Summary

Team: Blake Galbavy, Austin Metz, Galen Pogoncheff, Richard Poulson

Title: Checkers

Project Summary: **'**Checkers' is a application that allows users to play the classic checkers board game. User's of the application will have the ability to play checkers against other users on the network, as well as against an automated computer player (driven by artificial intelligence algorithms). Within the game, users can view and interact with a game board whose state reflects the moves of the players in real time. Users also have access to an in-game chat room with their opponent. User statistics, number of wins, number of losses, and total play time, will be recorded in a database. Standard users can access win/loss statistics on via a leaderboard, and administrators can view user play time as they wish.

Project Requirements

User Requirements			
ID	**Description**	**System Responsibilities**	**Priority**
US-01	As a user, I want to be able to host a game so that I can initialized a game to play with another user.	Create a server for the host. Initialize a game of checkers that is ready to play once a second user joins the game.	High
US-02	As a user, I want to be able to join an already initialized game so that I can play a game against other users on the network.	Given a user specified network address, connect the user to the server that is being hosted at the specified address.	High
US-03	As a user, I want to be able to play a game of checkers against another user so that I can play against my friends.	Present both players with an GUI that they can interact with. Map the user's interactions to changes in the game state, and show these changes as they occur.	High
US-04	As a user, I want to be able to chat with my opponent so that we can communicate with each other without being in the same location.	Listen for text input from players of the game. When input is submitted by a player, output the text stream to the other player's screen.	High
US-05	As a user, I want the ability to play	Initialize a game of checkers for	High

	against an automated computer player so that if I cannot find another user to play against, I can still play a game of checkers.	the user with a automated computer player as their opponent. The computer player must use artificial intelligence algorithms make decisions based on the current state of the game. The computer players “knowledge base” must be update as the game state changes.	
US-06	As a user, I want to see a visual representation of the game being played so that I can easily see the current state of the game.	Users playing a game must be presented with a real-time GUI that depicts the current game state. This interface must be update after every move made in the game.	High
US-07	As an administrator, I want to be able to view the play time of users so that I can monitor the system’s use.	Maintain a database that records the play time of system users. User play times must be updated after every game they play. Administrators have the option to access this database from the main menu.	Low
US-08	As a new user, I want access to a “Help” page so that I can learn the rules of the game and use of the application.	Provide a “Help” button that once clicked, shows the user details and rules of the game of Checkers.	Med
US-09	As a frequent user, I want to be able to change the difficulty of the computer player so that I can continue to challenge my skills as I progress.	After a user selects the option to play against a computer player, provide the user with an option to select difficulty prior to game initialization. To enforce this difficulty, the artificial intelligent algorithms must have pre-tuned parameters corresponding to different difficulty levels.	Low
US-10	As a user, I want to be able to view my ranking in the system so that I can compare my proficiency with that of other users.	Maintain a database table with records of number of wins and losses for each user. Users have option from main menu to view a subset of this table.	Med

Business Requirements

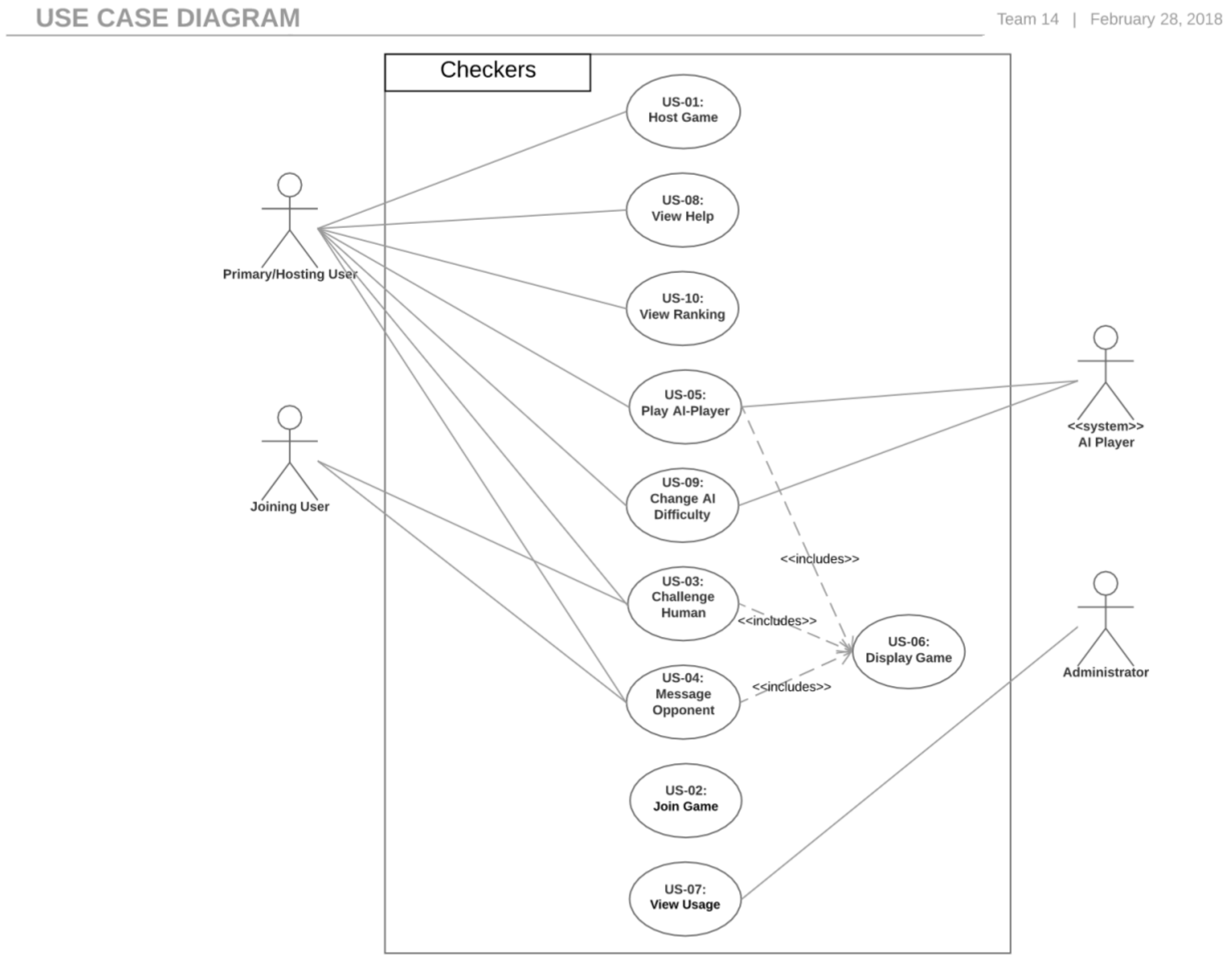
ID	Description	System Responsibilities	Priority
BR-01	No more than two users may connect to a single game.	Disallow any attempts to connect to join a game server that already has two players connected to it.	High

Functional Requirements			
ID	**Description**	**System Responsibilities**	**Priority**
FR-01	The results of each game (win or loss) are recorded for the users.	After each game is completed, update the database table that manages number of wins and losses for each user to reflect the outcome of the game.	High
FR-02	The total play time of a user is recorded.	After each game is completed, update each player’s play time in the system usage database table.	Med

Non-Functional Requirements			
ID	**Description**	**System Responsibilities**	**Priority**
NFR-01	Users need to log in with a username and password.	Enforce a user to log in when starting the application. Maintain a database with usernames and hashed passwords. Provide new users with capability of creating a new account.	High
NFR-02	Users can log in and play whenever they want.	No foreseeable additional system responsibilities.	High
NFR-03	Access to system usage statistics must be restricted to administrators only.	Only provide system administrators with the menu option of viewing system usage statistics.	Med

Use Cases

Use Case Overview:



Use Case Documents:

The two major workflows that we decided to diagram are the workflows of use cases UC-03 and UC-07. Use case documents for UC-01 and UC-02 are simply provided for reference, as they are preconditions to UC-03.

Use Case ID:	UC-01 (Reference)
Use Case Name:	Host Game
Description:	A user can host a game of checkers at a user specified network address. (Corresponds to user requirement US-01)

Actors:	User		
Pre-conditions:	User has internet access.		
Post-conditions:	Hosting user is logged in. A Checkers game board has been initialized for the hosting user. The hosting user is viewing the game board. A different user has the ability to join this game.		
Frequency of Use:	Daily to Multiple times per day		
Flow of Events:		**Actor Action**	**System Response**
	1.	Click on “Play Human” button in main menu.	Present user with a new screen with two buttons, a “Host Game” button and a “Join Game” button.
	2.	Click on “Host Game” button.	Prompt user to input the network address where they wish to host the game.
	3.	Enter a network address and click “Submit” button.	Initialize a new game of Checkers
	4.		Present user with Checkers game UI.
Variations:			
Exceptions:	3. Error Message: A game is already being hosted at that address.		

Developer Notes:	User may not know how to find a suitable network address to host a game on. Future iterations may require automatically generating a proper address for the user and then displaying the generated address.

Use Case ID:	UC-02 (Reference)
Use Case Name:	Join Game
Description:	A user can join an already initialized game that is being hosted. (Corresponds to user requirement US-02)

Actors:	Joining user, hosting user		
Pre-conditions:	Joining user is logged in. A game is being hosted by the hosting user (see UC-01). Joining user has internet access.		
Post-conditions:	The joining user has been connected to the game being hosted by the hosting user. Both the joining user and hosting user are viewing the game board. A game of Checkers game be played between the joining and hosting users.		
Frequency of Use:	Daily to Multiple times per day		
Flow of Events:		**Actor Action**	**System Response**
	1.	Joining user clicks on “Play Human” button in main menu.	Present joining user with a new screen with two buttons, a “Host Game” button and a “Join Game” button.
	2.	Joining user clicks on “Join Game” button.	Prompt joining user to input the network address of the host game that they wish to join.
	3.	Enter a network address and click “Submit” button.	Connect joining user to the game that is being hosted by the hosting user
	4.		Present joining user with Checkers game UI.

Variations:			
Exceptions:	3. Error Message: No game is currently being hosted at that address.		
Developer Notes:			

Use Case ID:	UC-03
Use Case Name:	Challenge Human
Description:	Two separate users can play a game of checkers against a each other over a network connection. (Corresponds to user requirement US-03)

Actors:	Hosting User, Joining User		
Pre-conditions:	Hosting user is hosting a game (see UC-01). Joining user has joined this hosted game (see UC-02).		
Post-conditions:	The hosting user and joining user have played a game of Checkers against each other.		
Frequency of Use:	Daily to Multiple times per day		
Flow of Events:		**Actor Action**	**System Response**
	1.	Hosting user makes a move.	Update current state of the game board. Update game UI for both hosting user and joining user.
	2.	Joining user makes a move.	Update current state of the game board. Update game UI for both hosting user and joining user.
	3.	*Steps 1 and 2 are repeated until one of the users has won the game.*	
Variations:	3.	Joining user quits game.	Redirect joining user to main menu. Reset game board.

	3.	Hosting user quits game	Kill server and redirect both users to main menu.
Exceptions:			
Developer Notes:	Once a game has been completed, statistics of the game (winner, loser, and play time) must be updated for each user (functional requirements FR-01 and FR-02).		

Use Case ID:	UC-07
Use Case Name:	View Usage
Description:	Administrative users can view the play-time statistics of system users. This statistic can be shown as an aggregate of system usage over a specified time period for all users for a specific user. (Corresponds to user requirement US-07)

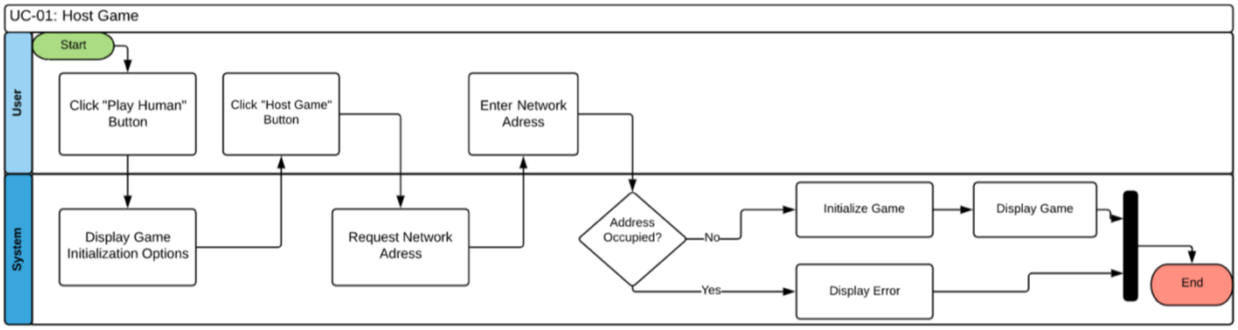
Actors:	Admin		
Pre-conditions:	Admin user logged in with an administrative user account.		
Post-conditions:	Admin is presented with user-play time statistics.		
Frequency of Use:	Daily-Weekly		
Flow of Events:		**Actor Action**	**System Response**
	1.	Click “View System Usage” button from main menu.	Prompt admin to input a time range that they would like to view system usage for.
	2.	Inputs time range.	Redirect admin to screen with two buttons, a “View Total System Usage” button and a “View Usage for Specific User” button.
	3.	Click “View Total System Usage” button.	Display total system usage statistics over the time range originally specified by the admin in Actor Action 2.

Variations:	3.	Click “View Usage for Specific User” button.	Prompt admin to input the username of the user they would like to view total usage for.
	4.	Input username of a user of the system.	Display system usage statistics for specified user over the time range originally specified by the admin in Actor Action 2.
Exceptions:	Error Message: Input time range invalid.		
	Error Message: Specified user does not exist.		
Developer Notes:	“View System Usage” button in main menu must be invisible for standard users.		

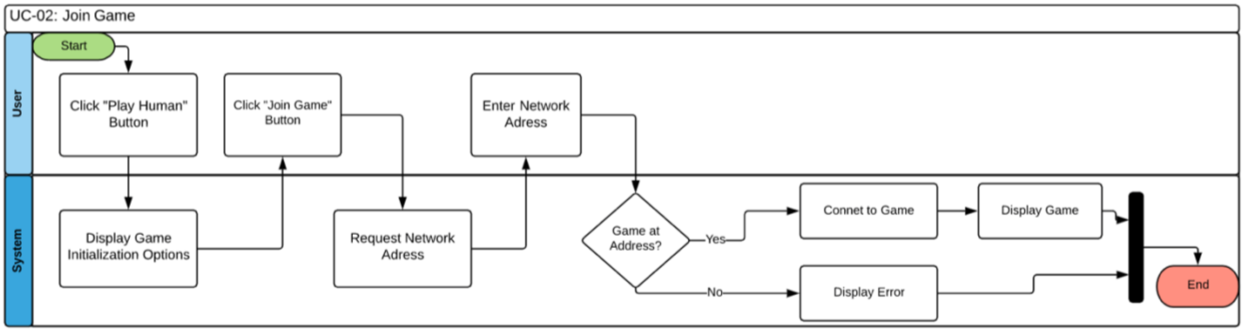
Activity Diagrams

The two major workflows that we decided to diagram are the workflows of use cases UC-03 and UC-07. Activity diagrams for UC-01 and UC-02 are simply provided for reference, as they are preconditions to UC-03.

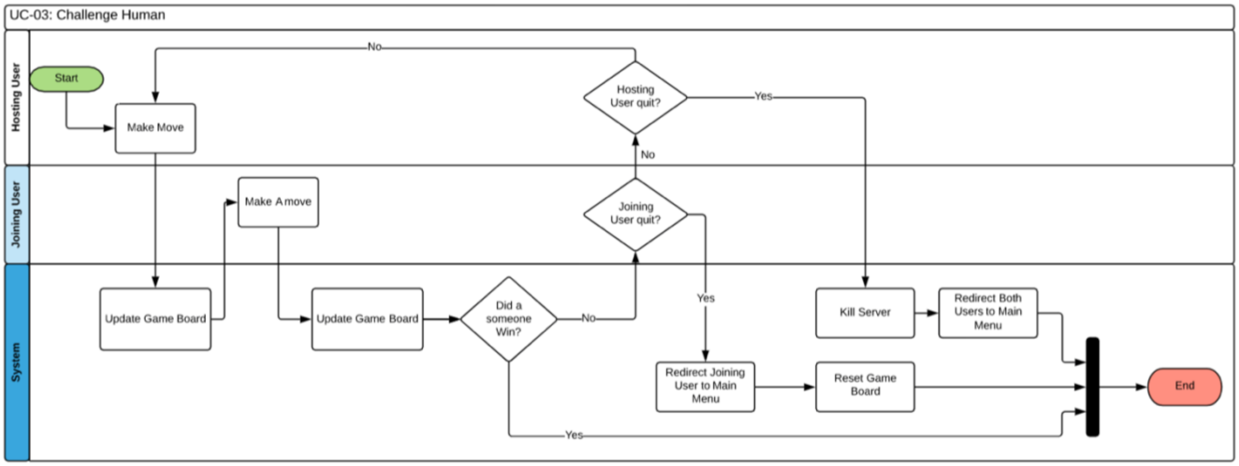
For Reference:
Requirement ID: US-01
Use Case ID: UC-01
Use Case Name: Host Game
Use Case Description: A user can host a game of checkers at a user specified network address.



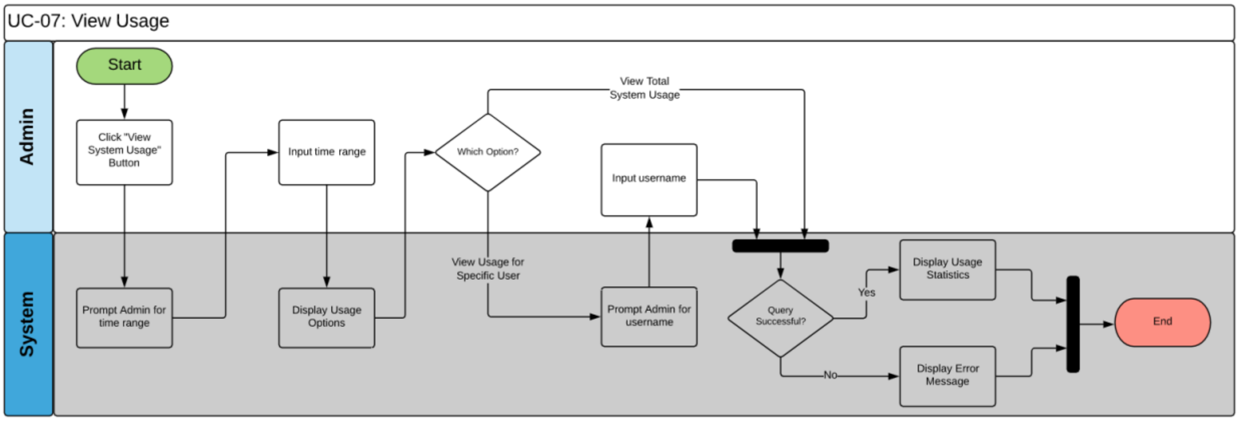
For Reference:
Requirement ID: US-02
Use Case ID: UC-02
Use Case Name: Join Game
Use Case Description: A user can join an already initialized game that is being hosted.



Requirement ID: US-03
Use Case ID: UC-03
Use Case Name: Challenge Human
Use Case Description: Two separate users can play a game of checkers against a each other over a network connection.

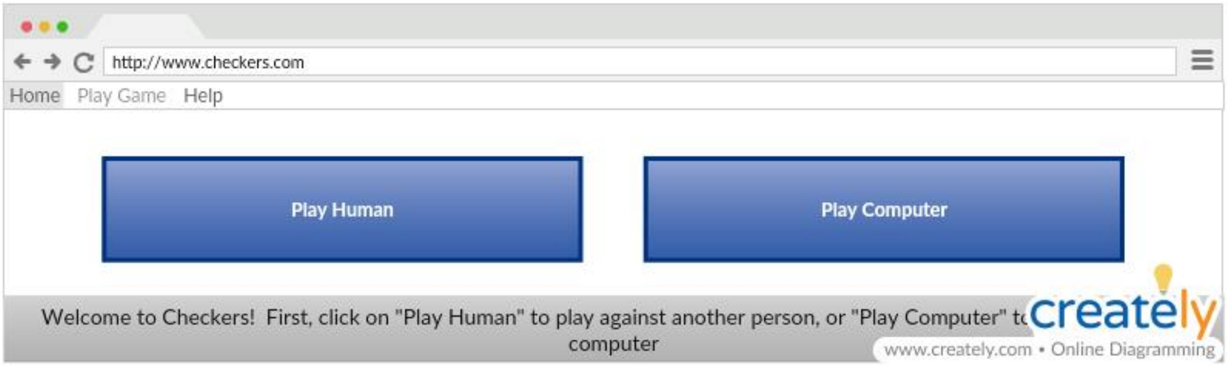


Requirement ID: US-07
Use Case ID: UC-07
Use Case Name: View Usage
Use Case Description: Administrative users can view the play-time statistics of system users. This statistic can be shown as an aggregate of system usage over a specified time period for all users for a specific user.



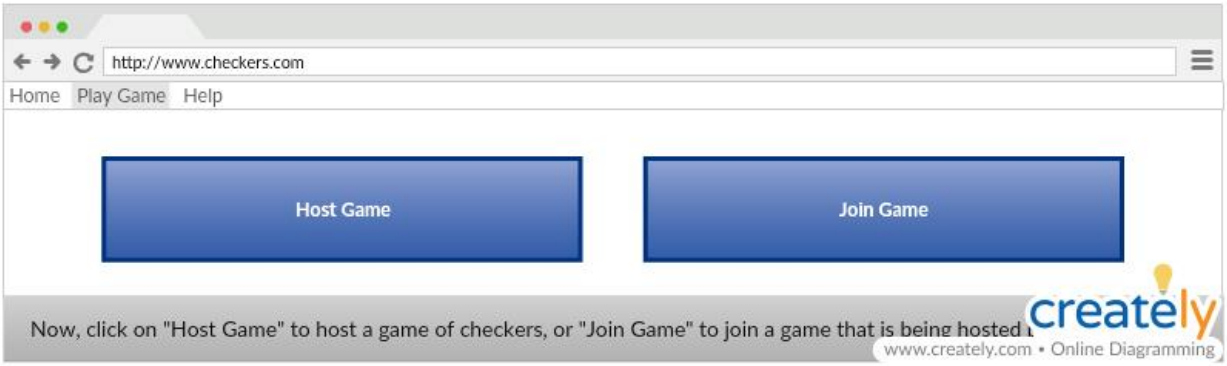
UI Mockups

User options to play a game of Checkers against a human user or an artificial intelligence computer player:



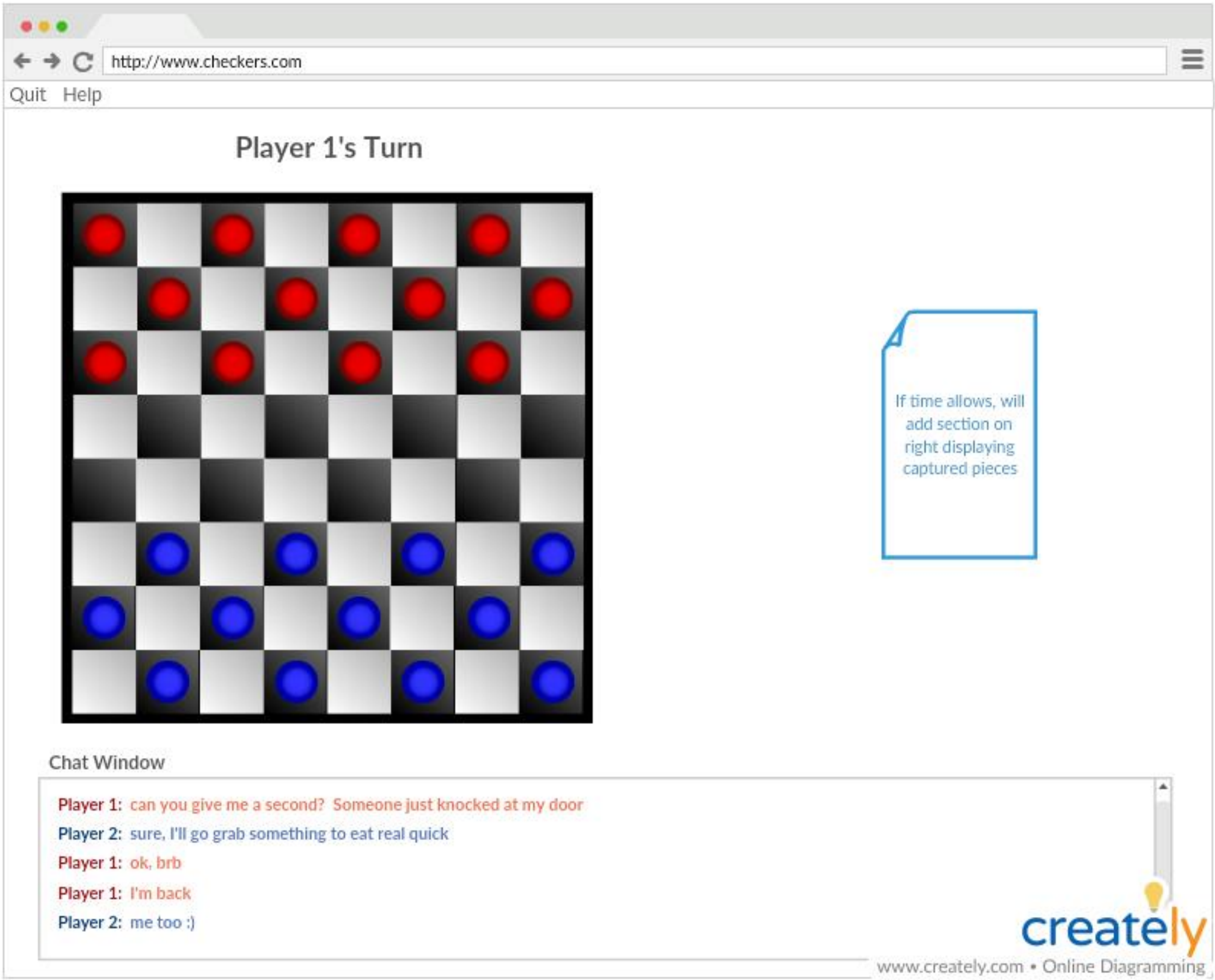
Welcome to Checkers! First, click on "Play Human" to play against another person, or "Play Computer" t computer

If user wants to play a game against another user, they have the option to host a game, or join a game:



Now, click on "Host Game" to host a game of checkers, or "Join Game" to join a game that is being hosted

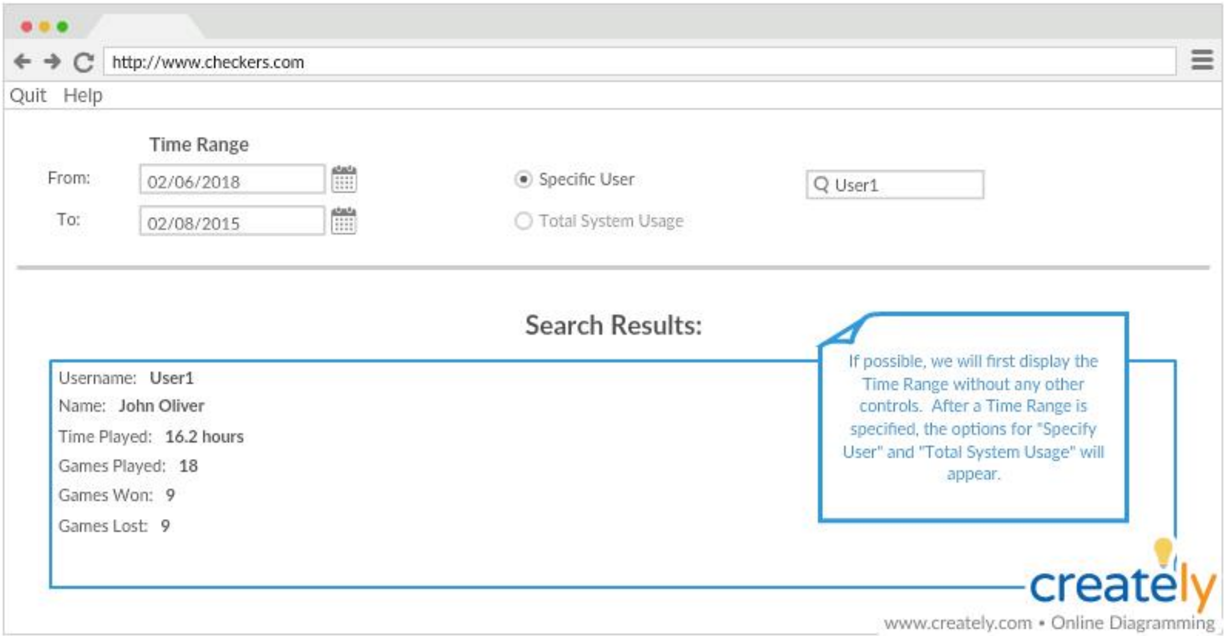
In-game view of real-time Checkers board. Below the game board is a chat room for the players on the game server:



http://www.checkers.com
Quit
Help
Player 1's Turn
If time allows, will add section on right displaying captured pieces
Chat Window
creately
www.creately.com • Online Diagramming

Player 1: can you give me a second? Someone just knocked at my door
Player 2: sure, I'll go grab something to eat real quick
Player 1: ok, brb
Player 1: I'm back
Player 2: me too :)

Administrative users have the option to view the usage of the system. They can view the system usage for all users or for specific users:



Time Range

From: 02/06/2018

To: 02/08/2015

● Specific User

○ Total System Usage

User1

Search Results:

Username: User1

Name: John Oliver

Time Played: 16.2 hours

Games Played: 18

Games Won: 9

Games Lost: 9

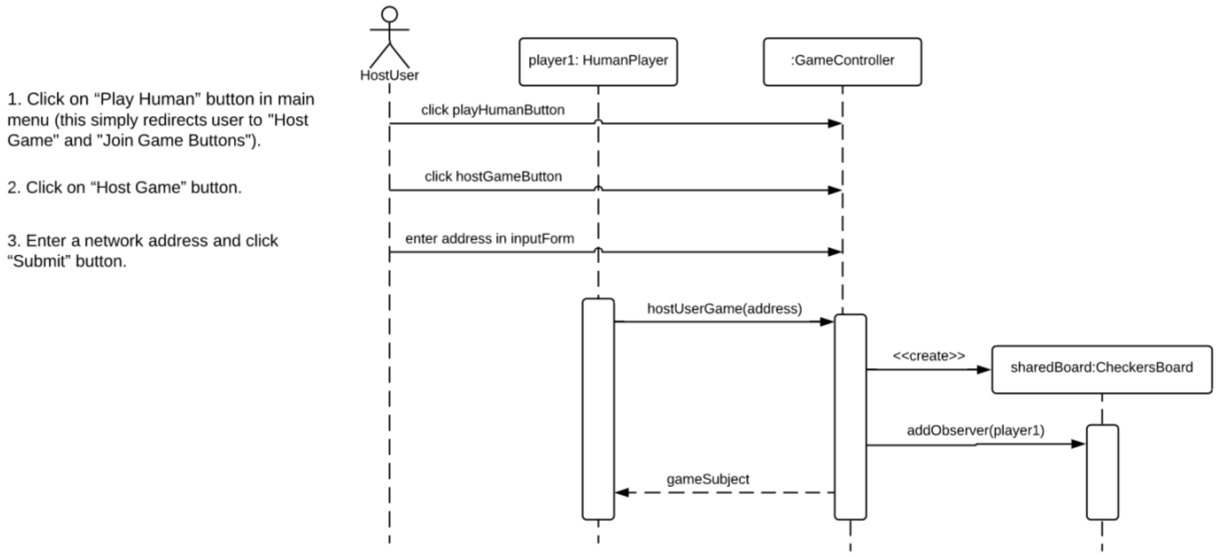
If possible, we will first display the Time Range without any other controls. After a Time Range is specified, the options for "Specify User" and "Total System Usage" will appear.

User Interactions

The two major workflows that we decided to diagram are the workflows of use cases UC-03 and UC-07. Sequence diagrams for UC-01 and UC-02 are simply provided for reference, as they are preconditions to UC-03.

For Reference:
Requirement ID: US-01
Use Case ID: UC-01
Use Case Description: A user can host a game of checkers at a user specified network address.

Use Case UC-01: Host Game



1. Click on “Play Human” button in main menu (this simply redirects user to "Host Game" and "Join Game Buttons").

2. Click on “Host Game” button.

3. Enter a network address and click “Submit” button.

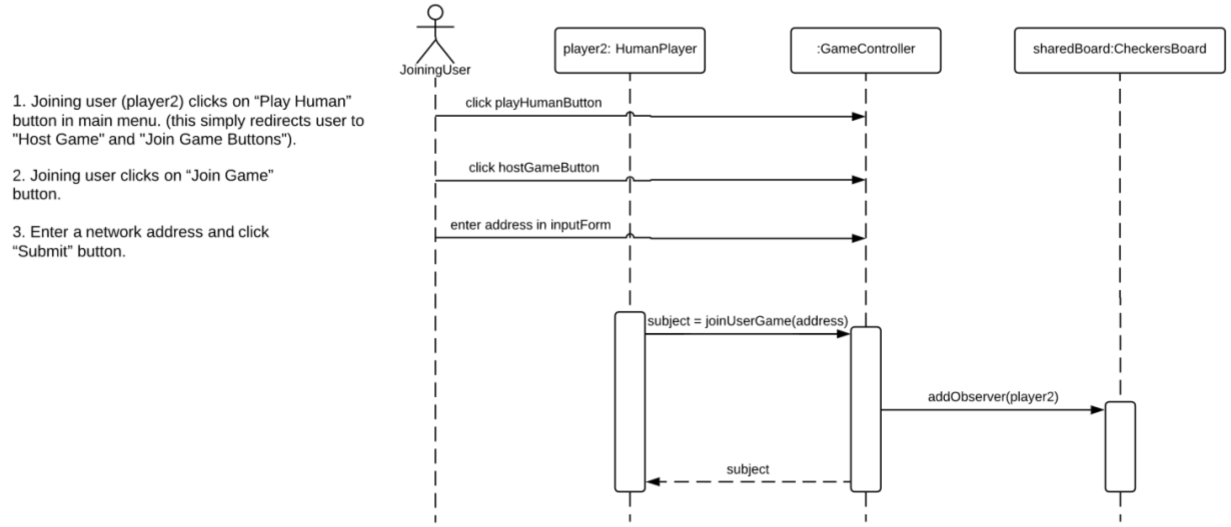
For Reference:

Requirement ID: US-02

Use Case ID: UC-02

Use Case Description: A user can join an already initialized game that is being hosted.

Use Case UC-02: Join Game



1. Joining user (player2) clicks on “Play Human” button in main menu. (this simply redirects user to "Host Game" and "Join Game Buttons").

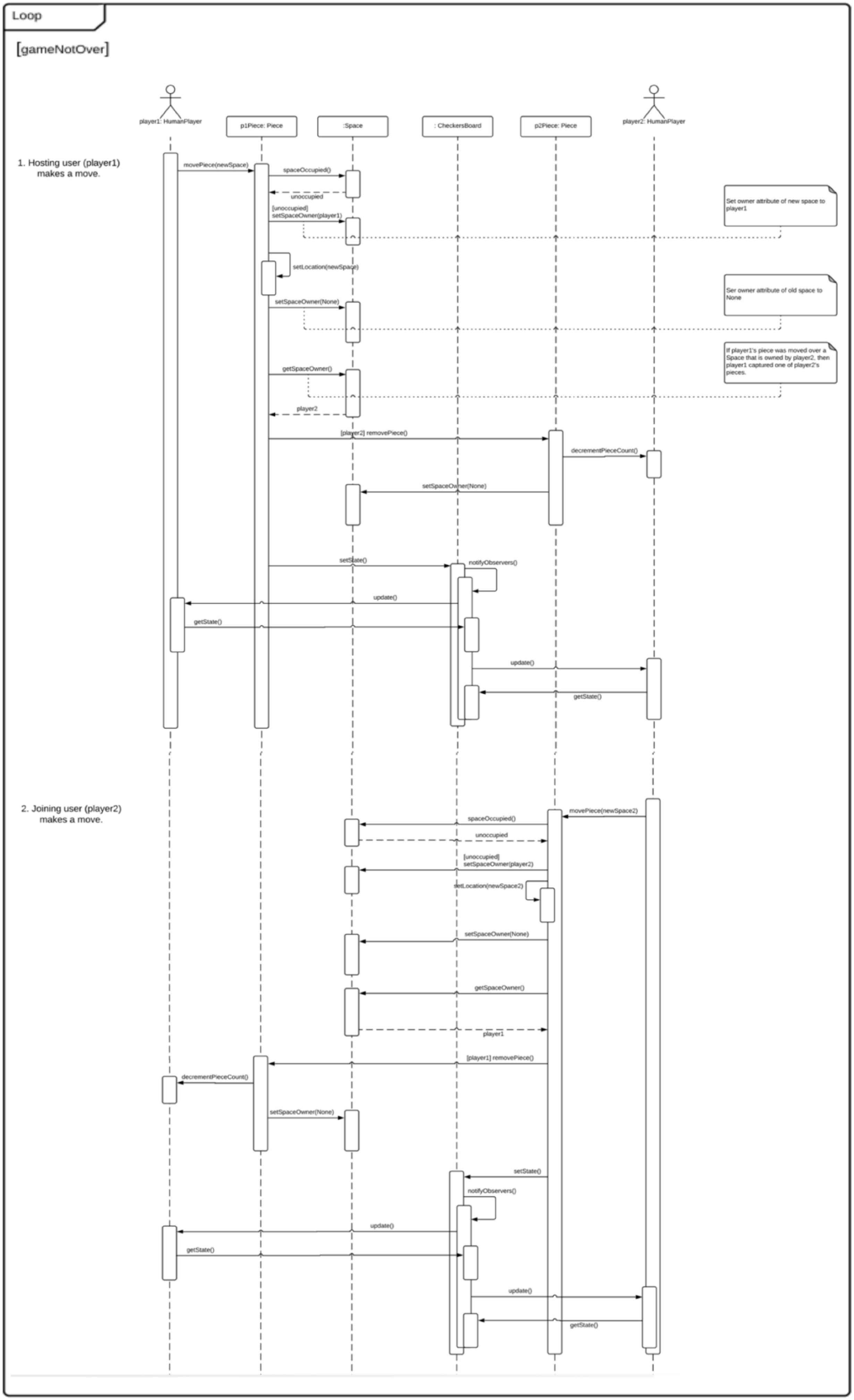
2. Joining user clicks on “Join Game” button.

3. Enter a network address and click “Submit” button.

Requirement ID: US-03
Use Case ID: UC-03
Use Case Description: Two separate users can play a game of checkers against a each other over a network connection.

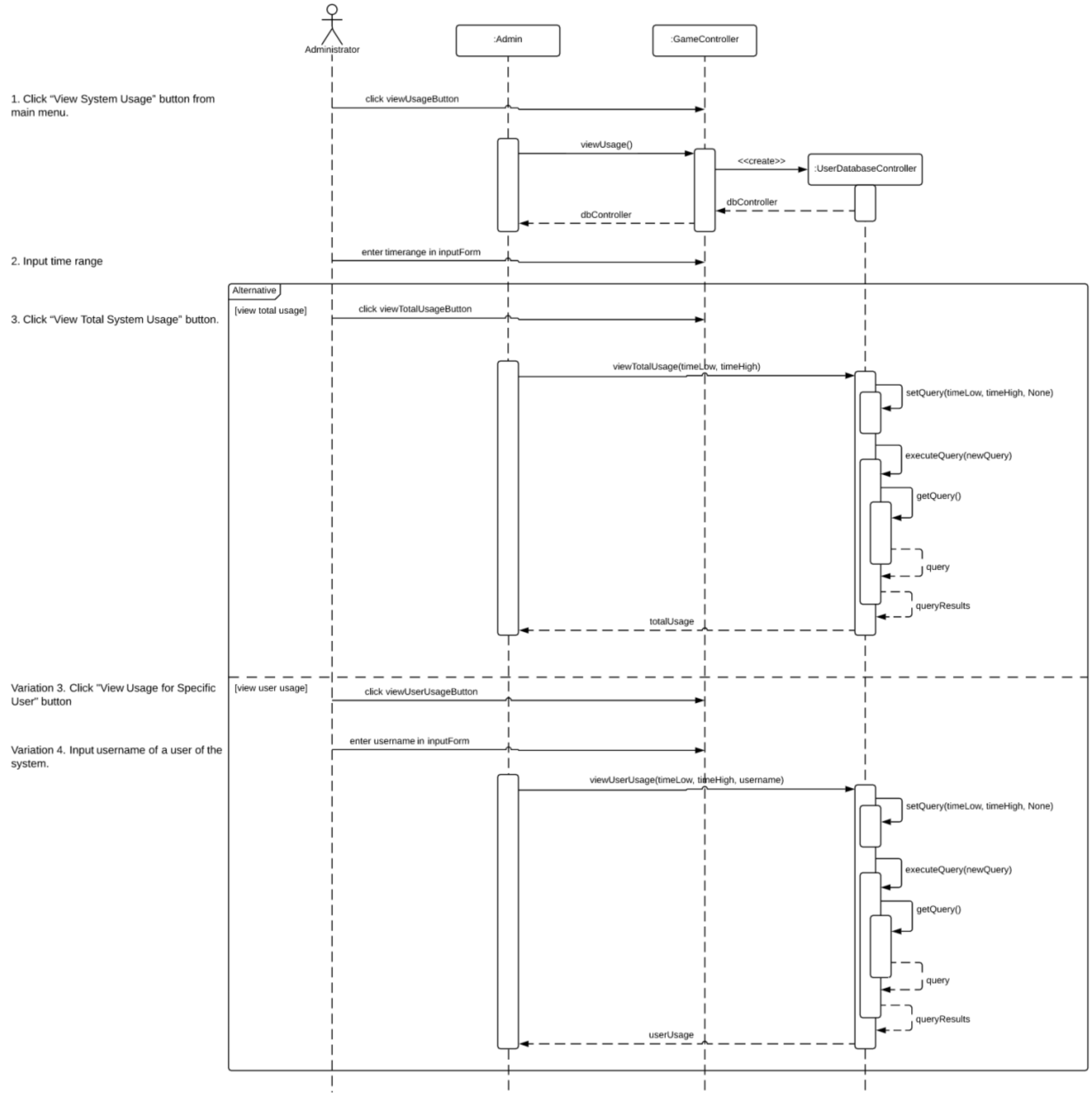
Sequence Diagram on Following Page

Use Case UC-03: Challenge Human

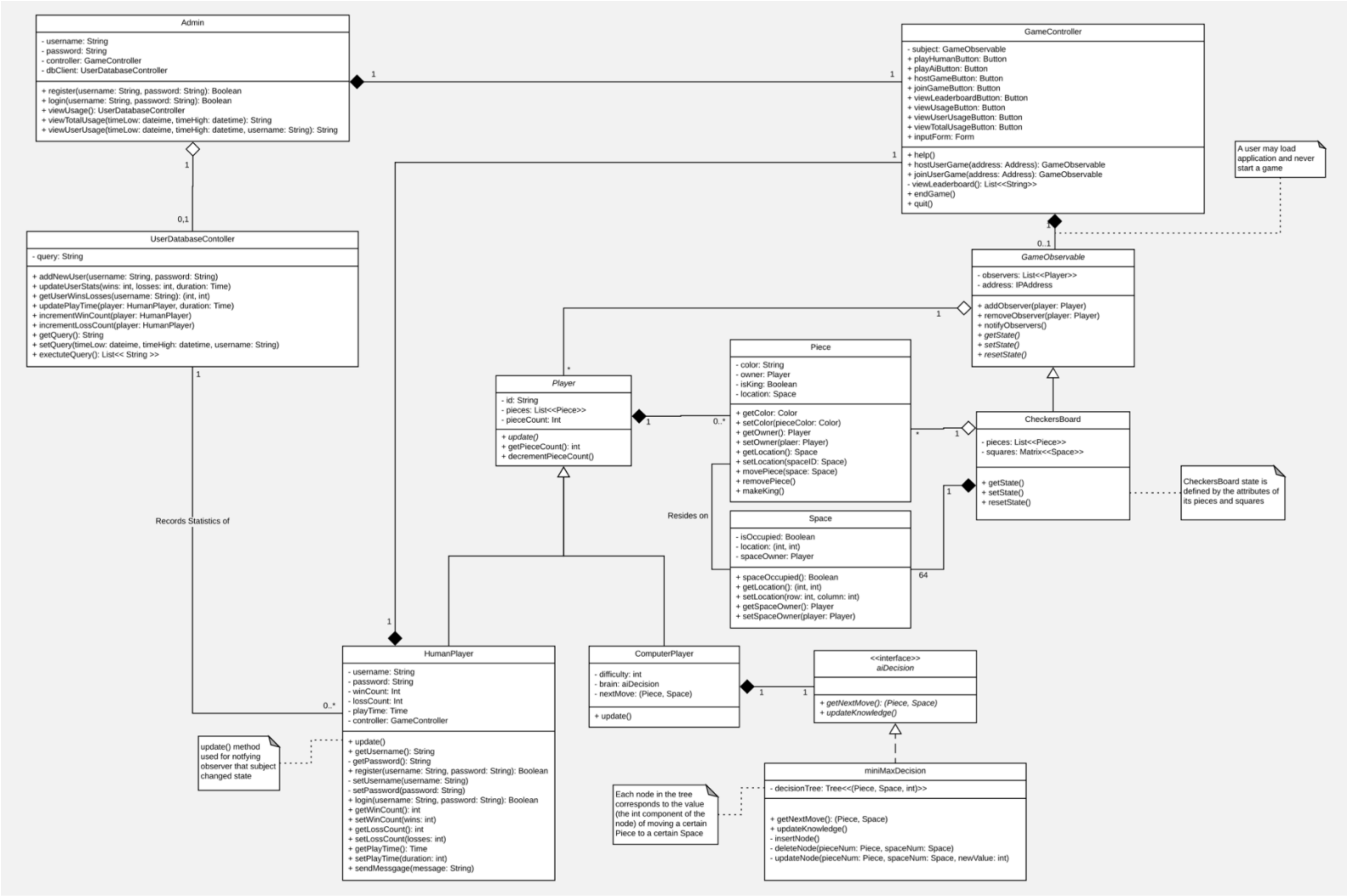


Requirement ID: US-07
Use Case ID: UC-07
Use Case Description: Administrative users can view the play-time statistics of system users. This statistic can be shown as an aggregate of system usage over a specified time period for all users for a specific user.

<u>Use Case UC-07: View Usage</u>


Administrator
:Admin
:GameController
1. Click "View System Usage" button from main menu.
click viewUsageButton
viewUsage()
<<create>>
:UserDatabaseController
dbController
dbController
2. Input time range
enter timerange in inputForm
Alternative
[view total usage]
3. Click "View Total System Usage" button.
click viewTotalUsageButton
viewTotalUsage(timeLow, timeHigh)
setQuery(timeLow, timeHigh, None)
executeQuery(newQuery)
getQuery()
query
queryResults
totalUsage
[view user usage]
Variation 3. Click "View Usage for Specific User" button
click viewUserUsageButton
Variation 4. Input username of a user of the system.
enter username in inputForm
viewUserUsage(timeLow, timeHigh, username)
setQuery(timeLow, timeHigh, None)
executeQuery(newQuery)
getQuery()
query
queryResults
userUsage

Class Diagram


Admin
- username: String
- password: String
- controller: GameController
- dbClient: UserDatabaseController
+ register(username: String, password: String): Boolean
+ login(username: String, password: String): Boolean
+ viewUsage(): UserDatabaseController
+ viewTotalUsage(timeLow: dateime, timeHigh: datetime): String
+ viewUserUsage(timeLow: dateime, timeHigh: datetime, username: String): String
1
1
1
0,1
UserDatabaseContoller
- query: String
+ addNewUser(username: String, password: String)
+ updateUserStats(wins: int, losses: int, duration: Time)
+ getUserWinsLosses(username: String): (int, int)
+ updatePlayTime(player: HumanPlayer, duration: Time)
+ incrementWinCount(player: HumanPlayer)
+ incrementLossCount(player: HumanPlayer)
+ getQuery(): String
+ setQuery(timeLow: dateime, timeHigh: datetime, username: String)
+ exectuteQuery(): List&lt;&lt; String &gt;&gt;
1
Records Statistics of
0..*
GameController
- subject: GameObservable
+ playHumanButton: Button
+ playAiButton: Button
+ hostGameButton: Button
+ joinGameButton: Button
+ viewLeaderboardButton: Button
+ viewUsageButton: Button
+ viewUserUsageButton: Button
+ viewTotalUsageButton: Button
+ inputForm: Form
+ help()
+ hostUserGame(address: Address): GameObservable
+ joinUserGame(address: Address): GameObservable
- viewLeaderboard(): List<<String>>
+ endGame()
+ quit()
1
A user may load application and never start a game
1
0..1
GameObservable
- observers: List<<Player>>
- address: IPAddress
+ addObserver(player: Player)
+ removeObserver(player: Player)
+ notifyObservers()
+ getState()
+ setState()
+ resetState()
1
*
Player
- id: String
- pieces: List<<Piece>>
- pieceCount: Int
+ update()
+ getPieceCount(): int
+ decrementPieceCount()
1
0..*
Piece
- color: String
- owner: Player
- isKing: Boolean
- location: Space
+ getColor: Color
+ setColor(pieceColor: Color)
+ getOwner(): Player
+ setOwner(plaer: Player)
+ getLocation(): Space
+ setLocation(spaceID: Space)
+ movePiece(space: Space)
+ removePiece()
+ makeKing()
*
1
CheckersBoard
- pieces: List<<Piece>>
- squares: Matrix<<Space>>
+ getState()
+ setState()
+ resetState()
CheckersBoard state is defined by the attributes of its pieces and squares
1
Resides on
Space
- isOccupied: Boolean
- location: (int, int)
- spaceOwner: Player
+ spaceOccupied(): Boolean
+ getLocation(): (int, int)
+ setLocation(row: int, column: int)
+ getSpaceOwner(): Player
+ setSpaceOwner(player: Player)
64
1
HumanPlayer
- username: String
- password: String
- winCount: Int
- lossCount: Int
- playTime: Time
- controller: GameController
+ update()
+ getUsername(): String
- getPassword(): String
+ register(username: String, password: String): Boolean
- setUsername(username: String)
- setPassword(password: String)
+ login(username: String, password: String): Boolean
+ getWinCount(): int
+ setWinCount(wins: int)
+ getLossCount(): int
+ setLossCount(losses: int)
+ getPlayTime(): Time
+ setPlayTime(duration: int)
+ sendMessgage(message: String)
update() method used for notfying observer that subject changed state
ComputerPlayer
- difficulty: int
- brain: aiDecision
- nextMove: (Piece, Space)
+ update()
1
1
<<interface>>
aiDecision
+ getNextMove(): (Piece, Space)
+ updateKnowledge()
miniMaxDecision
- decisionTree: Tree&lt;&lt;(Piece, Space, int)&gt;&gt;
+ getNextMove(): (Piece, Space)
+ updateKnowledge()
- insertNode()
- deleteNode(pieceNum: Piece, spaceNum: Space)
- updateNode(pieceNum: Piece, spaceNum: Space, newValue: int)
Each node in the tree corresponds to the value (the int component of the node) of moving a certain Piece to a certain Space