrics


ics: Class count metrics for Directory '...\src\main\ja...
Complexity metrics for Project 'checkers-app' fr...
Method metrics
Class metrics
Package metrics
Module metrics
Project metrics

lass	OCavg	WMC
com.webcheckers.appl.GameCenter	1.78	32
com.webcheckers.appl.GameCenterTest	1.05	22
com.webcheckers.appl.PlayerLobby	1.58	19
com.webcheckers.appl.PlayerLobbyExceptions	1.00	1
com.webcheckers.appl.PlayerLobbyTest	1.54	20
com.webcheckers.appl.PlayerServices	1.00	3
com.webcheckers.appl.PlayerServicesTest	1.50	6
com.webcheckers.appl.PostBackupMoveRoute	2.00	4
com.webcheckers.appl.PostBackupMoveRouteTest	1.00	4
com.webcheckers.appl.PostCheckTurnRoute	1.50	3
com.webcheckers.appl.PostCheckTurnRouteTest	1.00	3
com.webcheckers.appl.PostResignGameRoute	2.50	5
com.webcheckers.appl.PostResignGameRouteTest	1.00	7
com.webcheckers.appl.PostSpectatorCheckTurnRoute	3.00	6
com.webcheckers.appl.PostSpectatorCheckTurnRouteT	1.00	7
com.webcheckers.appl.PostSubmitTurnRoute	9.50	19
com.webcheckers.appl.PostSubmitTurnRouteTest	1.00	20
com.webcheckers.appl.PostValidateMoveRoute	7.50	15
com.webcheckers.appl.PostValidateMoveRouteTest	1.00	7
com.webcheckers.Application	1.40	7
com.webcheckers.model.Board	1.97	57
com.webcheckers.model.BoardTest	1.00	47
com.webcheckers.model.ComputerPlayer	5.50	11
com.webcheckers.model.ComputerPlayerTest	1.00	4
com.webcheckers.model.Move	1.50	15
com.webcheckers.model.MoveTest	1.00	16



com.webcheckers.model.MoveType		0
com.webcheckers.model.Piece	1.33	8
com.webcheckers.model.PieceColor		0
com.webcheckers.model.PieceTest	1.00	7
com.webcheckers.model.PieceType		0
com.webcheckers.model.Player	1.00	7
com.webcheckers.model.PlayerTest	1.00	7
com.webcheckers.model.Position	1.25	10
com.webcheckers.model.PositionTest	1.09	12
com.webcheckers.model.Removed	1.00	3
com.webcheckers.model.RemovedTest	1.00	3
com.webcheckers.model.Row	1.29	9
com.webcheckers.model.RowTest	1.00	8
com.webcheckers.model.Space	1.25	10
com.webcheckers.model.SpaceColor		0
com.webcheckers.model.SpaceTest	1.00	13
com.webcheckers.model.WinType		0
com.webcheckers.ui.BoardView	3.52	74
com.webcheckers.ui.BoardViewTest	1.33	20
com.webcheckers.ui.GetGameRoute	4.67	14
com.webcheckers.ui.GetGameRouteTest	1.00	10
com.webcheckers.ui.GetHomeRoute	2.50	5
com.webcheckers.ui.GetHomeRouteTest	1.00	6

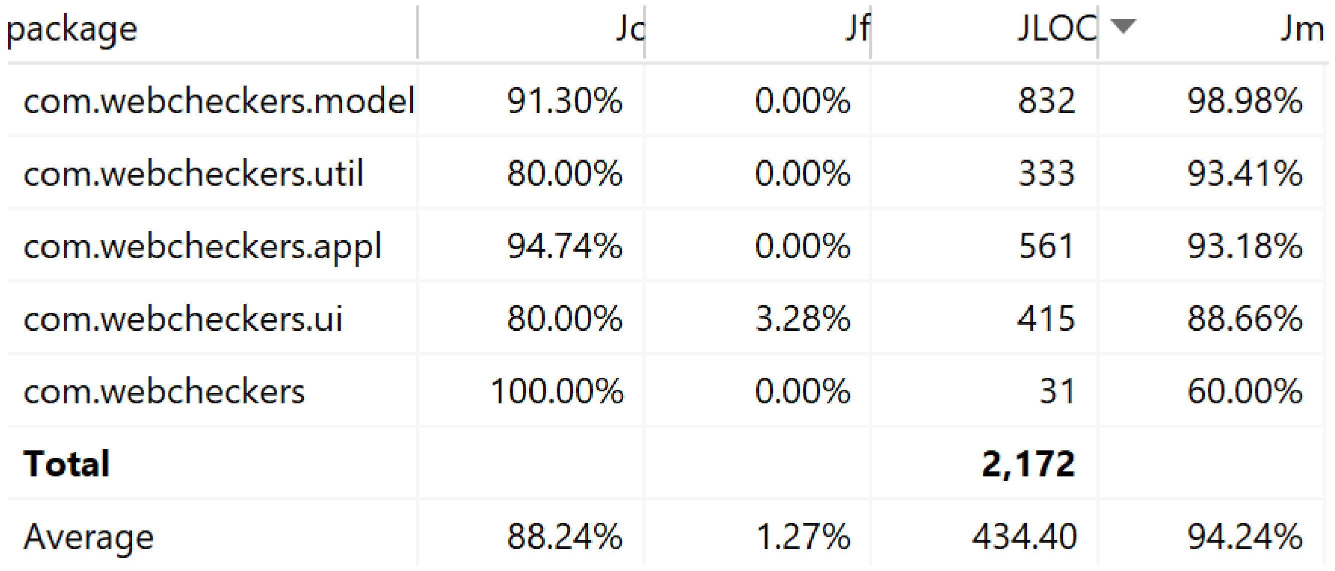


com.webcheckers.ui.GetSignInRoute	1.00	2
com.webcheckers.ui.GetSignInRouteTest	1.00	3
com.webcheckers.ui.GetSpectatorGameRoute	2.50	5
com.webcheckers.ui.GetSpectatorGameRouteTest	1.00	6
com.webcheckers.ui.GetSpectatorStopWatchingRoute	1.00	2
com.webcheckers.ui.GetSpectatorStopWatchingRouteT	1.00	3
com.webcheckers.ui.PostSigninRoute	1.00	3
com.webcheckers.ui.PostSignInRouteTest	1.00	3
com.webcheckers.ui.PostSignOutRoute	2.00	4
com.webcheckers.ui.PostSignOutRouteTest	1.00	4
com.webcheckers.ui.TemplateEngineTester	1.00	7
com.webcheckers.ui.ViewMode		0
com.webcheckers.ui.WebServer	1.00	2
com.webcheckers.ui.WebServerTest	1.00	2
com.webcheckers.util.Message	1.22	11
com.webcheckers.util.Message.Type		0
com.webcheckers.util.MessageTest	1.00	4
com.webcheckers.util.Validate	4.32	82
com.webcheckers.util.ValidateTest	1.03	61
Total		**790**
Average	1.51	11.62

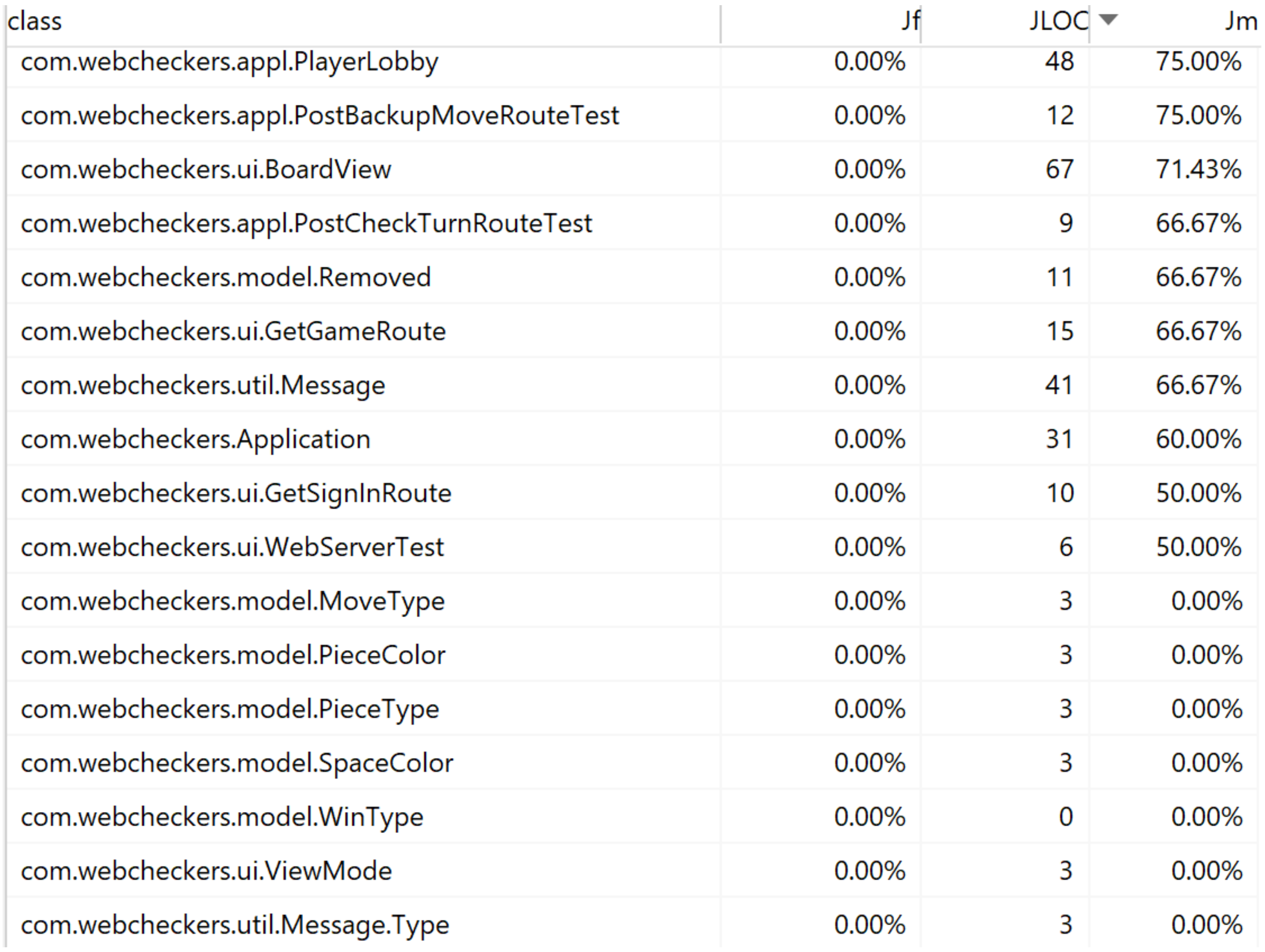
From the total number calculated it shows that our code is very complex. A high Cyclomatic complexity makes the code harder to test. It also makes our code harder to understand. Classes with a higher number are harder to test and they were. To have an easier to test and better to understand code complexity must be lower.

If you take a look at the numbers you will notice how the areas with the most complexity would be our test classes. Having a test class that tests very well may become very complex. Classes that are not labeled as tests should not be complex. The validate class unfortunately is our most complex class. This is a result of what we needed to validate. Validate handles a lot of moving rules. These rules create the complexity of the validate class. Without a complex validate class the rules for moving a piece in checkers would not be there. Another class labeled as complex is the board class. Board while in regular use is not complex at all and is easy to understand. In the board class most of its complexity comes from test methods to help out with testing the game. The same thing happens with the BoardView class. BoardView has multiple methods for creating a board for testing purposes. These methods create a lot of complexity but are not used under normal use.

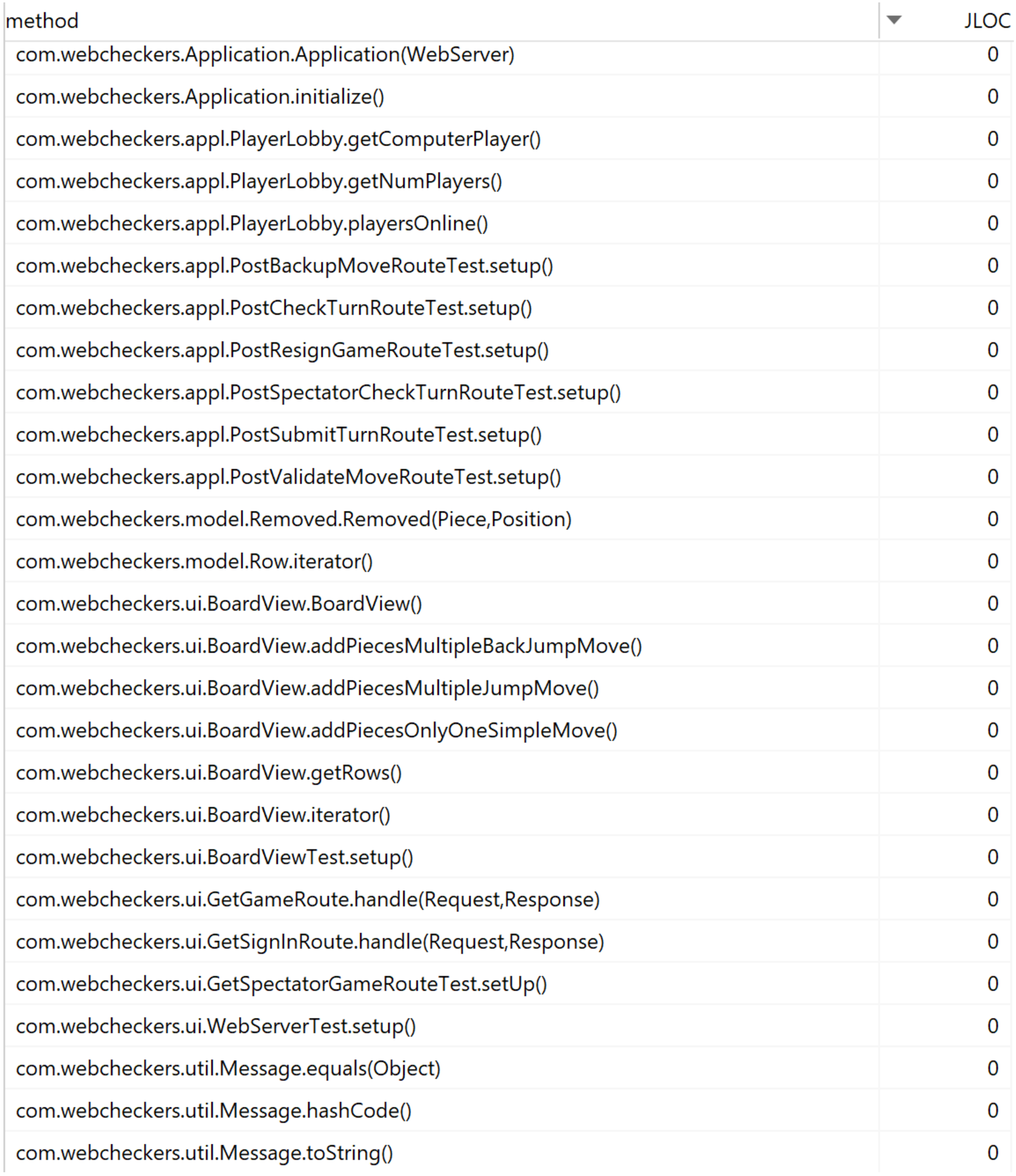
JavaDoc Metrics



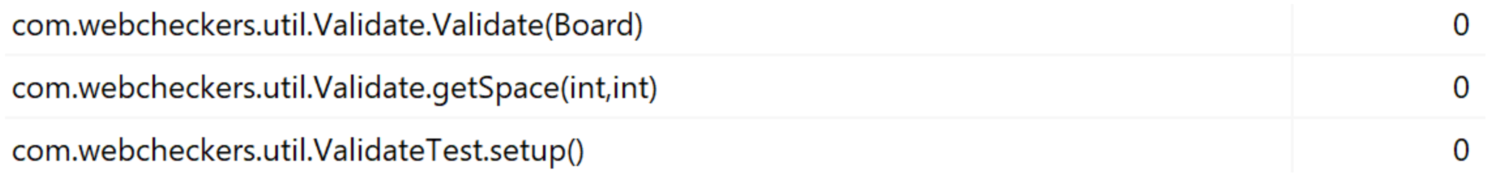
package	Jc	Jf	JLOC	Jm
com.webcheckers.model	91.30%	0.00%	832	98.98%
com.webcheckers.util	80.00%	0.00%	333	93.41%
com.webcheckers.appl	94.74%	0.00%	561	93.18%
com.webcheckers.ui	80.00%	3.28%	415	88.66%
com.webcheckers	100.00%	0.00%	31	60.00%
Total			**2,172**	
Average	88.24%	1.27%	434.40	94.24%



class	Jf	JLOC	Jm
com.webcheckers.appl.PlayerLobby	0.00%	48	75.00%
com.webcheckers.appl.PostBackupMoveRouteTest	0.00%	12	75.00%
com.webcheckers.ui.BoardView	0.00%	67	71.43%
com.webcheckers.appl.PostCheckTurnRouteTest	0.00%	9	66.67%
com.webcheckers.model.Removed	0.00%	11	66.67%
com.webcheckers.ui.GetGameRoute	0.00%	15	66.67%
com.webcheckers.util.Message	0.00%	41	66.67%
com.webcheckers.Application	0.00%	31	60.00%
com.webcheckers.ui.GetSignInRoute	0.00%	10	50.00%
com.webcheckers.ui.WebServerTest	0.00%	6	50.00%
com.webcheckers.model.MoveType	0.00%	3	0.00%
com.webcheckers.model.PieceColor	0.00%	3	0.00%
com.webcheckers.model.PieceType	0.00%	3	0.00%
com.webcheckers.model.SpaceColor	0.00%	3	0.00%
com.webcheckers.model.WinType	0.00%	0	0.00%
com.webcheckers.ui.ViewMode	0.00%	3	0.00%
com.webcheckers.util.Message.Type	0.00%	3	0.00%



method	JLOC
com.webcheckers.Application.Application(WebServer)	0
com.webcheckers.Application.initialize()	0
com.webcheckers.appl.PlayerLobby.getComputerPlayer()	0
com.webcheckers.appl.PlayerLobby.getNumPlayers()	0
com.webcheckers.appl.PlayerLobby.playersOnline()	0
com.webcheckers.appl.PostBackupMoveRouteTest.setup()	0
com.webcheckers.appl.PostCheckTurnRouteTest.setup()	0
com.webcheckers.appl.PostResignGameRouteTest.setup()	0
com.webcheckers.appl.PostSpectatorCheckTurnRouteTest.setup()	0
com.webcheckers.appl.PostSubmitTurnRouteTest.setup()	0
com.webcheckers.appl.PostValidateMoveRouteTest.setup()	0
com.webcheckers.model.Removed.Removed(Piece,Position)	0
com.webcheckers.model.Row.iterator()	0
com.webcheckers.ui.BoardView.BoardView()	0
com.webcheckers.ui.BoardView.addPiecesMultipleBackJumpMove()	0
com.webcheckers.ui.BoardView.addPiecesMultipleJumpMove()	0
com.webcheckers.ui.BoardView.addPiecesOnlyOneSimpleMove()	0
com.webcheckers.ui.BoardView.getRows()	0
com.webcheckers.ui.BoardView.iterator()	0
com.webcheckers.ui.BoardViewTest.setup()	0
com.webcheckers.ui.GetGameRoute.handle(Request,Response)	0
com.webcheckers.ui.GetSignInRoute.handle(Request,Response)	0
com.webcheckers.ui.GetSpectatorGameRouteTest.setUp()	0
com.webcheckers.ui.WebServerTest.setup()	0
com.webcheckers.util.Message.equals(Object)	0
com.webcheckers.util.Message.hashCode()	0
com.webcheckers.util.Message.toString()	0



com.webcheckers.util.Validate.Validate(Board)	0
com.webcheckers.util.Validate.getSpace(int,int)	0
com.webcheckers.util.ValidateTest.setup()	0

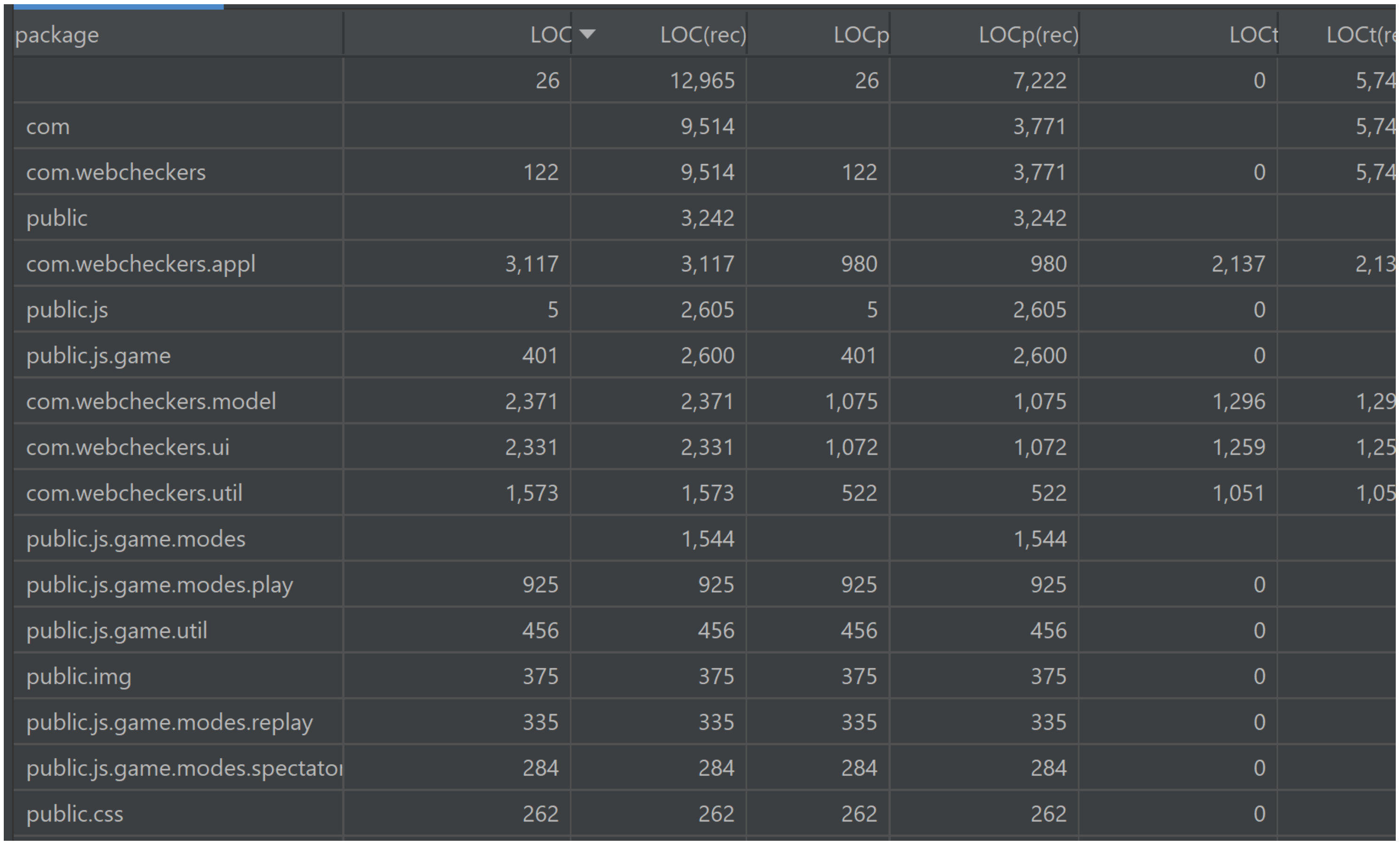
As there are no predefined targets for the Javadoc coverage of the project, the team came up with our own standards for what we deemed satisfactory. At a class level, the unsatisfactory range was those that have below 80% coverage, and at the method level, the unsatisfactory range was those that have 0 lines of Javadoc comments.

If the metrics are viewed at the package level, it is shown that the class outside of any tier, the Application class, has Javadoc method coverage (Jm) of 60.00%. Although this number is technically accurate, it does not include the single line comments that are given above some classes, meaning that the class has more comments than it appears at first glance. For a better metric at this level, these single line comments can be converted to proper javadoc comments. Looking at the Javadoc field coverage (Jf) metric at this level, it is evident that nearly every field is not commented with the only coverage.

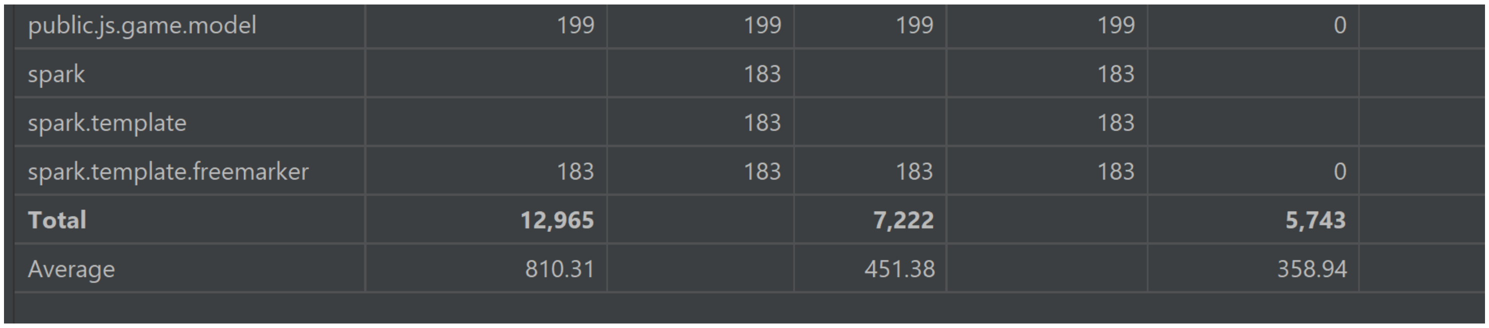
Looking at the metrics from a class level, the classes that have 0% Javadoc method coverage are those that are used as enums including MoveType, PieceColor, and PieceType. In these classes, there are no methods, and as such, the team is making no recommendations to better the Javadoc method coverage for them. Looking at the Javadoc Lines of Code (JLOC) method for these same classes, it is shown that the WinType enum has no comments whatsoever. The team is recommending that a class comment be added to this class to remedy this issue. Those that have some Javadoc method coverage, but not enough to meet coverage standards, are those that have many functions that are not commented in relation to the number of functions in the class itself. The commenting of these methods is discussed further in the next section.

The Javadoc coverage metrics at the method level allows for it to be known exactly which methods do not have comments associated with them. From this level, it is evident that across multiple test classes across multiple tiers, the ‘setup’ function which runs prior to each test is not commented. Although the use of this function is somewhat evident by its name along with the ‘@BeforeEach’ tag, the team is recommending putting a brief description such as ‘This function runs before each test to set up mock objects for testing’ be added. Another function type that is frequently seen to not have comments are those that override existing functions such as equals, or handle. In this case, it is recommended that comments be added to these functions using the ‘@inheritDoc’ tag. Yet another frequently uncommented function type is constructors, and it is recommended that comments be added to these functions, at the very least informing users of what the associated parameters are. Some other uncommented functions are not used in the application, such as ‘BoardView.addPiecesMultipleBackJumpMove’, and the recommendation is to delete these functions entirely rather than add comments to them.

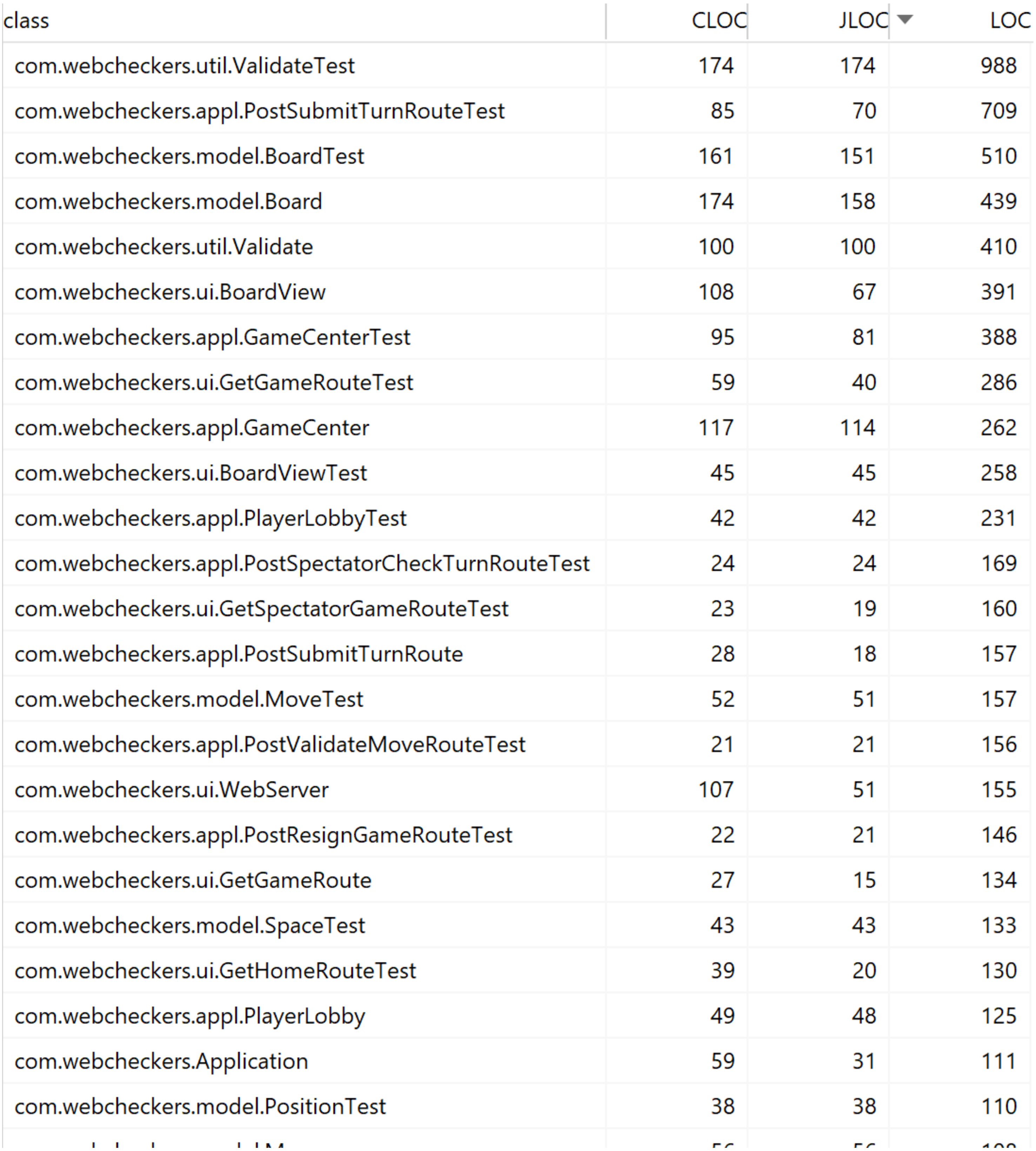
Lines of Code Metrics



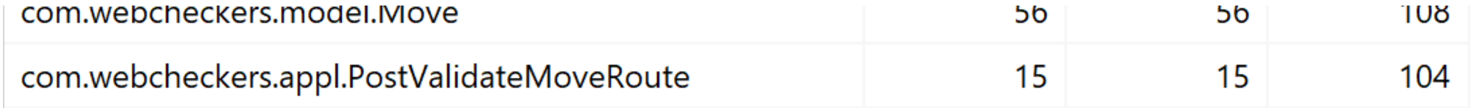
package	LOC	LOC(rec)	LOCp	LOCp(rec)	LOCt	LOCt(re
	26	12,965	26	7,222	0	5,74
com		9,514		3,771		5,74
com.webcheckers	122	9,514	122	3,771	0	5,74
public		3,242		3,242		
com.webcheckers.appl	3,117	3,117	980	980	2,137	2,13
public.js	5	2,605	5	2,605	0	
public.js.game	401	2,600	401	2,600	0	
com.webcheckers.model	2,371	2,371	1,075	1,075	1,296	1,29
com.webcheckers.ui	2,331	2,331	1,072	1,072	1,259	1,25
com.webcheckers.util	1,573	1,573	522	522	1,051	1,05
public.js.game.modes		1,544		1,544		
public.js.game.modes.play	925	925	925	925	0	
public.js.game.util	456	456	456	456	0	
public.img	375	375	375	375	0	
public.js.game.modes.replay	335	335	335	335	0	
public.js.game.modes.spectator	284	284	284	284	0	
public.css	262	262	262	262	0	



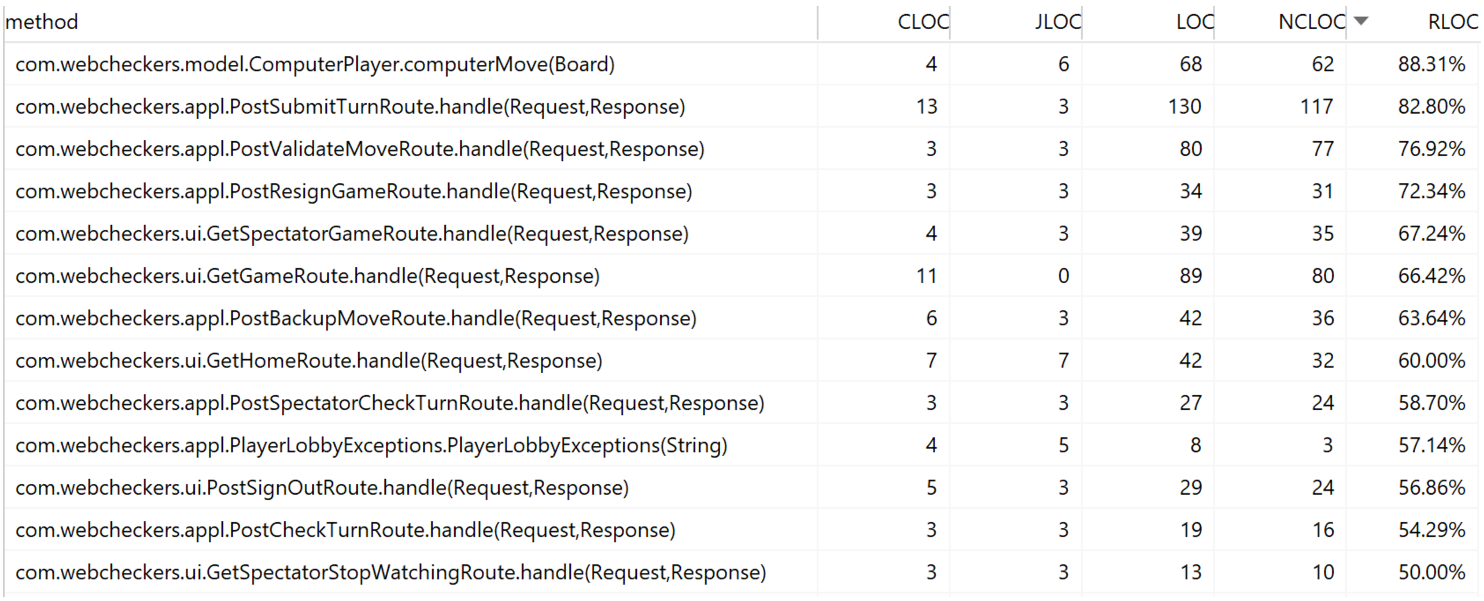
public.js.game.model	199	199	199	199	0	
spark		183		183		
spark.template		183		183		
spark.template.freemarker	183	183	183	183	0	
Total	**12,965**		**7,222**		**5,743**	
Average	810.31		451.38		358.94	



class	CLOC	JLOC	LOC
com.webcheckers.util.ValidateTest	174	174	988
com.webcheckers.appl.PostSubmitTurnRouteTest	85	70	709
com.webcheckers.model.BoardTest	161	151	510
com.webcheckers.model.Board	174	158	439
com.webcheckers.util.Validate	100	100	410
com.webcheckers.ui.BoardView	108	67	391
com.webcheckers.appl.GameCenterTest	95	81	388
com.webcheckers.ui.GetGameRouteTest	59	40	286
com.webcheckers.appl.GameCenter	117	114	262
com.webcheckers.ui.BoardViewTest	45	45	258
com.webcheckers.appl.PlayerLobbyTest	42	42	231
com.webcheckers.appl.PostSpectatorCheckTurnRouteTest	24	24	169
com.webcheckers.ui.GetSpectatorGameRouteTest	23	19	160
com.webcheckers.appl.PostSubmitTurnRoute	28	18	157
com.webcheckers.model.MoveTest	52	51	157
com.webcheckers.appl.PostValidateMoveRouteTest	21	21	156
com.webcheckers.ui.WebServer	107	51	155
com.webcheckers.appl.PostResignGameRouteTest	22	21	146
com.webcheckers.ui.GetGameRoute	27	15	134
com.webcheckers.model.SpaceTest	43	43	133
com.webcheckers.ui.GetHomeRouteTest	39	20	130
com.webcheckers.appl.PlayerLobby	49	48	125
com.webcheckers.Application	59	31	111
com.webcheckers.model.PositionTest	38	38	110
com.webcheckers.model.M	56	56	108



com.webcheckers.model.Move	56	56	108
com.webcheckers.appl.PostValidateMoveRoute	15	15	104



method	CLOC	JLOC	LOC	NCLOC	RLOC
com.webcheckers.model.ComputerPlayer.computerMove(Board)	4	6	68	62	88.31%
com.webcheckers.appl.PostSubmitTurnRoute.handle(Request,Response)	13	3	130	117	82.80%
com.webcheckers.appl.PostValidateMoveRoute.handle(Request,Response)	3	3	80	77	76.92%
com.webcheckers.appl.PostResignGameRoute.handle(Request,Response)	3	3	34	31	72.34%
com.webcheckers.ui.GetSpectatorGameRoute.handle(Request,Response)	4	3	39	35	67.24%
com.webcheckers.ui.GetGameRoute.handle(Request,Response)	11	0	89	80	66.42%
com.webcheckers.appl.PostBackupMoveRoute.handle(Request,Response)	6	3	42	36	63.64%
com.webcheckers.ui.GetHomeRoute.handle(Request,Response)	7	7	42	32	60.00%
com.webcheckers.appl.PostSpectatorCheckTurnRoute.handle(Request,Response)	3	3	27	24	58.70%
com.webcheckers.appl.PlayerLobbyExceptions.PlayerLobbyExceptions(String)	4	5	8	3	57.14%
com.webcheckers.ui.PostSignOutRoute.handle(Request,Response)	5	3	29	24	56.86%
com.webcheckers.appl.PostCheckTurnRoute.handle(Request,Response)	3	3	19	16	54.29%
com.webcheckers.ui.GetSpectatorStopWatchingRoute.handle(Request,Response)	3	3	13	10	50.00%

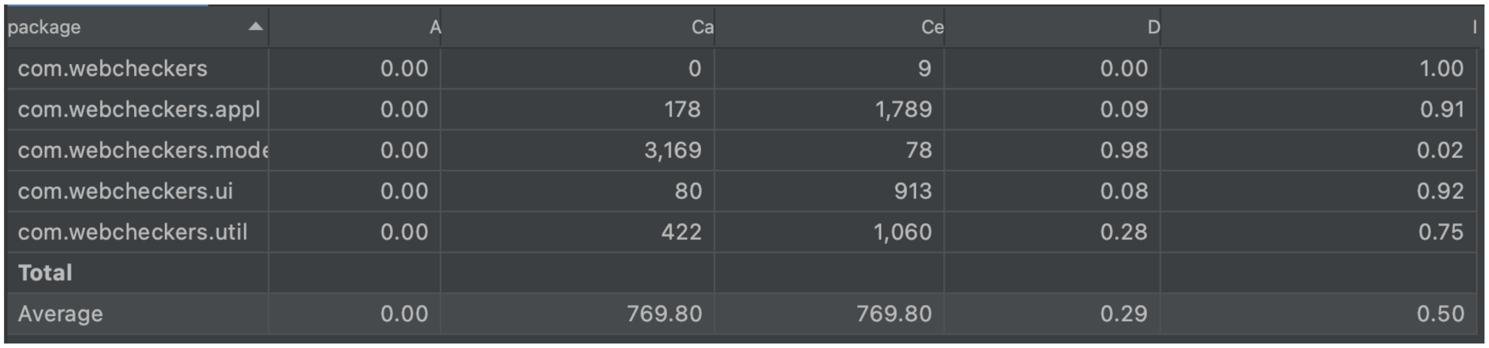
Throughout the project, the amount of lines of code used are well spread out at the package level, some classes did have a lot of code when compared to others which makes sense since these classes took care of most of the logic behind the web checkers game. Improvements can be made by creating classes that decrease the amount of work done on the other classes and distribute the amount of code a class has into various classes and therefore make the code easier to read and therefore easier to understand.

In terms of lines of code in the package level the amount of lines in the main packages which were the application, the ui and the util were well spread out since no one package takes care of the whole application.

At our class level we had many lines of code for the tests and the classes that handled the most amounts of logic for the game like the submit turn route and the validate class along with the test classes had many lines of code. For the test classes, it may have been better to make more reusable tests so that we did not have to repeat so many things that only had some changes to them and maybe making more reusable code would have been a good idea for the submitTurnRoute, board and the validate classes.

For some methods we have them containing a lot of the percentage of the class like in the case of the computer move method which could have been broken down into several helper methods that could make the code more readable and reduce the RLOC.

Martin Package Metrics



package	A	Ca	Ce	D	I
com.webcheckers	0.00	0	9	0.00	1.00
com.webcheckers.appl	0.00	178	1,789	0.09	0.91
com.webcheckers.mode	0.00	3,169	78	0.98	0.02
com.webcheckers.ui	0.00	80	913	0.08	0.92
com.webcheckers.util	0.00	422	1,060	0.28	0.75
Total					
Average	0.00	769.80	769.80	0.29	0.50

For Abstractness(A), it has 0 for all packages as there are no abstract classes in the project.

From Afferent Coupling(Ca), the ideal values are in the range of 0 to 500. High values of metric Ca usually suggest high component stability. In the model package, it has an unusually high value of incoming dependencies, given that we rely on many other classes like Piece, PieceColor, Row, Space, SpaceColor, Player, etc for the Board class itself. We can conclude that the package is extremely stable and concrete, therefore, this results in difficulties to modify codes as the probability of spreading such changes increases.

From Efferent Coupling(Ce), the application package has the most outgoing dependencies. The reason is that we rely on Validate.java to validate most of the moves and turns as the game begins. The util package also has an enormous amount of outgoing dependencies due to the access of structures of board which could break down into Players, Positions, Moves, etc. The higher value of outgoing dependencies, the more problems needed to deal with care. Given that the average value of Ca is still relatively high, it is not an ideal coding practice.

For Normalized Distance from Main Sequence(D), it should be as low as possible so that the components are located close to the main sequence. Given that the average value we have is 0.29, so it is pretty decent.

For Instability(I), packages should be very stable or unstable. In our project all packages except model packages are rather unstable due to the many outgoing dependencies and not many incoming dependencies. Although all packages fall into preferred values of Instability, the average is 0.5. In other words, we have intermediate stability for all packages on average. The solution is either to increase stability on all other packages except model packages or vice versa.

Design Improvements

During the implementation process, the design principles learned were not completely followed, so reflecting back on the way in which we built our application we noticed several areas in which we can improve upon. There are quite a few spots that we feel we could have better adhered to the single responsibility principle to make classes much more understandable and cohesive. For instance, the GameCenter is currently handling the implementation of the spectator feature, which should have been separated to a separate Spectator class that extends the Player class. This would allow the GameCenter to be more focused on the actual game related functionality, as well as make a simple class for spectator that in addition to making the code more cohesive would also adhere to the polymorphism principle. The team also concluded that the Player class could have been given more functionality by giving the player a gameID. This would make it unnecessary to rely on classes like PlayerLobby and GameCenter to get this value which helps the project adhere to the information expert principle.

Another area we noticed the design could improve on was in the Validate class, which at first was a solution to a specific problem we had in terms of how to check if a move was valid or not and it eventually became into a blob class that does a lot more than it’s supposed to do. A possible solution we concluded was the creation of other classes that separate the functionality of what was being handled in the validate, for example: making a validate for checking if a board is valid, and also making a validate that checks for the possible jumps, possible moves in separate classes makes the code more cohesive and understandable as well.

There are also improvements that can be done in the classes for the Board and the BoardView since we ended up creating several constructors for being able to test specific game conditions. A separate class that manages those cases could have been created to adhere better to the Open/Closed principle along with the high cohesion, as additional functionality not needed for the core behavior of the Board and BoardView classes does not need to be present in them.

Testing

The testing that was performed was created using Mockito. Mock classes were made to simulate the code actually running and seeing if we get the desired outputs. Some classes only had to test whether their getters were getting the correct information while other classes had more complicated methods to test. One major issue that was resolved via testing was a method in Validate.java, where a method was returning the same value regardless of what its arguments were. This allowed us to resolve that issue and carry on.

Acceptance Testing

The acceptance criteria for each of the user stories have been fully completed.

Unit Testing and Code Coverage

The unit testing strategy for this sprint was to get as high coverage percentage as possible. Every class that was actively used has become a friendly class, with a complete coverage of the code.

Coverage Tests

[all classes] [com.webcheckers.appl]

Coverage Summary for Package: com.webcheckers.appl

Package	Class, %	Method, %	Line, %
com.webcheckers.appl	100% (13/ 13)	98.1% (51/ 52)	95.5% (445/ 466)

Class ▲	Class, %	Method, %	Line, %
GameCenter	100% (1/ 1)	100% (18/ 18)	98.9% (88/ 89)
PlayerLobby	100% (1/ 1)	100% (12/ 12)	100% (43/ 43)
PlayerLobbyExceptions	100% (1/ 1)	100% (1/ 1)	100% (2/ 2)
PlayerServices	100% (1/ 1)	100% (3/ 3)	100% (7/ 7)
PostBackupMoveRoute	100% (1/ 1)	100% (2/ 2)	100% (36/ 36)
PostCheckTurnRoute	100% (1/ 1)	100% (2/ 2)	100% (16/ 16)
PostResignGameRoute	100% (1/ 1)	100% (2/ 2)	100% (30/ 30)
PostSignOutRoute	100% (1/ 1)	100% (2/ 2)	100% (25/ 25)
PostSigninRoute	100% (1/ 1)	66.7% (2/ 3)	80% (24/ 30)
PostSpectatorCheckTurnRoute	100% (1/ 1)	100% (2/ 2)	100% (25/ 25)
PostSubmitTurnRoute	100% (1/ 1)	100% (2/ 2)	86% (86/ 100)
PostValidateMoveRoute	100% (2/ 2)	100% (3/ 3)	100% (63/ 63)

[all classes] [com.webcheckers.model]

Coverage Summary for Package: com.webcheckers.model

Package	Class, %	Method, %	Line, %
com.webcheckers.model	100% (14/ 14)	100% (91/ 91)	96.6% (344/ 356)

Class ▲	Class, %	Method, %	Line, %
Board	100% (1/ 1)	100% (29/ 29)	100% (180/ 180)
ComputerPlayer	100% (1/ 1)	100% (3/ 3)	76.9% (40/ 52)
Move	100% (1/ 1)	100% (10/ 10)	100% (25/ 25)
MoveType	100% (1/ 1)	100% (2/ 2)	100% (6/ 6)
Piece	100% (1/ 1)	100% (6/ 6)	100% (13/ 13)
PieceColor	100% (1/ 1)	100% (2/ 2)	100% (3/ 3)
PieceType	100% (1/ 1)	100% (2/ 2)	100% (3/ 3)
Player	100% (1/ 1)	100% (7/ 7)	100% (14/ 14)
Position	100% (1/ 1)	100% (8/ 8)	100% (19/ 19)
Removed	100% (1/ 1)	100% (3/ 3)	100% (6/ 6)
Row	100% (1/ 1)	100% (7/ 7)	100% (14/ 14)
Space	100% (1/ 1)	100% (8/ 8)	100% (15/ 15)
SpaceColor	100% (1/ 1)	100% (2/ 2)	100% (3/ 3)
WinType	100% (1/ 1)	100% (2/ 2)	100% (3/ 3)

[all classes] [com.webcheckers.ui]

Coverage Summary for Package: com.webcheckers.ui

Package	Class, %	Method, %	Line, %
com.webcheckers.ui	100% (7/ 7)	94.4% (17/ 18)	91.7% (187/ 204)

Class	Class, %	Method, %	Line, %
GetGameRoute	100% (1/ 1)	100% (4/ 4)	100% (85/ 85)
GetHomeRoute	100% (1/ 1)	100% (3/ 3)	97% (32/ 33)
GetSignInRoute	100% (1/ 1)	100% (2/ 2)	100% (8/ 8)
GetSpectatorGameRoute	100% (1/ 1)	100% (2/ 2)	100% (36/ 36)
GetSpectatorStopWatchingRoute	100% (1/ 1)	100% (2/ 2)	100% (11/ 11)
ViewMode	100% (1/ 1)	100% (2/ 2)	100% (4/ 4)
WebServer	100% (1/ 1)	66.7% (2/ 3)	40.7% (11/ 27)

[all classes] [com.webcheckers.util]

Coverage Summary for Package: com.webcheckers.util

Package	Class, %	Method, %	Line, %
com.webcheckers.util	100% (3/ 3)	100% (31/ 31)	99.5% (220/ 221)

Class	Class, %	Method, %	Line, %
Message	100% (2/ 2)	100% (12/ 12)	100% (19/ 19)
Validate	100% (1/ 1)	100% (19/ 19)	99.5% (201/ 202)