P3 - Runtime Terrors

Darla Drenckhahn
Patrick Lee
Garrett Olsen
Daniel Stephenson
Falkyn Worm

Omega Chess Recap

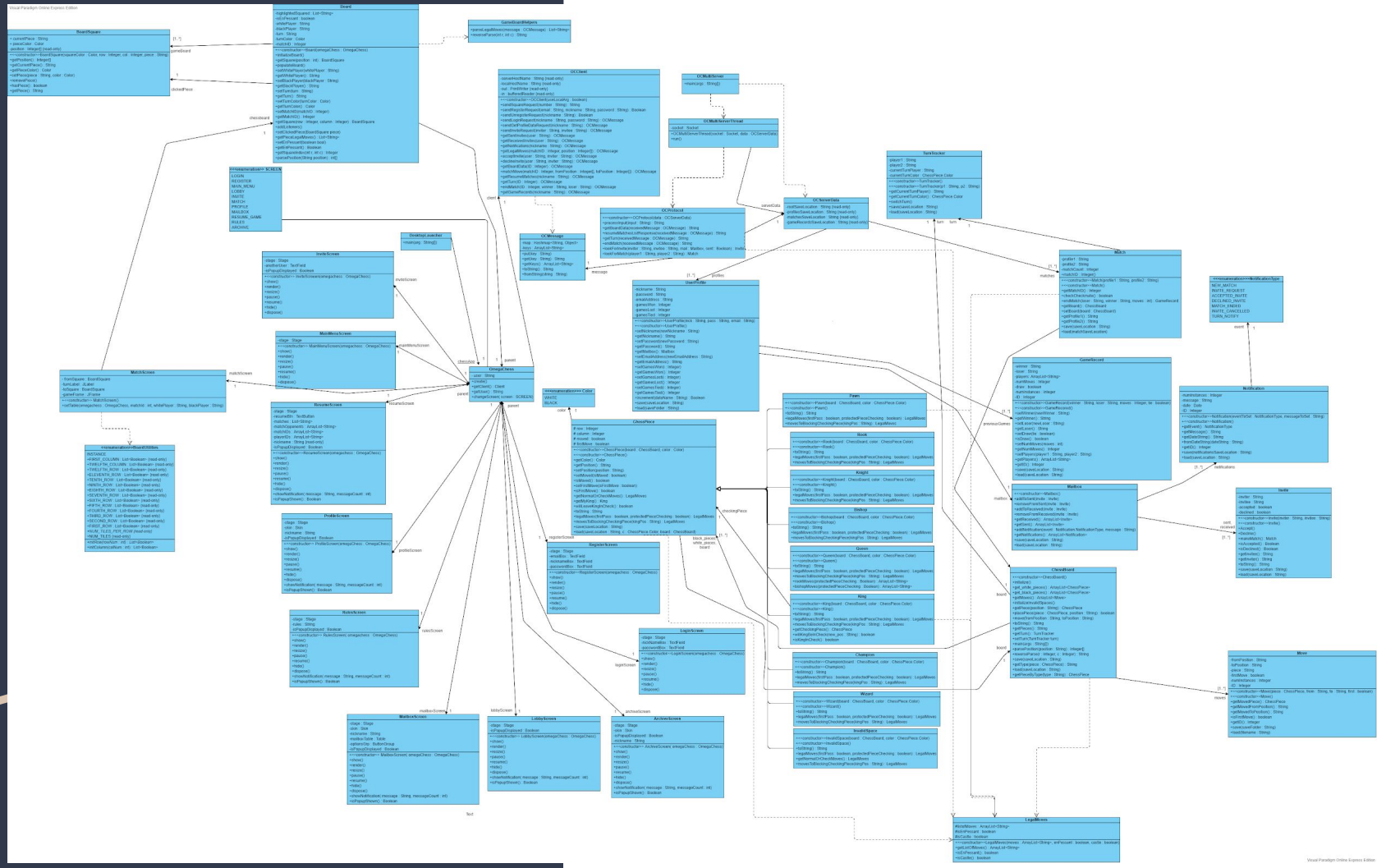
- Omega Chess is played on a 10x10 board and four “Wizard” squares at each corner.
- It has the same rules as chess except for the behavior of Pawns. The pawn can only move forward and in its initial position it can move one, two or three squares forward.
- There are two additional pieces in Omega Chess, the Champion and the Wizard.
- The Champion is like the Knight. It can move one square orthogonally forward, backward or to either side. It can also jump over these squares and also jump two square diagonally in all four directions.
- The Wizard can move one square diagonally in all four directions, or it can move as a Knight would, but three squares in one direction instead of two.

Processes/Practices /Tools

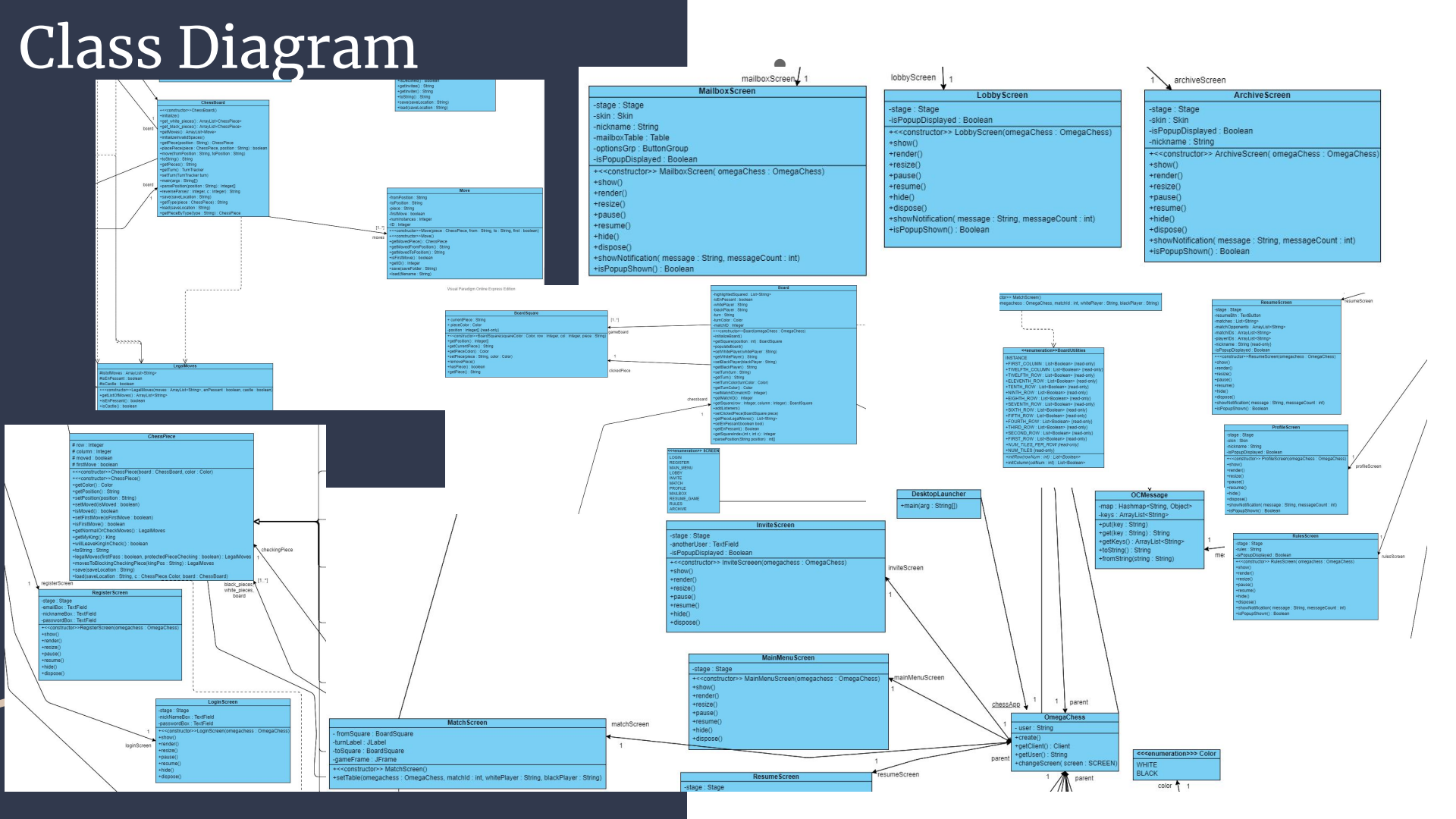
- Followed Agile Methodology with 2 week sprints, meetings with the PO, etc.
- GitHub Repository
- IntelliJ IDEA
- Gradle Build Tool
- Travis CI
- Junit Testing Framework
- LibGDX graphical library
- Java Swing graphical utilities

Design Patterns

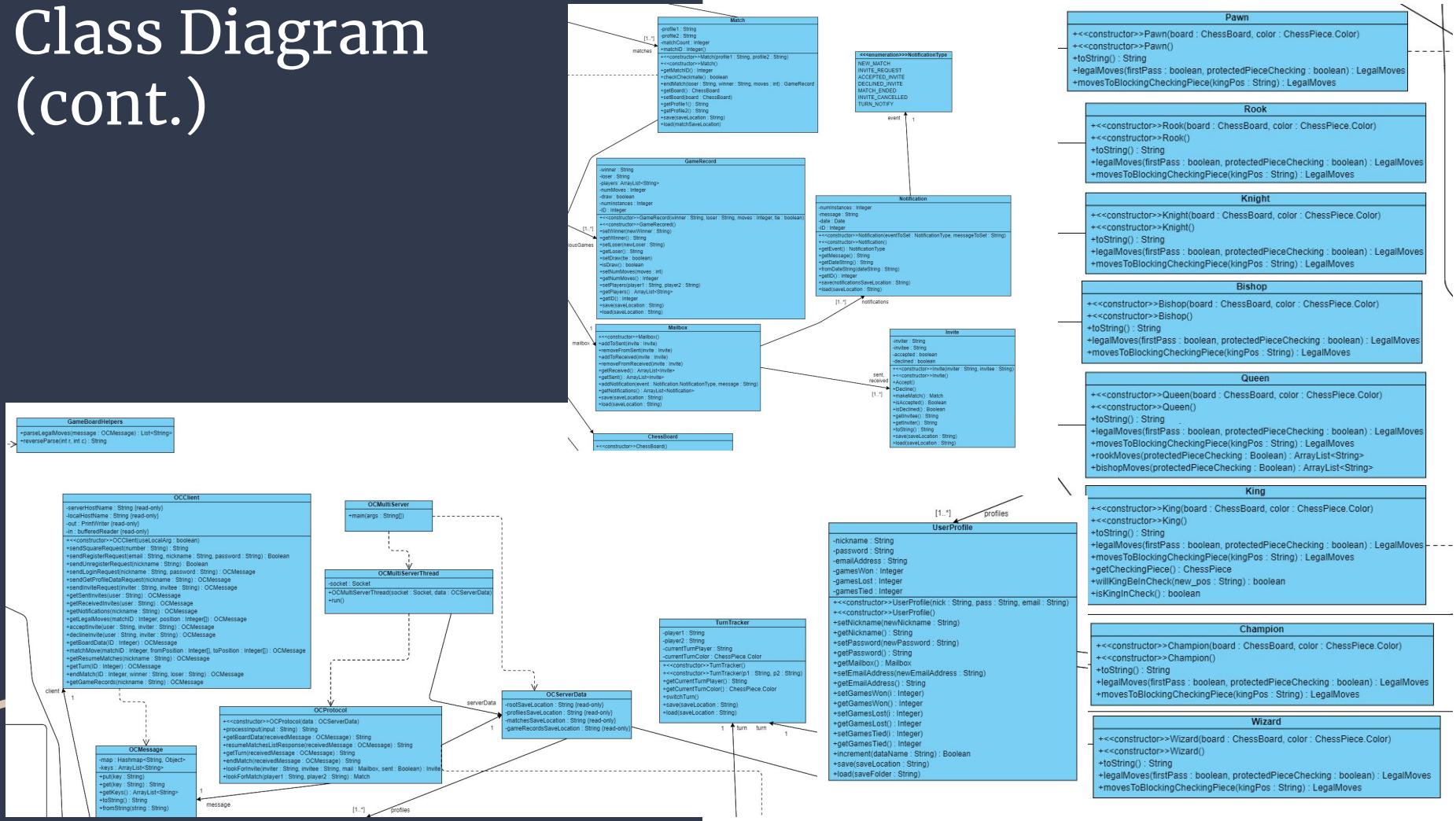
- We used the Command Pattern to communicate between our client and server. Our file “OCMessage.java” was used to encapsulate messages as objects that were then able to be read and processed by the server depending on the “process” variable that was set by the client. This pattern was a good fit because it allowed us to create multiple client requests and process them easily.


Visual Paradigm Online Express Edition
Board
BoardSquare
OCMultiServer
OCMultiServerThread
OCClient
OCProtocol
OCServerData
TurnTracker
GameBoardHelpers
DesktopLauncher
OCMessage
InviteScreen
MainMenuScreen
MatchScreen
ChessGame
ResumeScreen
ProfileScreen
RegisterScreen
RulesScreen
LoginScreen
MailboxScreen
LobbyScreen
ArchiveScreen
User
UserProfile
Match
GameRecord
Notification
Mailbox
Invite
ChessPiece
Pawn
Rook
Knight
Bishop
Queen
King
Champion
Wizard
InvalidSpace
ChessBoard
Move
LegalMoves
Visual Paradigm Online Express Edition

Class Diagram



Class Diagram (cont.)



Traceability Matrix

Classes / User Story #s	ChessBoard	Move	Bishop	Champion	ChessPiece	InvalidSpace	King	Knight	LegalMoves	Pawn	Queen	Rook	Wizard
1													
2													
3													
4													
5													
6													
7													
8													
9													
10													
11													
12													
13													
14													
15													
16													
17	X	X	X	X	X	X	X	X	X	X	X	X	X
18	X		X	X	X	X	X	X	X	X	X	X	X

Traceability Matrix (cont.)

GameRecord	Invite	Mailbox	Match	Notification	server/.../omegachess/server/OCMessage	OCMultiServer	OCMultiServerThread	OCProtocol
					X	X	X	X
			X					
	X		X					
	X							
	X							
			X					
X			X					
			X					
			X					
	X	X		X				
			X					
					X			X

Traceability Matrix (cont.)

OCServerData	TurnTracker	UserProfile	InviteScreen	LobbyScreen	LoginScreen	MainMenuScreen	MatchScreen	OCClient	core/.../game/OCMessage
X		X			X	X		X	X
							X		
	X		X						
							X		
		X							
		X							
	X								
	X								
				X					
							X		
							X		
X	X						X		

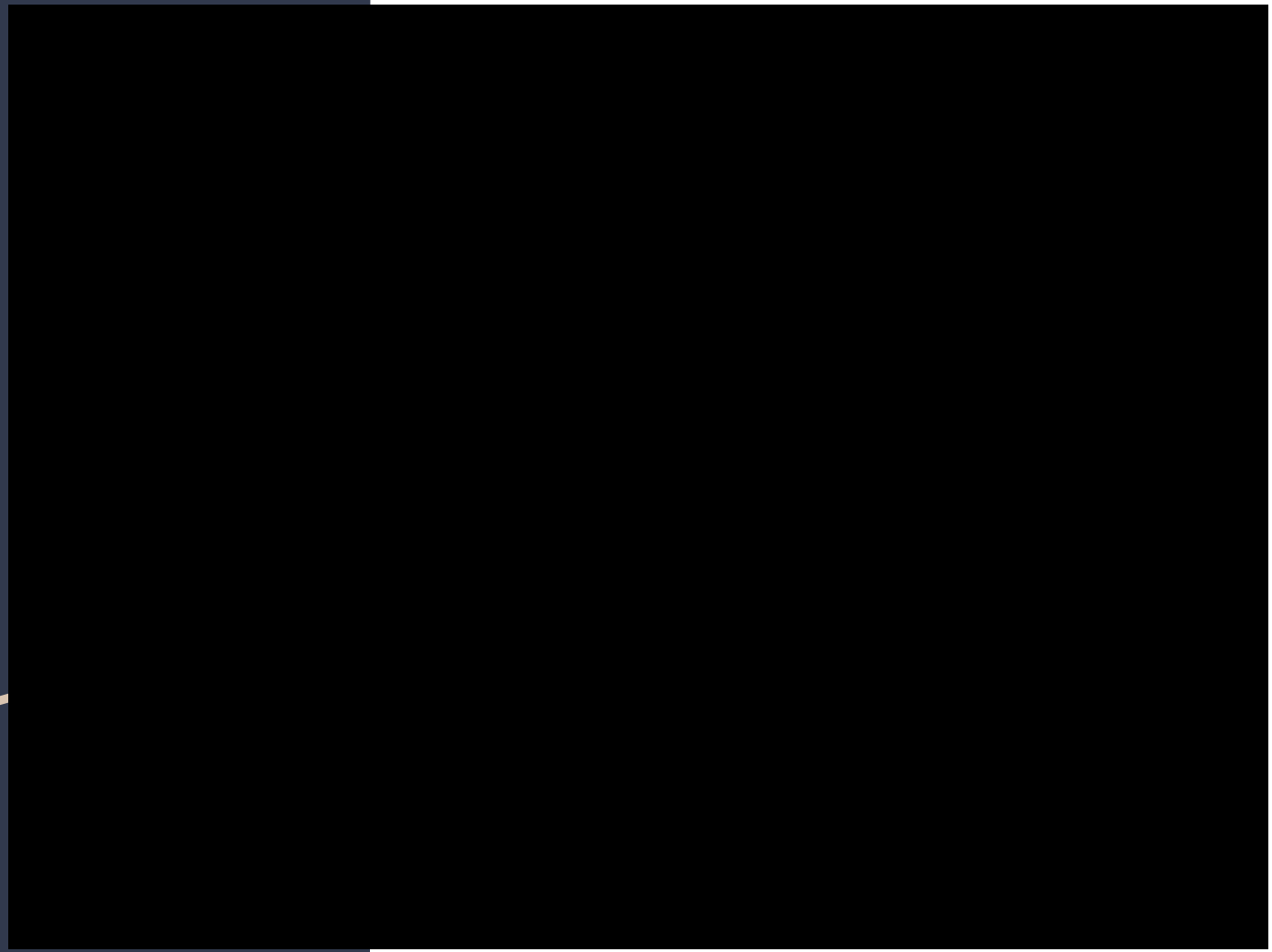
Traceability Matrix (cont.)

OmegaChess	ProfileScreen	RegisterScreen	BoardSquare	ArchiveScreen	MailboxScreen	ResumeScreen	RulesScreen
X		X					
					X		
						X	
				X			
	X						
						X	
				X			
					X		
							X
			X				
			X				

Demo

Shows:

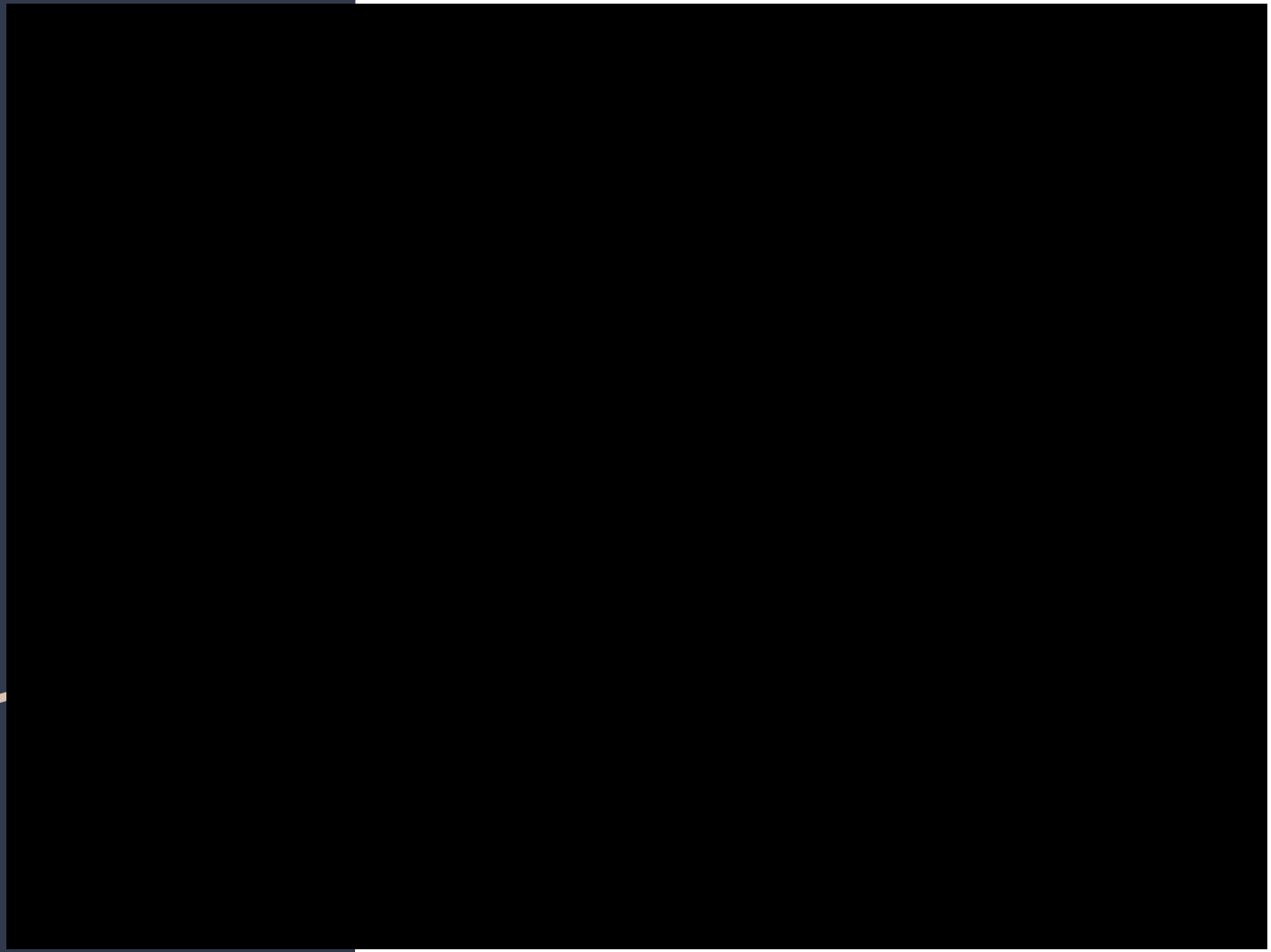
- Registering
- Inviting
- Getting a notification
- Accepting an invitation
- Viewing Profile
- Viewing Mailbox
- Viewing Current Games
- Moving Pieces
- Forfeiting
- Logging out



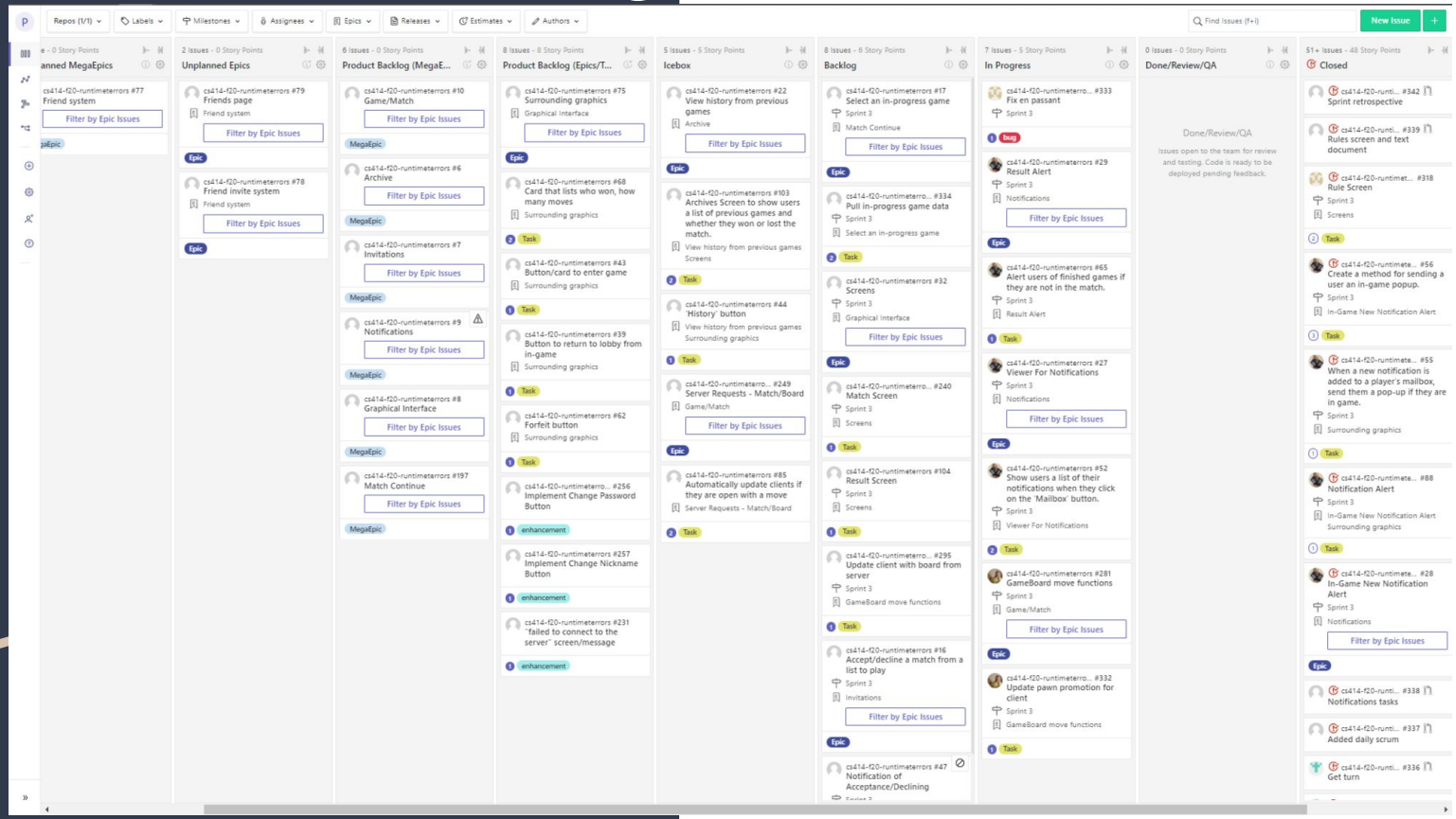
Demo Pt2

Shows:

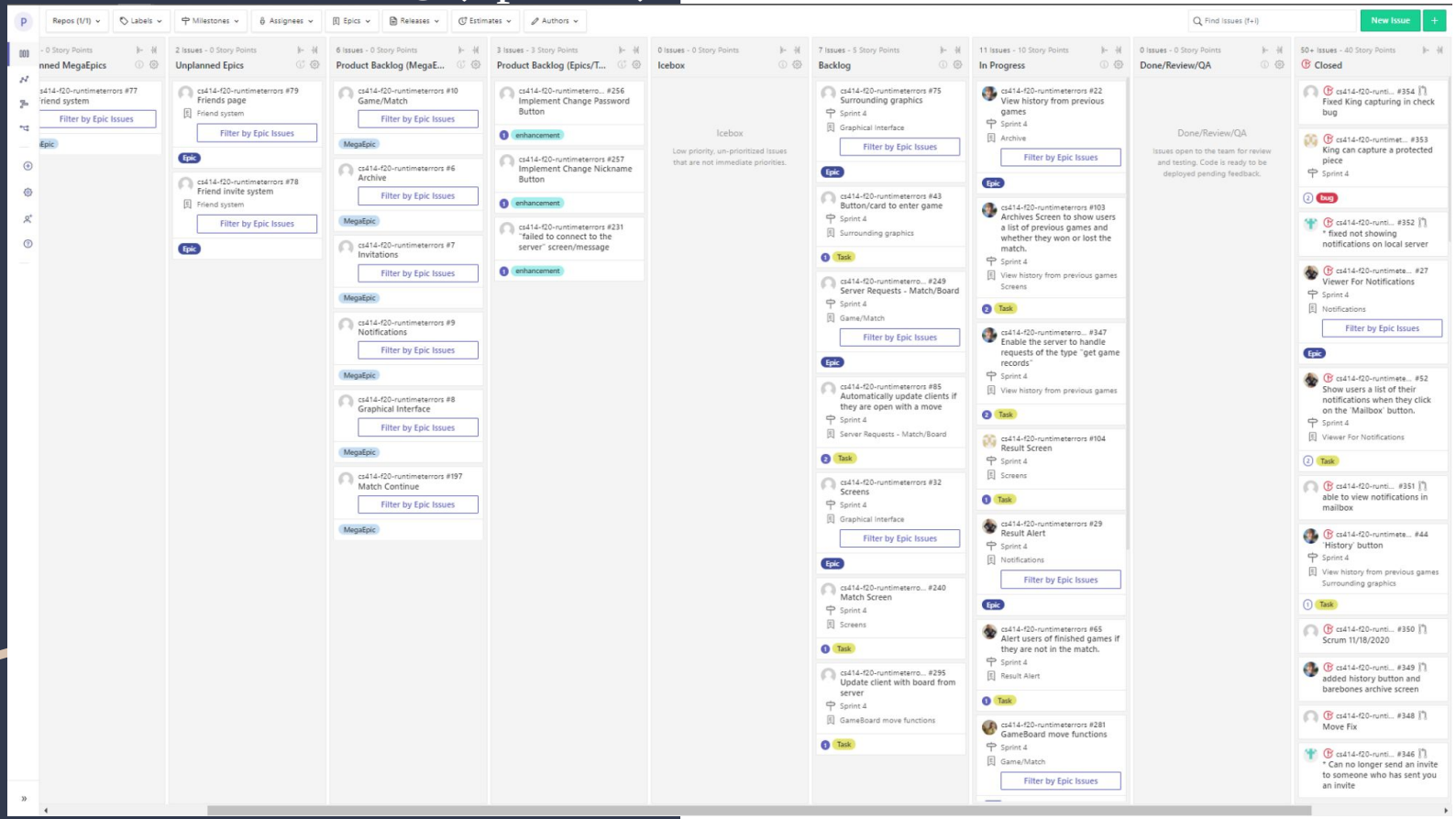
- Logging in
- Viewing Current Games
- Moving Pieces
- Viewing Profile
- Viewing Archive
- Unregister



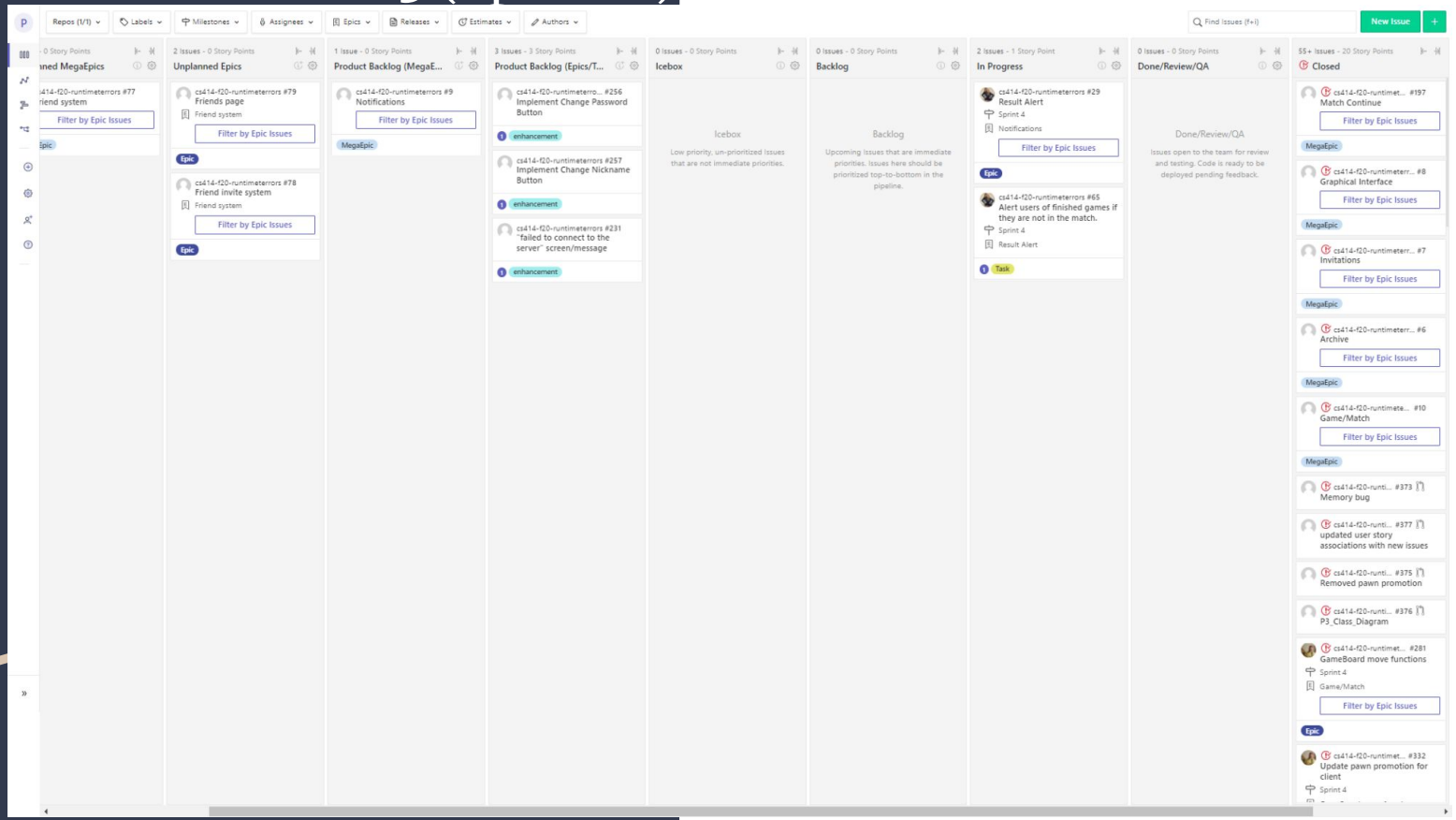
Kanban Board Start of P3



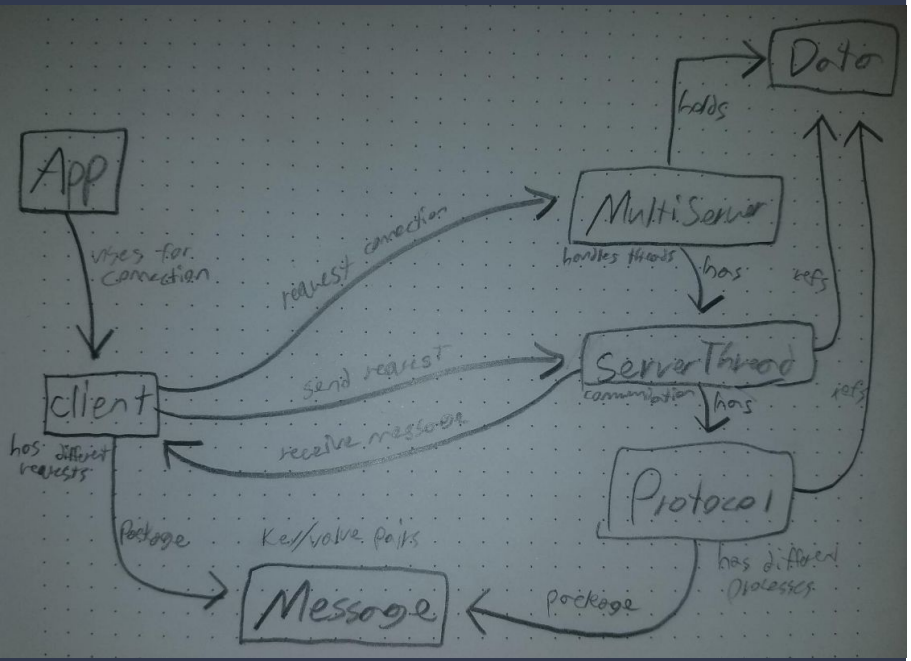
Kanban Board Middle of P3 (Sprint 1)



Kanban Board End of P3 (Sprint 2)



Server Setup



The **MultiServer** was made using java sockets. It waits for new connections to pair with a new **ServerThread**.

The **App** uses a **Client** class as its interface to the **MultiServer** and calls request methods based on user input.

The **Client** establishes a connection with the **MultiServer** and gets assigned a **ServerThread**. It has requests that it can send to the **ServerThread** on behalf of the **App**.

The **ServerThread** reads the content of requests and sends responses back to the **Client** according to the **Protocol**.

The **Protocol** accesses the **Data** in order to fulfill requests.

Both the **Client** and the **Protocol** package messages to be sent using a custom **Message** class, which is essentially a String-to-String HashMap that is capable of being turned into a string and back.

Current Supported Requests

- Register
- Unregister
- Login
- Get Profile Data
- Send Invite
- Get Sent Invites
- Get Received Invites
- Get Notifications
- Invite Response
- Get Board Data
- Get Legal Moves
- Match Move
- Get In-Progress Matches
- Get Turn
- Get Game Records
- End Match
- Check Checkmate
- Check Forfeit

Completed User Stories (for P3)

#6 "As a user, I can participate in multiple games at the same time."

#7 "As a user, I can forfeit from any game at any time."

#9 "As a user, I can view history statistics from previous games."

#11 "As a user, I can make the first move if I create the match."

#12 "As a user, I can return to a saved game."

#13 "As a user, I can see the end result of a game I participate in."

#16 "As a user, I will be able to quit the game using a Quit button."

#18. "As a user, when I choose to move a piece the board will show where I can move the piece."

Incomplete User Stories

#10 “As a user, I can view a user profile if registered.” (Would Have)

#14 “As a user, I will be alerted when a user accepts/declines an invitation, when it is my turn, or a game I am participating in is over.” (Could Have)

User Story Acceptance Criteria

(for completed user stories)

#6 As a user, I can participate in multiple games at the same time.

a) Test that a user can have multiple games happening at the same time (pass)

#7 As a user, I can forfeit from any game at any time.

a) Test that a user can hit the forfeit button and the game is ended (pass)
b) Test that if a user tries to forfeit, a confirmation window is shown asking the user if they are sure they want to forfeit (pass)
c) Test that when a user forfeits a game, it shows that they lost a game in their match history (pass)

#9 As a user, I can view history statistics from previous games.

a) Test that after a game is finished, user can go to profile and history and view the correct statistics from that game (pass)
b) Test that order of history is oldest first (pass)

#11 As a user, I can make the first move if I create the match.

a) Test that if a user creates the match, they get to make the first move (pass)

User Story Acceptance Criteria (continued)

#12 As a user, I can return to a saved game.

a) Test that after a user can hit a back button to return to the lobby (pass)
b) Test that if a user hits back to exit the game and then clicks the game to open it again, it returns with the saved most recent data (pass)

#13 As a user, I can see the end result of a game I participate in.

a) Test that if I am currently in the game, a banner/end result screen of some sort pops up with the results (pass)

#16 As a user, I will be able to quit the game using a Quit button.

a) Test that the Quit button will quit from a game (pass)
b) Test that a confirmation popup appears asking the user if they are sure they want to quit and exit this game (pass)

#18 As a user, when I choose to move a piece the board will show where I can move the piece.

a) Test that when it is a users turn, they can click on one of their pieces and it will show the possible spots they can move it to (pass)
b) Test that if the user selects an opponent's piece, it will not show where that piece could move (pass)

CRC Cards

OmegaChess	
• Has user String for the current user • Has Enumeration SCREEN • Is the core of our program, it launches the client which processes requests from the different screens and sends them to the server.	

OCClient	
• Has objects serverHostName, localHostName, hostName, portNumber, socket, out, and in	• OmegaChess • OCMessage

OCMessage	
• Has objects map and keys • Converts a request and response to and from a string for Server-Client communication	• OCClient • OCProtocol

OCMultiServer	
• Listens for new connections from clients • If the connection is successful it passes the client to a new OCMultiServerThread	

OCMultiServerThread	
• Uses OCProtocol to process requests from the client	• OCMultiServer

OCProtocol	
• Processes requests sent from the client and returns an OCMessage as a String to the client	• OCMultiServerThread

OCServerData	
• Has objects profiles, matches, and previousGames • Stores variables on the server to allow users to come and go as they please	• OCMultiServer • OCMultiServerThread • OCProtocol

MainMenuScreen	
• Has objects stage, loginBtn, registerBtn, and exitBtn • Choose a screen from login or register	• OmegaChess • OCClient

CRC Cards (continued)

InviteScreen	
• Has objects stage, submit, home, opponentName, anotherUser • Sends invite request to the server from the client	• OmegaChess • OCClient

LoginScreen	
• Has objects stage, loginBtn, backBtn, nickNameBox, passwordBox • Sends login request to the server • Logs the user in and takes them to the lobby screen	• OmegaChess • OCClient

RegisterScreen	
• Has objects stage, registerBtn, backBtn, emailBox, nickNameBox, and passwordBox • Checks password strength • Sends a register request to the server then returns to MainMenuScreen	• OmegaChess • OCClient

RulesScreen	
• Has rules • Shows the rules of Omegachess to the user	• OmegaChess

LobbyScreen	
• Has objects stage, createGameBtnm, resumeGameBtn, logoutBtn, profileBtn, and exitBtn • Gives the player the option to view their profile, send an invite to another user, and resume a match they are in	• OmegaChess

ProfileScreen	
• Has objects, stage, unregisterBtn, lobbyBtn, changePwBtn, changeNicknameBtn, skin, style_label, nickname • Displays the User's information • If you are viewing your own profile option to change nickname and password • View the users win/loss/tie ratio • Sends a request to the server to get the user's data	• OmegaChess • OCClient

MailboxScreen	
• Has objects outBox, inBox, outCount, inCount, • Has a button to show the user their outBox/inBox • When viewing outBox cancel an invite • When viewing inBox accept/decline an invite	• OmegaChess • OCClient

ArchivesScreen	
• Has objects stage and nickname of the current user • Show previous games that the user has won and lost	• OmegaChess • OCClient

CRC Cards (continued)

BoardSquare	
• Has currentPiece • Contains the sprite of the currentPiece on it • Highlights if it is contained in the legalMoves of a piece	• MatchScreen

GameRecord	
• Has objects winner, loser, players, numMoves, and draw • Creates an object that holds the results of a game • Stores it in the OCServerdata/UserProfile	• OCServerData

UserProfile	
• Has objects nickname, password, emailAddress, gamesWon, gamesLost, gamesTied, playedWith, and mailbox • Stores the data of a user	• OCServerData

Mailbox	
• Has objects sent and received • Holds lists of sent and received invites between other users	• UserProfile

Invite	
• Has objects inviter, invitee, accepted, and declined • Send an invite to a user you deem by their username	• Mailbox • OCMultiServer

Notification	
• Has objects event, message, date • Notifies the user if they have any change in their mailbox • Notify if you have a game request • Notify if someone accepts/declines game invite	• Mailbox • OCMultiServer

CRC Cards (continued)

Match	
• Has objects board, player1Pieces, player2Pieces, and turn • The match is the instance of a game between two players. • I keeps track of the current board state • When a game is ended it sends a notification to the server regarding if it was ended by forfeit or Checkmate	• OCServerData

TurnTracker	
• Has objects player1, player2, currentTurnPlayer, and currentTurnColor • Keeps track of whose turn it is	• Match

ChessBoard	
• Has objects moves, black_pieces, white_pieces, and board • Allows a piece to move based on its list of legalMoves	• Match • ChessPiece

Abstract **ChessPiece** Pawn, Rook, Knight, Bishop, Queen, King, Champion, Wizard	
• Has objects row, column, and moved • Has Enumeration Color • Returns a list of Moves as a piece's legalMoves	• ChessBoard • Move

LegalMoves	
• Has objects listofMoves, isEnPessant, and isCastle • This is a helper class for holding the legal moves of a piece	• ChessPiece

MatchScreen	
• Has objects chessBoard, fromSquare, toSquare, and turnLabel • Show the users a game board to play against another user • Shows the users the name of the user of whose current turn it is • When a game is ended it shows a pop-up regarding what happens • Players take turns, preventing the users from moving pieces when it is not their turn. • Players are able to return to the lobby or forfeit a match	• OmegaChess • OCClient

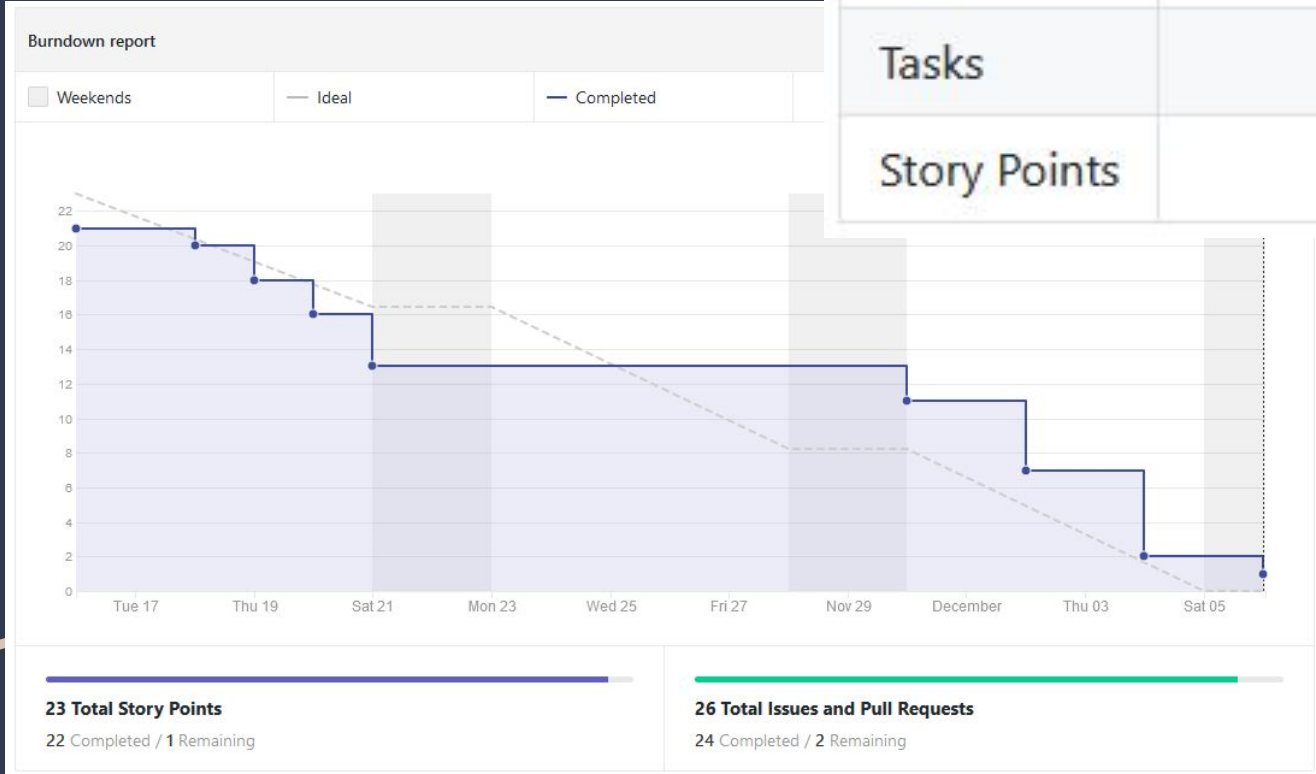
Sprint 3
11/2 to 11/16

Statistic	# Planned	# Completed
Epics	7	2
Tasks	45	36
Story Points	73	62

Sprint 4

11/16 to 12/6

Statistic	# Planned	# Completed
Epics	8	7
Tasks	18	17
Story Points	23	22



Lessons Learned

- Thoroughly research the frameworks you want to use. Using the LIBGDX framework we encountered a memory leak that caused a crash. We ended up having to change that screen out for a Java Swing screen at the last minute which causes our screens to look different and not be cohesive.
- Diagramming the classes of your programs can be enlightening and show problems with coupling and cohesion.
- There are always improvements to be made and new design patterns and tools that can make your application better.
- Always meet with the PO to make sure everyone is on the same page and that they are happy.