

WebCheckers Design Documentation

Team Information

- Team name: Team D
- Team members
 - Nicholas Pueschel
 - Navid Nafiuazzaman
 - Alexander Wall
 - Andrew Hammerstone

Executive Summary

This project is an online web checkers application. The game allows two players to play a game of checkers adhering to standard rules. Players can see other online players and attempt to join a game. The game will continue until one player wins, or one player resigns.

Purpose

The goal of this project is to allow users to carry out an online game of checkers.

Glossary and Acronyms

Term	Definition
VO	Value Object
MVP	Minimal Viable Product
UI	User Interface
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
POJO	Plain Old Java Object

Requirements

The primary feature of the application is to start an online checkers game between two players. When a player connects to the web server they will be prompted to sign in to a player account. Once then they will be presented with a list of other online players. If that player is not in a game the user can create a game with that player. The webapp will then generate the checkers board, with the each player's pieces in their respected positions. A standard game of checkers will then begin with each player taking turns moving their pieces on the board. Each move is validated before it is completed. The players will switch off after every move. If a player jumps over an enemy piece, that piece is removed. If a player reaches the other side of the board with a piece, that piece becomes a king and gets access to king moves. The game will continue until one player loses all of his pieces, or a player resigns.

Definition of MVP

The minimum viable product for this application consists of being able to successfully start a game and perform valid moves in order to carry out the game between two players. Each player should be able to take alternating turns, and each turn should be validated.

MVP Features

The following Epics, are features that are required for the MVP

Epic - Start Game

Epic - Move Piece

Epic - Take Turn

Epic - Ensure Valid Move

Epic - Resignation

Enhancements

The following Epics are product enhancements that have been implemented

Epic - AI * AI Player Implementation * Autonomous Move Made by the Player * Designed Algorithm with the ability to win game against human player * Ability to become king and make valid moves

Epic - Help Page * Useful resources for the player to initiate moves and capture opponents' piece * Verified rules of American Checkers Game included * External engaging resources like Youtube Tutorials are included

Application Domain

The figure above represents the application domain of the project. As a player, one can access the help page, play against an AI player, or make a turn on a board during a current game. A turn is composed of a move which is then validated by MoveRule to ensure that the move, either a jump or a single, is lawful according to the American rules of Checkers. A move consists of two positions: a start and an end position, and each position corresponds to the coordinates of a space on the game board. A piece can occupy a space on the board and can be either red or white colored depending on the the player.

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 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 standard procedure for a user is to first connect to the homepage. This renders the web application interface. From there, the user can click on the Sign-in button, and this will render the sign-in page. On this

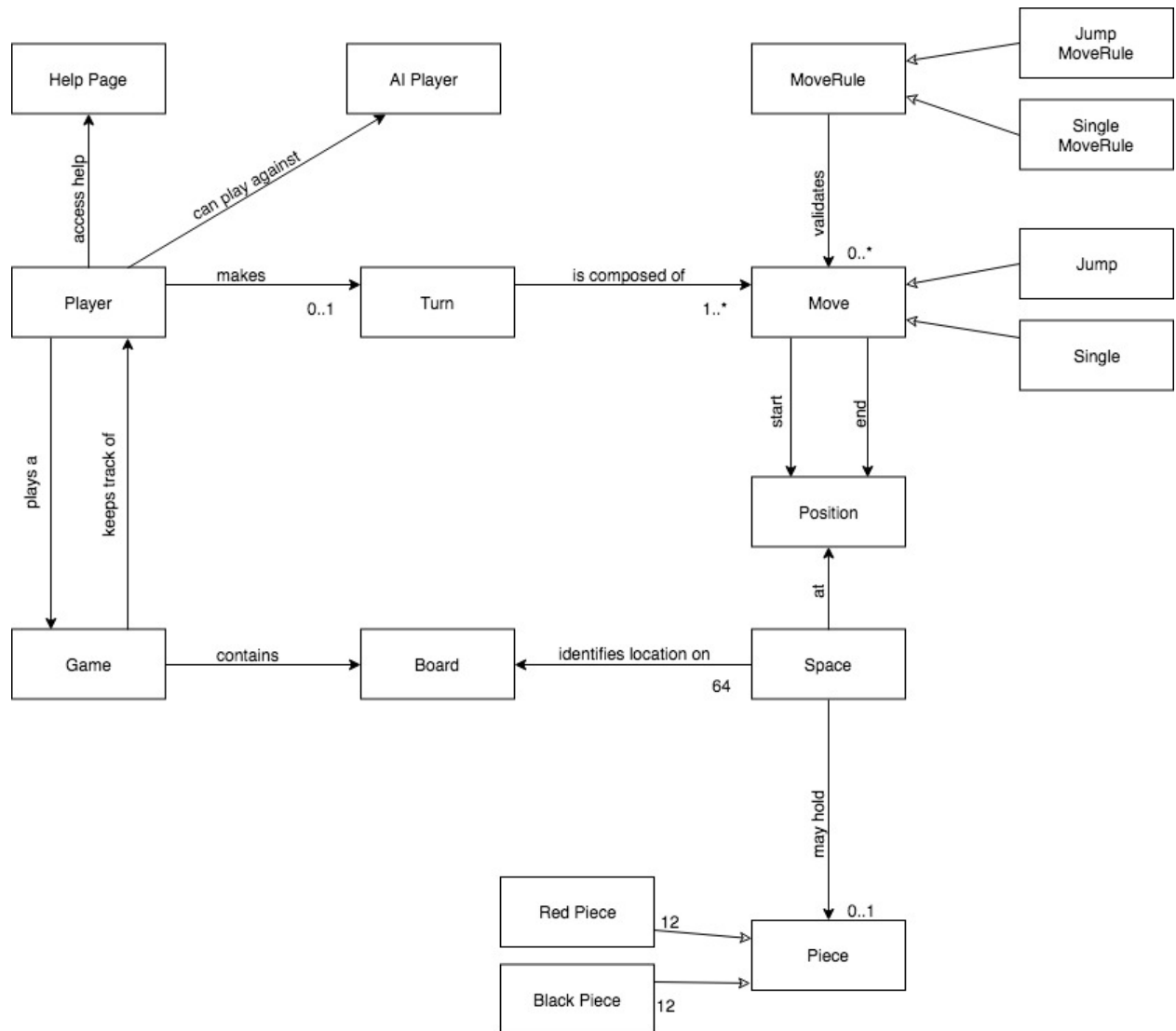


Figure 1: The WebCheckers Domain Model

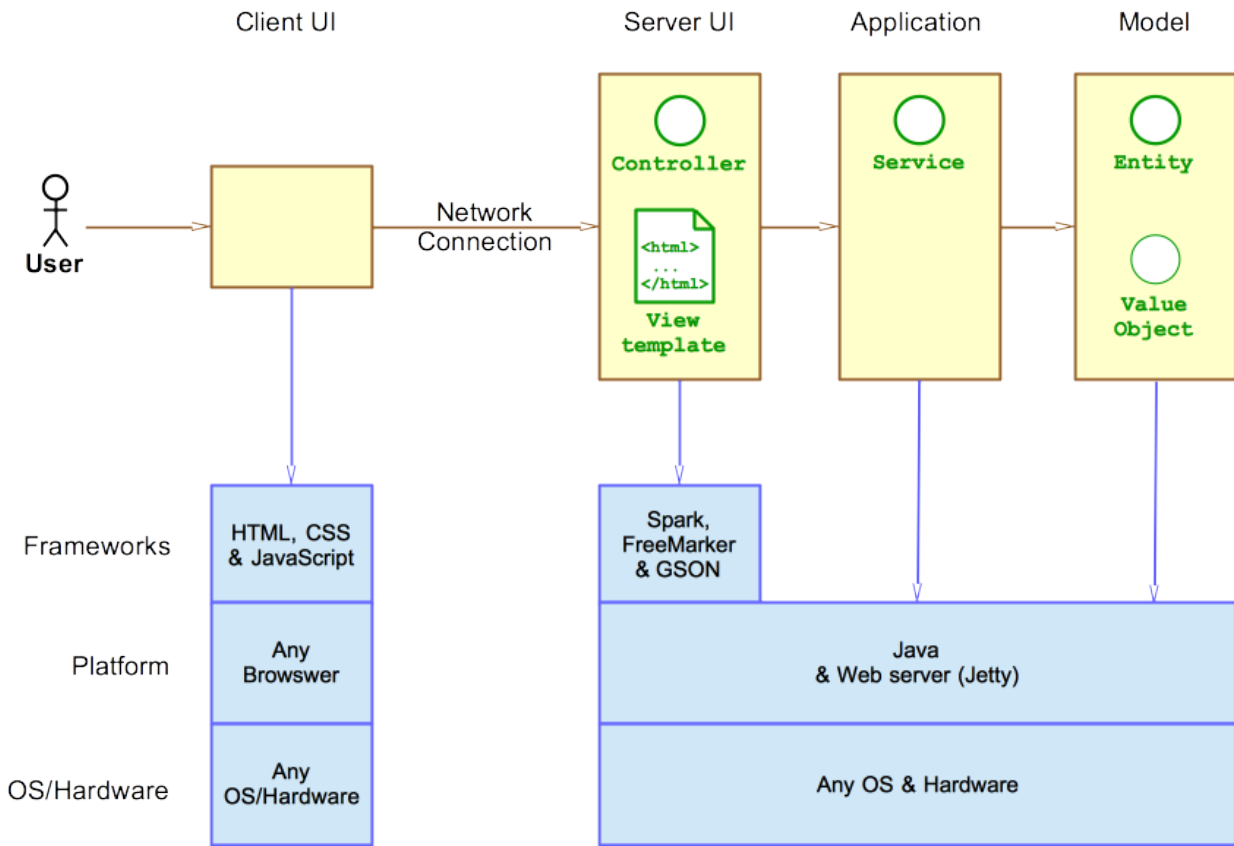


Figure 2: The Tiers & Layers of the Architecture

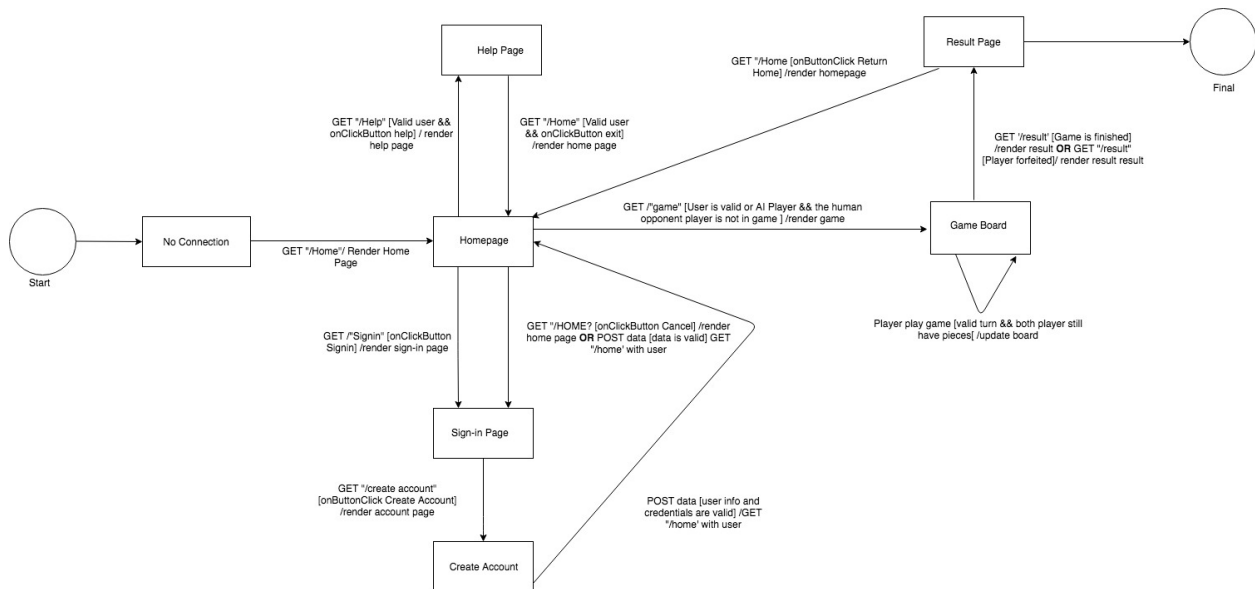


Figure 3: The WebCheckers Web Interface Statechart

UI Tier

[illegible]

Application Tier

Model Tier

Enhancement Feature - AI Player

Enhancement Feature - Help Page

5

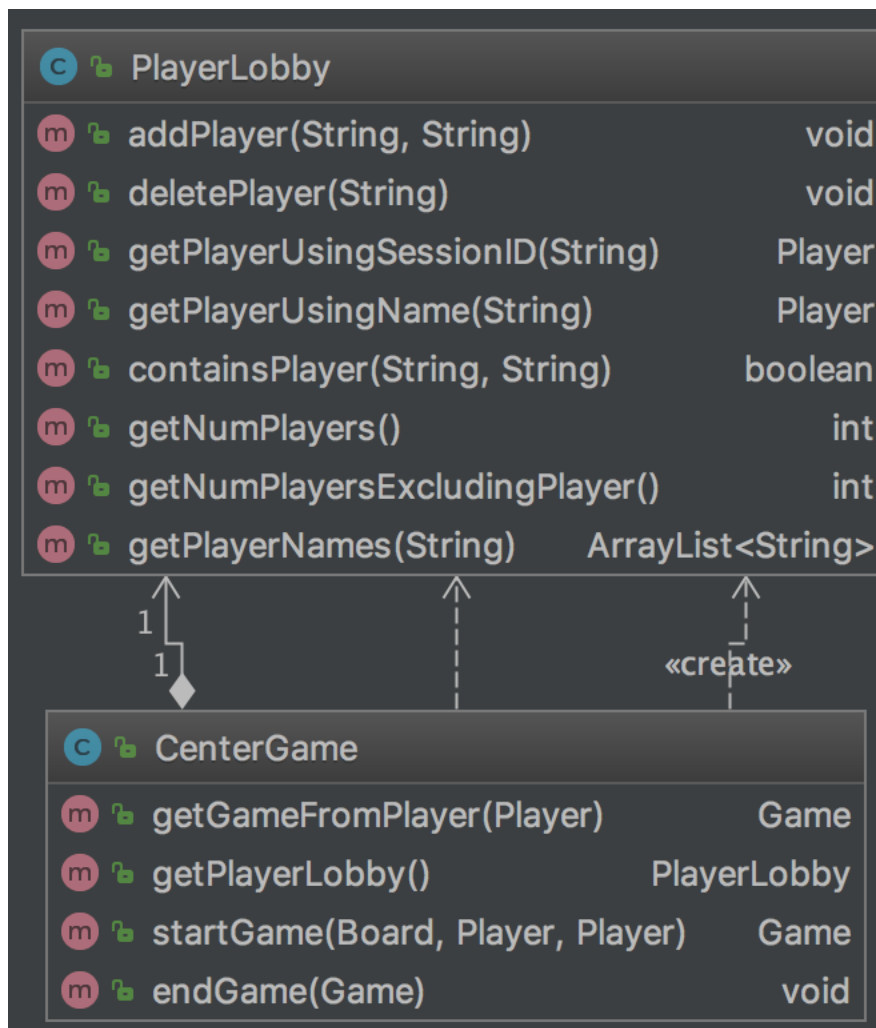


Figure 5: Application Tier Class Diagram

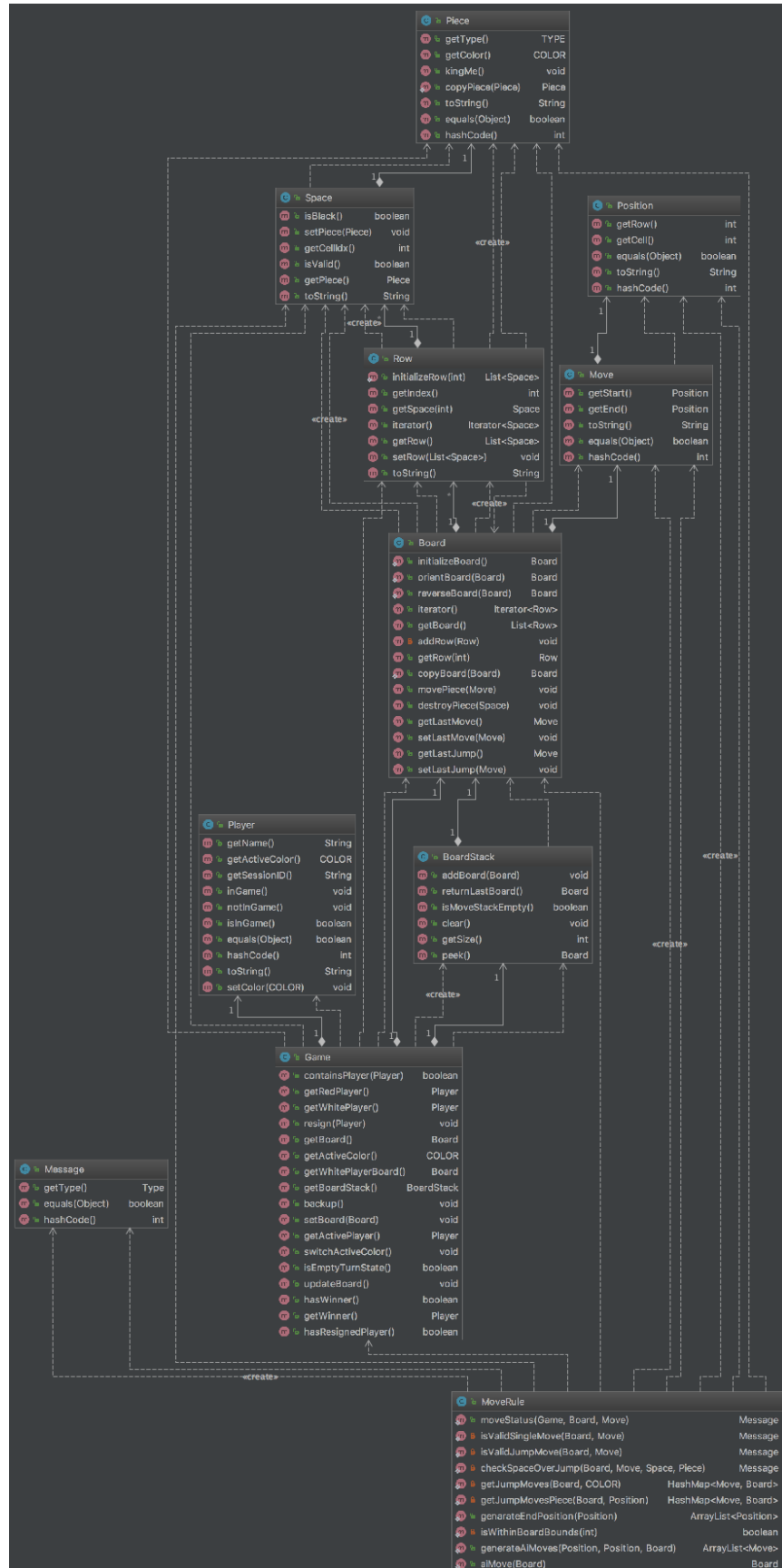










Figure 6: Model Tier Class Diagram

included * External engaging resources like Youtube Tutorials are included * Future Enhancement: Addition of more useful resources to increase player's gaming ability.

Code Coverage

The code coverage for our WebCheckers Game is around 90%. We tried to do extensive testing for our every class and methods ensuring that everything is working as expected.

Web Checkers a'la Spark/Java8

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.webcheckers.model		90%		90%	3	10	2	285	2	96	1	18
com.webcheckers.ui		93%		93%	10	58	23	260	37	41	1	13
com.webcheckers.appl		91%		91%	2	20	5	39	14	14	2	2
com.webcheckers		93%		93%	1	4	23	23	4	4	1	1
Total	2,950 of 3,122	92%	188 of 192	92%	16	252	53	607	57	155	5	34

Created with JaCoCo 0.8.0.201801022044

Figure 7: Code Coverage

Code Metrics

Lines of Code

com.webcheckers	84	3,920	84	2,777	0	1,143
com.webcheckers.appl	493	493	150	150	343	343
com.webcheckers.model	2,001	2,001	1,595	1,595	406	406
com.webcheckers.ui	1,342	1,342	948	948	394	394

Figure 8: Lines of Codes

From the lines of code analysis, there is nothing of importance or attention needed that would benefit from a redesign or re-implementation. It makes sense that the model package would contain the most lines of code for the product since it contains all the objects essential to the application. The application package should have the least amount of code since it controls the interaction between the model and UI packages and also the user's interaction to the application. However, one thing worth mentioning is the lines of code for the test cases. Test classes must be implemented properly to test out all models and events for the program rather than a selected few. Making changes to the size of each of these modules respective to line length would mostly be arbitrary and makes sense the way it is so it would most likely be best to leave this. Of course test classes must be implemented, but that is separate and is just a relative meter in of itself.

JavaDoc

package	Jc	Jf	JLOC ▼	Jm
com.webcheckers.appl	0.00%	0.00%	58	92.86%
com.webcheckers.ui	71.43%	1.43%	223	75.86%
com.webcheckers.model	57.14%	0.00%	305	63.00%
com.webcheckers	100.00%	0.00%	16	33.33%
Total			602	
Average	61.29%	0.91%	150.50	67.81%

In terms of Javadocs, most of the classes do not have field coverage as you can see from the first column of percentages. A recommendation

for this is to include javadocs for any variables or declarations in all the classes declared above; a good method is to have at least 60% in all coverage, and this includes method docstrings.

Martin Packaging

Metrics Martin packaging metrics for Project 'checkers-app' from Mon, 23 Apr 2018 00:01:44 EDT					
Package metrics					
package	A	Ca	Ce	D	I
com.webcheckers	0.00	1	2	0.33	0.67
com.webcheckers.appl	0.00	4	3	0.57	0.43
com.webcheckers.model	0.00	32	16	0.67	0.39
com.webcheckers.ui	0.00	15	44	0.25	0.76
Total					
Average	0.00	13.00	16.25	0.46	0.60

Reading from the abstract (A) column, all packages are reported to be concrete and not depended on any derived or abstract classes, which is appropriate for this application. For the afferent coupling (Ca) column, It makes sense to have the model package to have a high number since most of the classes the model package are objects dependent on the application class and UI in terms of the board and players. For efferent coupling (Ce), the UI package appears to be the most unstable since it has the highest number from the other two. In other words, any changes from the outside classes can have an impact on the UI class in this instance. A recommendation for this instability is to group together code that appears in the same cohesion so that any changes from the external classes would only affect that group of code in which the UI class can use without significantly having to change any of code in the UI classes. Another thing to mention is the normalized distance (D) and instability (I). The application package appears normalized between abstractness and stability while the model package is leaning towards the instability side and the UI package towards the abstract side. I believe no modification should be made due to the fact that the route classes should not be extended since their intended purposes is to send messages back and forth from client to server. For the model class, since it is halfway between normalized and instability, some minor implementation can be made to ensure balance. Overall it would be advisable to at least implement some helper functions in the code, and increase code reuse overall. A lot of classes use information they otherwise shouldn't have access to, there should be more getters for these things, and there should be additional classes made which be a "middle-man" and transfer data between the two classes, this way to follow information expert, and overall to decrease coupling, increase cohesion.

Complexity

team-project-team-d

Metrics Complexity metrics for Project 'checkers-app' from Sun, 22 Apr 2018 20:38:18 EDT

Method metrics	Class metrics	Package metrics	Module metrics	Project metrics
Class	OCavg	WMC		
com.webcheckers.appl.CenterGame	1.40	7		
com.webcheckers.appl.PlayerLobby	1.44	13		
com.webcheckers.Application	1.00	3		
com.webcheckers.model.Board	1.47	22		
com.webcheckers.model.BoardStack	1.12	9		
com.webcheckers.model.Game	1.82	40		
com.webcheckers.model.Message	1.25	5		
com.webcheckers.model.Message.Type		0		
com.webcheckers.model.MessageTest	1.00	3		
com.webcheckers.model.Move	1.20	6		
com.webcheckers.model.MoveRule	8.10	81		
com.webcheckers.model.MoveTest	1.00	5		
com.webcheckers.model.Piece	1.29	9		
com.webcheckers.model.Piece.COLOR		0		
com.webcheckers.model.Piece.TYPE		0		
com.webcheckers.model.PieceTest	1.00	5		
com.webcheckers.model.Player	1.09	12		
com.webcheckers.model.Position	1.20	6		
com.webcheckers.model.PositionTest	1.00	6		
com.webcheckers.model.Row	1.78	16		
com.webcheckers.model.RowTest	1.00	18		
com.webcheckers.model.Space	1.00	8		
com.webcheckers.model.SpaceTest	1.00	6		
com.webcheckers.model.Turn		0		
com.webcheckers.ui.GetGameRoute	4.50	9		
com.webcheckers.ui.GetHelpRoute	1.00	2		
com.webcheckers.ui.GetHomeRoute	1.67	5		
com.webcheckers.ui.GetHomeRouteTest	1.00	2		
com.webcheckers.ui.GetSignInRoute	1.00	3		
com.webcheckers.ui.GetSignInRouteTest	1.00	2		
com.webcheckers.ui.GetSignOutRoute	1.00	2		
com.webcheckers.ui.PostBackupMoveRoute	1.50	3		
com.webcheckers.ui.PostCheckTurn	2.00	4		
com.webcheckers.ui.PostResignGameRoute	1.50	3		
com.webcheckers.ui.PostSignInRoute	2.00	4		
com.webcheckers.ui.PostSignInRouteTest	1.00	5		
com.webcheckers.ui.PostSubmitTurn	4.00	8		
com.webcheckers.ui.PostSubmitTurnTest	1.00	2		
com.webcheckers.ui.PostValidateMove	2.50	5		
com.webcheckers.ui.PostValidateMoveTest	1.00	2		
com.webcheckers.ui.TemplateEngineTester	1.00	7		
com.webcheckers.ui.UserServices	1.60	8		
com.webcheckers.ui.UserServices.userNameResult		0		
com.webcheckers.ui.WebServer	1.00	2		
Total		358		
Average	1.67	8.14		

From the complexity analysis, it appears that the

MoveRule class contains the highest complexity in terms of following principles and maintaining greater polymorphism. Although this is a static class to test the validity of a move, the code within can be broken down to ensure less complexity and more reusability for other classes to use. For example, codes pertaining to generating moves should immediately be deleted from this class (i.e. aiMoves and getting a list of moves even though they are helper functions).

Chidamber-Kemerer

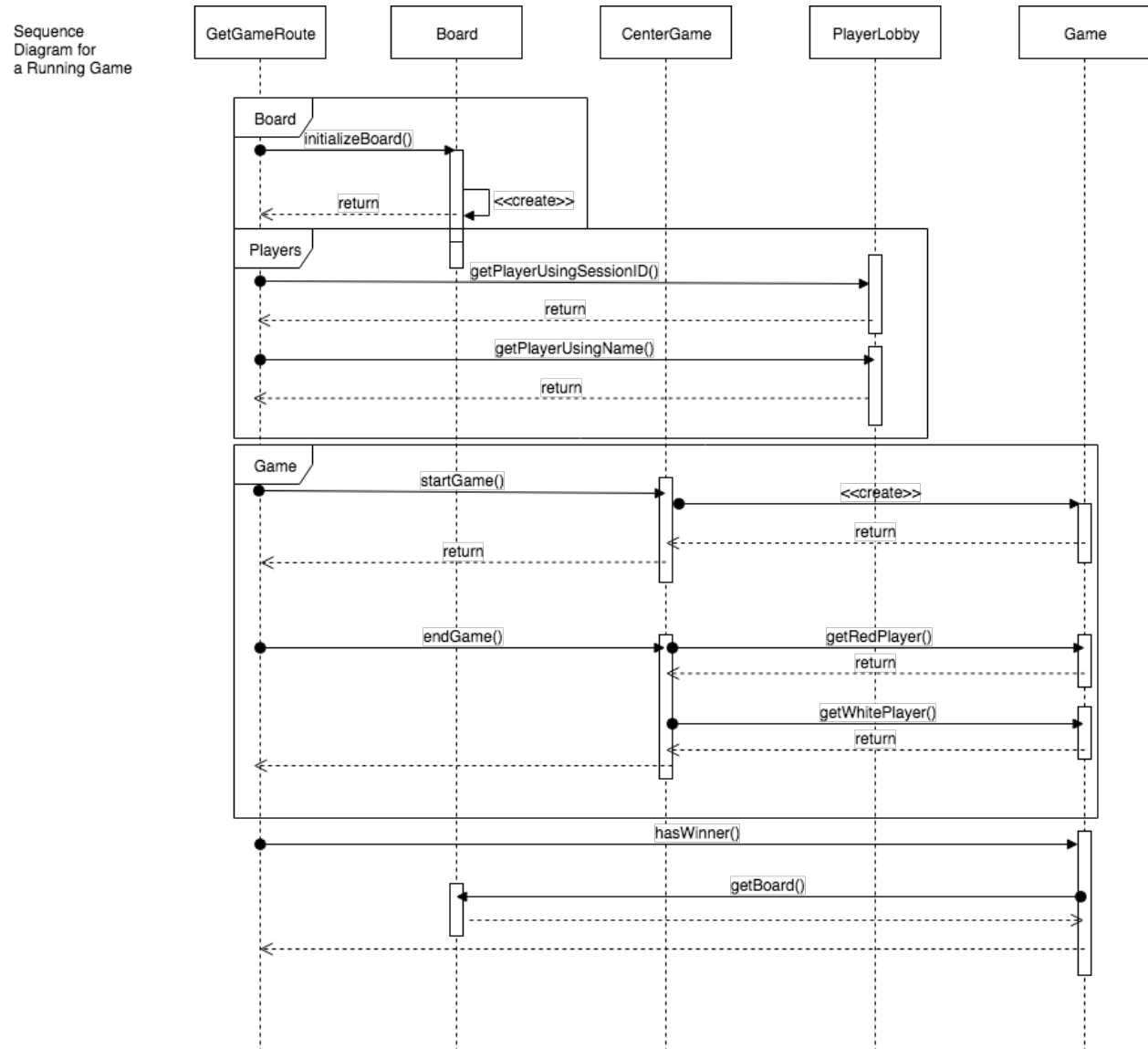
Class	CBO	DIT	LCOM	NOC	RFC	WMC
model.MoveRule	15	1	1	0	59	167
model.Game	18	1	3	0	38	33
model.Board	16	1	2	0	37	22
model.Row	9	1	3	0	19	15
appl.PlayerLobby	18	1	1	0	22	13
model.Player	16	1	4	0	13	12
model.Piece	16	1	1	0	10	10
ui.GetGameRoute	10	1	1	0	34	9
ui.PostSubmitTurn	17	1	1	0	37	8
ui.UserServices	10	1	5	0	24	8
model.BoardStack	7	1	1	0	14	8
appl.CenterGame	20	1	2	0	14	7
model.Space	8	1	2	0	8	7
model.Move	5	1	2	0	9	7
model.Position	5	1	2	0	8	7
model.Message	11	1	1	0	6	5
ui.PostSignInRoute	10	1	1	0	23	4
ui.GetHomeRoute	9	1	1	0	21	4
ui.PostCheckTurn	8	1	1	0	18	4
ui.PostValidateMove	14	1	1	0	24	3
ui.PostBackupMoveRoute	9	1	1	0	20	3
Application	7	1	1	0	19	3
ui.PostResignGameRoute	7	1	1	0	17	3
ui.WebServer	13	1	1	0	20	2
ui.GetSignInRoute	5	1	1	0	10	2
ui.GetSignOutRoute	5	1	1	0	17	2
ui.GetHelpRoute	2	1	1	0	11	2
model.Piece.COLOR	13	n/a	0	n/a	0	0
model.Message.Type	8	n/a	0	n/a	0	0
model.Piece.TYPE	7	n/a	0	n/a	0	0
ui.UserServices.UserNameResult	3	n/a	0	n/a	0	0

In terms of coupling between objects (CBO), the higher the number, the difficult it is to maintain and test. This means that testing would be harder since this particular class would need these specific objects to ensure if the class works. A recommendation to reduce the high coupling The depth of inheritance (DIT) for each class remains low at a value of 1, which also mean no children classes (NOC), which is also at value of 0. This should be the case for the route classes since there should not be any children subbed the main route classes of the application, and also for the application class. In terms of the model classes, there are not that many instances where the classes would have a children since the objects can mostly be comprised of their own entities. For lack cohesion of methods (LCOM), the analysis shows low values, with the highest being 5, indicating that there is a high cohesion between classes. No major redesign or re-implementation is needed for this part.

Dynamic Behaviors

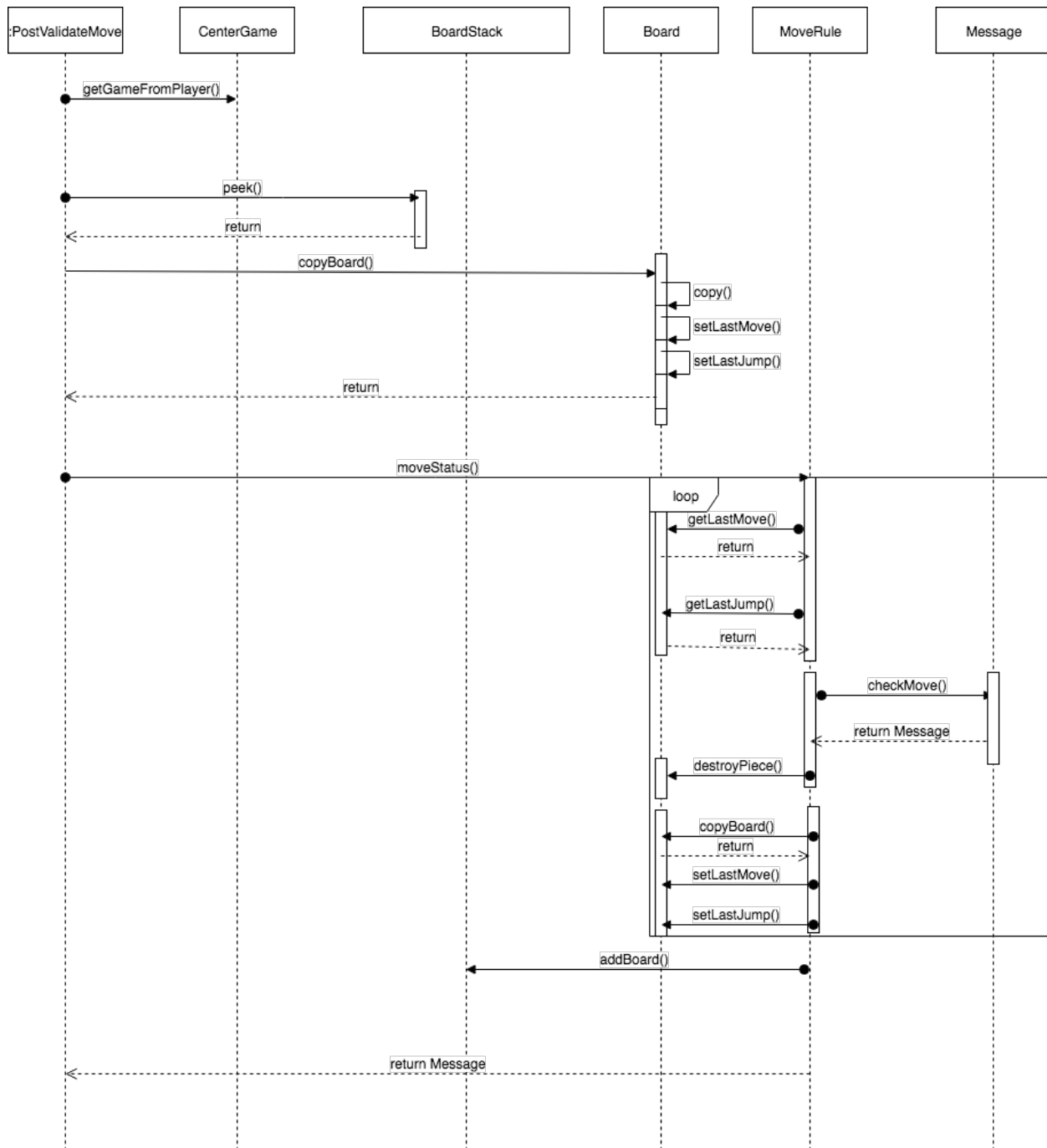
Few Dynamic Behaviors in our WebCheckers Game are as follows:

Sub-system: Current Game



This sequence diagram illustrates the flow of a current game from the system retrieving the players from the player list in the PlayerLobby class to playing a game and ending a game. The GetGameRoute object retrieves the player via sessionID and the opponent via their username and also initializes the board from the Board object. This collection is then passed to the Game object where a game will be instantiated and stored in the CenterGame object. During the game, the session will constantly check to see if the game has a winner or not and determine whether to exit the game.

Sub-system: Valid Move



####

This sequence diagram displays the interactions of objects when validating a player's move. When a move is made on the board, the PostValidateMove object will get the player who has moved a piece and latest board configuration from the BoardStack object. It will then get a copy of the board just in case if the player makes multiple moves in their single turn. Then, the board and the move is checked by the MoveRule object, which will get the last move made on the board and develop a Message made by the Message object and passed back to the PostValidateMove object to validate the move for the player.