CS451

Checkers Design Document

Summer 2017

Group members	Hajer Karoui, Samuel Nathanson, Curtis Bechtel, Julie Soderstrom
Faculty advisor	Dr. Filippos Vokolos, Ph. D.

Name	**Date**	**Reason for change**	**Revision**
Hajer Karoui Samuel Nathanson Curtis Bechtel Julie Soderstrom	08/01/2017	First Draft - Sections outlined	0.5
Hajer Karoui Samuel Nathanson Curtis Bechtel Julie Soderstrom	08/10/2017	Second Draft - Content added to all sections	1.0
Hajer Karoui Samuel Nathanson Curtis Bechtel Julie Soderstrom T. Adams	08/13/2017	Third and final Draft - Revisions from other students' feedback.	1.1

Table of Contents

1. Introduction

1.1. Purpose of the document

This document describes the design of our real-time two-player checkers game accessed remotely, described in Checkers Game Requirement. It lays out a detailed architecture of different components of the system. It is used as a reference guide by developers for a faster and organized coding process.

1.2. Scope of the document

This document describes implementation details of the remote checkers game. The application will consist of 4 main requests/functions: creating a new game, joining a game, updating a game state, and getting the game state. Data will be requested from the client and sent back from the server.

1.3. Definitions

- **A game** consists of a standard checkers board, pieces, and at least one online player. (further describe standard checkers board and pieces)
- **Public games** are displayed on a list to all players interested in joining a game.
- **Private games** are only visible to and accessible by players with the game's private ID key.
- **New games** are ones which do not yet have a second player.
- **Active games** are ones with two online players.
- **Checker piece**: a piece that can be moved and which did not reach the last row yet.
- **King**: a piece that reaches the last row (the King Row).

2. System Overview

2.1. Description of Software

The checkers game allows a player to host a public or private game, or join an already created game. One game only supports two players, who have the ability to play remotely against one another, as long they have internet connection and the server is up and running. Players have access to a GUI of a checkerboard, where they can make moves and interact with their opponent.

2.2. Technologies Used

The Checkers game requires internet connectivity to operate correctly and have remote players.

2.2.1. Database
MongoDB is used in order to store JSON objects. Each JSON object stores information about players, games, and messages. This database enables queries to be made on any of these collections of objects.

2.2.2. GUI
CSS/HTML are used to create the web pages, and populate the checkerboard.

2.2.3. Network
Data is communicated between client and servers using *Ajax* queries through HTTP. GET requests will be used for updating the HTML and CSS of the webpage.

2.2.4. Client/Server
Node.js and the Node Package Manager are used to run the server-side JavaScript code. It also uses Node.js packages including Express for asynchronous routing and BodyParser for accessing POST requests.

3. System Architecture

3.1. Architectural Design Components

3.1.1. Database
The database is used to store the states of currently running games, pending requests, and connected players. The database will primarily consist of two collections of objects: players and games. Each object in the players collection will follow the specification given in section 4.2 for player objects. Every object in the games collection will similarly follow the specification given in section 4.4 on game objects.

The database platform and language will be MongoDB. The database service provider will be mLab.

3.1.2. GUI
The user's browser will display the GUI, so the pages will be written with HTML, CSS, and Javascript. The GUI will be separated into two components: sidebar menu and main screen. The sidebar menu is a static container, and the main screen is displayed dynamically based on the current page. The main screen will use functions that manipulate the CSS styling of elements to "move" checkers pieces.

3.1.3. Client

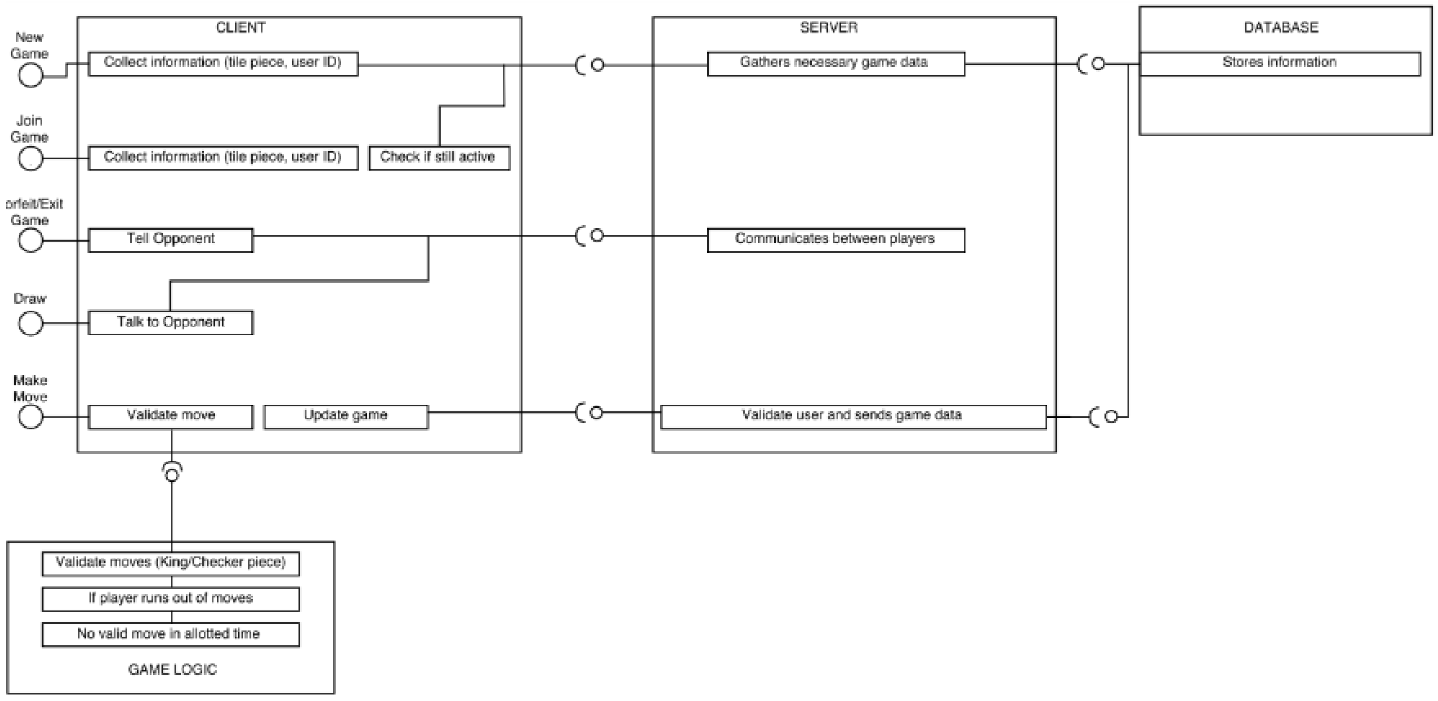
The client component is written using node.js, and it handles a few things: It sends requests to the server when hosting or joining a game. Once in a game, the client periodically requests and receives game updates in order to refresh the state of the game board. It handles move validation to ensure that the user does not make an illegal move. After move validation, the client sends game update requests to the server. The client also holds the game state model for the player.

3.1.4. Server

The server component handles a few things. It handles requests from clients and directs them to the proper components. The server handles initializing game objects and sending them to clients. It also handles sending game updates to and receiving game updates from clients. When receiving game updates, the server contains a module to verify the validity of the move and the identity of the player. The server handles creating, reading, updating, and deleting game states and messages from the database.

3.2. Architectural Structure Components

Diagram describing the relationship between the different components stated above.



CLIENT
SERVER
DATABASE
New Game
Join Game
orfeit/Exit Game
Draw
Make Move
Collect information (tile piece, user ID)
Collect information (tile piece, user ID)
Check if still active
Tell Opponent
Talk to Opponent
Validate move
Update game
Gathers necessary game data
Communicates between players
Validate user and sends game data
Stores information
Validate moves (King/Checker piece)
If player runs out of moves
No valid move in allotted time
GAME LOGIC

4. Data Design

Describes the JSON format (including properties and values) for client-side objects, server-side objects, HTTP POST requests, and HTTP POST responses.

4.1. Message Object

Stores a message sent from one player to their opponent. Messages will be sent when one player joins a game, forfeits, requests a draw, accepts a draw, rejects a draw, or sends a chat message (if implemented).

Property Name	**Value Type**	**Description**
type	string	What type of message this is. Must be one of “join”, “forfeit”, “request_draw”, “accept_draw”, “reject_draw”, or “chat”
message	string	Contains the text of the message, if there is any

4.2. Player Object

Stores information about a user who is playing a game of checkers.

Property Name	**Value Type**	**Description**
player_id	string	Unique ID for the player
player_name	string	Username of the player displayed by the client
player_number	integer	Which game player this is (0 or 1)
game_id	string	Unique ID for the game that this user is playing in
opponent_id	string	Unique ID for the opponent (same as the opponents player ID)

last_request	integer	The UNIX time stamp for the last request made by this user. This field is used to determine players which have disconnected quietly so that they can be removed from the database.
new_messages	message object	Unread messages sent from the opponent to this player

4.3. Piece Object

It’s an element of the checkerboard object. It stores information about the type of the piece (normal or king), and whose player it is.

Property Name	**Value Type**	**Description**
player	integer	The player’s number (0 or 1)
king	boolean	Whether or not the piece is a king

4.4. Game Object

Stores information about the game of checkers. The object includes information about each player and the state of the game as well as a copy of the board.

Property Name	**Value Type**	**Description**
game_id	string	The unique ID for the game
player_names	array of strings	The names of the players participating in this game
player_colors	array of strings	The colors of each player participating in this game
turn	integer	Which player’s turn it is (0 or 1)
public	boolean	Whether or not it’s a public game

active	boolean	Whether or not it’s an active game (waiting for a player to join)
board	array of arrays of piece objects	The current state of the board. A matrix of piece objects that each denote which player it belongs to as well as if it is a king or not

4.5. Get Games Request Object

Requests the list of games to be displayed for the user (if the game is both public and active). This object is sent when a player requests to join a game.

Property Name	Value Type	Description
(none)		

4.6. Get Games Response Object

Sends back a list of game objects to be displayed to the user.

Property Name	Value Type	Description
error	boolean / string	This will be true if the request was successful, otherwise it will contain a string describing the error.
games	array of game objects	The list of all active and public games in the database.

4.7. New Game Request Object

Requests a new game to be made including the name of the player who hosts it as well as whether it’s open to other to join or if it is a private game.

Property Name	Value Type	Description
player_name	string	The username of player

public	boolean	Whether or not the game will be public or private

4.8. New Game Response Object

Responds with the player who is hosting’s name as well as the newly created game object

Property Name	**Value Type**	**Description**
error	boolean / string	This will be true if the request was successful, otherwise it will contain a string describing the error.
player	player object	The player object with data about the player that made this request
game	game object	The game object with data about the current state of the game

4.9. Join Game Request Object

Requests sending information about the player that wants to join the game with the given id.

Property Name	**Value Type**	**Description**
player_name	string	The name of the player
game_id	string	The unique ID of the game

4.10. Join Game Response Object Object

Responds with the guest player's object and the game created by the host.

Property Name	**Value Type**	**Description**
error	boolean / string	This will be true if the request was successful, otherwise it will contain a string describing the error.
player	player object	The player object with data about the player that made this request
game	game object	The game object with data about the current state of the game

4.11. Make Move Request Object

Requests for a move to be made sending the id of the player who is requesting to move the piece as well as the coordinates of where it should be moved.

Property Name	**Value Type**	**Description**
player_id	string	The player's unique ID
move	array of arrays of integers	The coordinates of where the piece started (row and column) and each consecutive position throughout the move

4.12. Make Move Response Object

Sends a *success* or *failure* response to the UI-Manager. If it is an invalid move, the user will be prompted to make another attempt. If it's a success, the board will be updated with the new piece's coordinates and the opponent gets to play.

Property Name	**Value Type**	**Description**

error	boolean / string	This will be true if the request was successful, otherwise it will contain a string describing the error.
game	game object	the game's information (including the game id, players, colors, etc)

4.13. Get Updates Request Object

Requests for any changes in the game board, messages, etc for the given player.

Property Name	Value Type	Description
player_id	string	The player's unique ID

4.14. Get Updates Response Object

Sends the game object and messages objects back to update the players.

Property Name	Value Type	Description
error	boolean / string	This will be true if the request was successful, otherwise it will contain a string describing the error.
game	game object	The game's information (including game id, players, colors, etc)
messages	array of message objects	The message objects sent to each player

4.15. Send Message Request Object

Sends a message object from the sender to the receiver (opponent). The message, as described above, describes when one player joins a game, forfeits, requests a draw, accepts a draw, rejects a draw, or sends a chat message (if implemented).

Property Name	Value Type	Description

player_id	string	The player’s unique id
message	message object	Message sent to the opponent

4.16. Send Message Response Object
The HTTP response to a send message request. This tells the client whether the request was successful.

Property Name	**Value Type**	**Description**
error	boolean / string	This will be true if the request was successful, otherwise it will contain a string describing the error.

5. Component Design

5.1. Overview
This section further describes each component of the system, along with their attributes and method descriptions.

5.2. GUI
The GUI includes two main components: the sidebar menu and the main screen. Within the main screen will be four more visual components:

5.2.1. Sidebar Menu
The sidebar menu includes buttons which interact with the client component: host and join game.

Start Game Button

Clicking on this button changes the view of the main screen to display the Host Game Form.

Join Game Button

Clicking on this button changes the view of the main screen to display a list of the public active games and a form to input the username and the ID of the game to join.

5.2.2. Main Screen
The main screen includes different screens depending on the user’s selections:

Host Game Form

It contains input box for username,two radio buttons for checker piece color (Black and White or Red and White), two radio buttons for public or private, and a Submit button to send the data to the server to create the new game.

Join Game Form

It contains input boxes for username and the game ID, and a submit button to send the data to the server to join the selected game referenced by its ID.

Public Games List

It is displayed when the player requests to join the game. The main screen displays a list of active, public games where each element/game is a radio button that can be selected by the user.

Game Board

Displays the checkerboard created through CSS and 3 buttons for calling the forfeit/exit function, the raw function, and pausing the game. Also displays the time left for each player.

5.3. Client

The client component interacts with the user through the GUI and allows the user to perform certain tasks such as creating a game, joining a game, forfeiting a game, etc by communicating with the server.

function newGame

Gets information about username and whether it is public to store in the database

Parameter Name	**Value Type**	**Description**
player_name	string	The username of the player creating the game
is_public	boolean	Whether or not the game is public
callback	function	The function to be called when this operation has completed

function joinGame

Used to join active public or private game utilizing the given game ID and player's username.

Parameter Name	Value Type	Description
player_name	string	The username of the player joining the game
game_id	string	The game id of the game being joined
callback	function	The function to be called when this operation has completed

function forfeitGame

Sends an asynchronous message to opponent notifying them that the other player has forfeit the game.

Parameter Name	Value Type	Description
callback	function	The function to be called when this operation has completed

function requestDraw

Sends an asynchronous message to the opponent requesting a draw (tie game).

Parameter Name	Value Type	Description
callback	function	The function to be called when this operation has completed

function makeMove

Gets the move coordinate and checks if move is valid. If valid, returns the updated board.

Parameter Name	Value Type	Description

move	array of arrays of integers	The coordinates for the current location of the piece and the designated location(s) of the piece
callback	function	The function to be called when this operation has completed

function drawBoard

Redraws the checkers game board

Parameter Name	**Value Type**	**Description**
board	array of arrays of pieces	The game board data (a matrix of game pieces) to be drawn to the game screen

5.4. app.js

This module contains logic for initializing and starting the server and will be the main entry point for the client to communicate with the server. It will also manage each HTTP request endpoint by routing each request to the correct destination by using external modules such as Express and BodyParser.

const path

An instance of the Path module for file paths.

const express

An instance of the Express module for routing requests to endpoints asynchronously.

const body_parser

An instance of the BodyParser module for parsing the POST requests into JSON.

const requests

An instance of the RequestManager local module for managing different kinds of requests.

const port

The port number for this server to use. It is set to the value of the environment variable PORT, or 8080 if PORT isn't set.

const public_dir

The directory for files with public access such as static HTML and CSS files. These will be served statically from the server to the client.

route get/

Redirects requests for the root page to index.html

Parameter Name	Value Type	Description
req	object	The Node.js HTTP-request object
res	object	The Node.js HTTP-response object

function returnResponse

Creates and returns a callback function for automatically sending a HTTP response containing an error code and an object.

Parameter Name	Value Type	Description
res	object	The Node.js HTTP-response object

route post/get-games

Route for the client to get a list of public and active games

Parameter Name	Value Type	Description
req	object	The Node.js HTTP-request object.
res	object	The Node.js HTTP-response object

route post/join-game

Route for the client to request the creation of a new game

Parameter Name	Value Type	Description
req	object	The Node.js HTTP-request object.
res	object	The Node.js HTTP-response object

route post/make-move

Route for the client to make a move and update the game state

Parameter Name	Value Type	Description
req	object	The Node.js HTTP-request object.
res	object	The Node.js HTTP-response object

route post/get-updates

Route for the client to request updates on the state of the game

Parameter Name	Value Type	Description
req	object	The Node.js HTTP-request object.
res	object	The Node.js HTTP-response object

route post/send-message

Route for the client to send messages to an opponent

Parameter Name	Value Type	Description
req	object	The Node.js HTTP-request object.

res	object	The Node.js HTTP-response object

5.5. requestManager.js

Receives request routes from app.js

const path

an instance of the Path module for file paths

const db

an instance of the databaseManager module for connecting to the database

const checkers

an instance of the checkersGame module

function getGames

Receives all HTTP POST requests to get the list of public active games. This method will make a request to the database for the list of games. Then, it will filter out all game IDs and usernames to be returned to the original client.

Parameter Name	**Value Type**	**Description**
callback	function	The function to be called when this operation has completed

function newGame

Receives all HTTP POST requests to create a new game of checkers. This method will attempt to generate a new game ID and then store this ID along with a default game board into the database. It will then generate a new player ID and store this ID along with a new player object into the database. Finally, it will return the generated game and player objects to the original client.

Parameter Name	**Value Type**	**Description**
player_name	string	The username of the player creating the game
is_public	boolean	Whether or not the game is public
callback	function	The function to be called when this operation has completed

function joinGame

Receives all HTTP POST requests to join an active game of checkers. This method will check that the given game ID corresponds to an active game, then it will generate a new player ID and store this ID along with a new player object into the database. It will also update the game object to reflect the new player. Finally, it will then return the game and player objects to the original client.

Parameter Name	**Value Type**	**Description**
player_name	string	The username of the player joining the game
game_id	string	The game ID of the game being joined
callback	function	The function to be called when this operation has completed

function makeMove

Receives all HTTP POST requests for the player to make a valid move on the board. This method will verify that the player's ID is valid, then retrieve the corresponding game object from the database. It will verify that the given move is valid, then update the board to reflect this and store it to the database. Finally, it will return the updated game board to the original client.

Parameter Name	**Value Type**	**Description**
player_id	string	The unique ID of the player
move	array of arrays of integers	A list of coordinates representing the intended checkers move.
callback	function	The function to be called when this operation has completed

function getUpdates

Receives all HTTP POST requests to get updates about the game state and any unread messages. This method will retrieve the player object from the database and compile all updated messages. If it indicates that the game state has also been updated, it will add this to the return object. Finally, the unread messages and optional game object will be returned to the original client.

Parameter Name	Value Type	Description
player_id	string	The unique ID of the player
callback	function	The function to be called when this operation has completed

function sendMessage

Receives all HTTP POST requests to send messages from one player to their opponent. This method will lookup the player sending the message, determine the ID of their opponent, then add the message to the opponent's list of unread messages. It will then return a simple success message to the original client.

Parameter Name	Value Type	Description
player_id	string	The unique ID of the player
message	string	The message object to be sent to the opponent
callback	function	The function to be called when this operation has completed

5.6. databaseManager.js

This module will be the main point of communication with the MongoDB database. It will include methods for initiating a connection with the database and for retrieving and inserting data.

const client

Loads required modules

const playerIndexes

Indexes for the players collection

const gameIndexes

Indexes for the games collection

const id_chars

The set of valid characters from which IDs should be created.

const id_length

The number of characters in each ID string. Using 12 characters from a 36 character set produces approximately ln(3612)ln(2)62bits of information, which nearly guarantees very few ID collisions.

var db

The database object (null if not connected to a database)

var colls

The set of collections in the current database (players and games)

function createRandomID

Creates and returns a random ID for the game. The generated ID will have the length and character set specified above.

Parameter Name	**Value Type**	**Description**
(return)	string	A random ID for a user or game. This

function createUrl

Creates and returns the MongoDB connection URL.

Parameter Name	**Value Type**	**Description**
username	string	the MongoDB account username
password	string	the MongoBD account password
address	string	the address of the MongoDB database
database	string	the name of the database to be used

(return)	string	the MongoDB connection URL

function isConnected

Determines if this manager has connected to the database.

Parameter Name	Value Type	Description
(return)	boolean	true if it is connected, otherwise false

function loadCollection

Loads a collection from the database into the manager. If the collection doesn't exist, it creates it with the given indexes.

Parameter Name	Value Type	Description
name	string	the name of the database
indexed	object	an object containing the indexing options for the collection
callback	function	the function to be called when this operation has completed

function connect

Connects to the database specified by the environment variables DB_USER, DB_PASS, DB_ADDR, and DB_NAME.

Parameter Name	Value Type	Description
callback	function	the function to be called when this operation has completed

function newPlayer

Creates a new player with the given username and player number by querying the database until it generates a unique player ID.

Parameter Name	Value Type	Description
username	string	player’s username
number	integer	if it’s player 0 or 1
callback	function	the function to be called when this operation has completed

function newGame

Creates a new game.

Parameter Name	Value Type	Description
is_public	boolean	whether or not the game is public
callback	function	the function to be called when this operation has completed

function getPlayer

Retrieves the player with the given ID from the database and passes it as the second parameter to the callback function.

Parameter Name	Value Type	Description
player_id	string	the unique ID of the player
callback	function	the function to be called when this operation has completed

function updatePlayer

Updates a player object in the database.

Parameter Name	Value Type	Description
player	player object	the player object to be updated in the database

callback	function	the function to be called when this operation has completed

function getGame

Retrieves the game with the given ID from the database and passes it as the second parameter to the callback function.

Parameter Name	Value Type	Description
game_id	string	the unique ID of the game
callback	function	the function to be called when this operation has completed

function updateGame

Updates a game object in the database

Parameter Name	Value Type	Description
game	game object	the game object to be updated in the database
callback	function	the function to be called when this operation has completed

function getGamesList

Retrieves a list of all active public games from the database and passes it as the second parameter to the callback function.

Parameter Name	Value Type	Description
callback	function	the function to be called when this operation has completed

function sendMessage

Sends a message to a player's opponent by storing it in their opponent's list of messages

Parameter Name	Value Type	Description

callback	function	the function to be called when this operation has completed

5.7. Database

The server communicates with the client component and the database to gather the necessary information to then perform tasks sent from the client component. The database is designed to store two types of objects: player and game. MongoDB databases split the two objects into different “collections” which are formatted like SQL tables.

Player Object

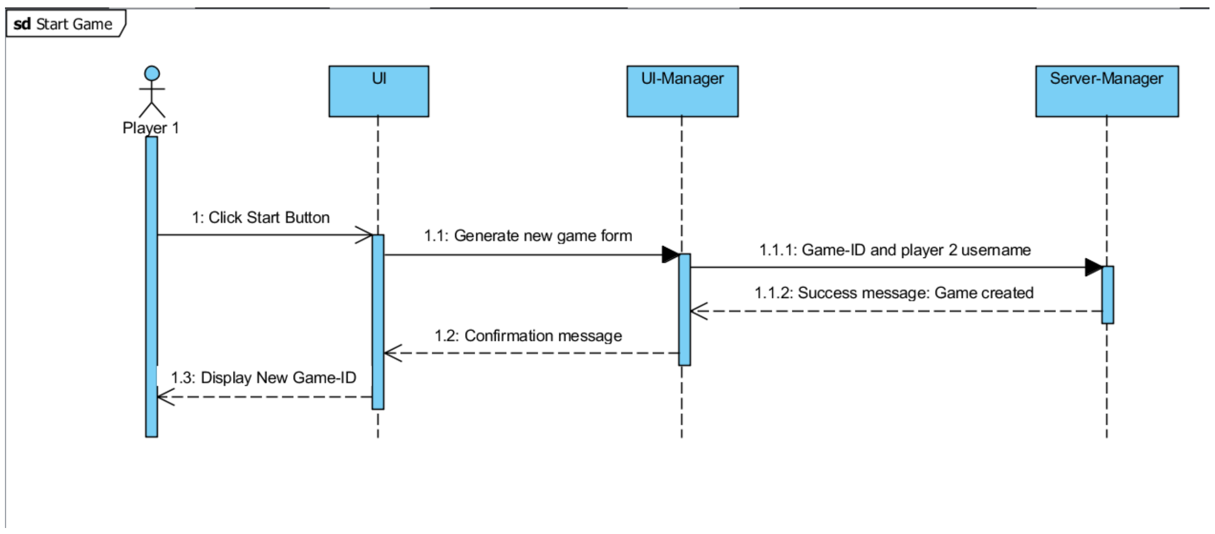
The player object will hold information about each player connected to the game-- the player’s ID, name, number, as well as the game ID of that player’s current game. It will also store messages in an array.

Game Object

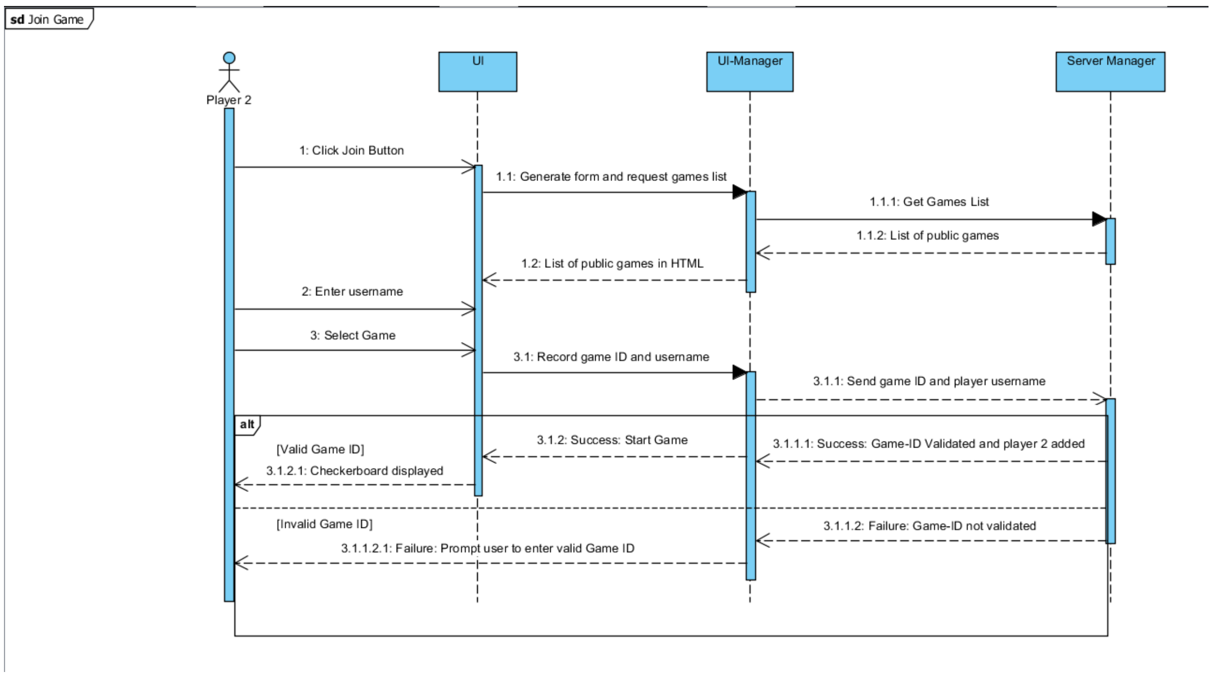
The game object holds information about the game ID, player names, player colors, whose turn it is, whether it is public or private, and whether it is active or not. It also stores the board which is constructed by an array of arrays with the player number and the type of checker piece.

6. Diagrams

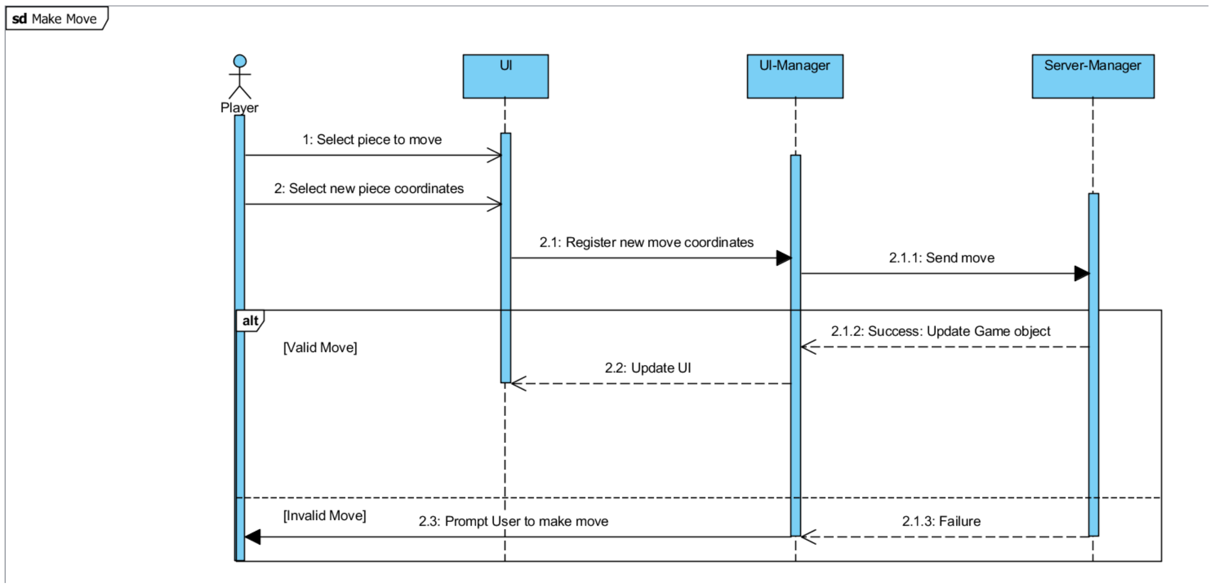
Start Game Sequence:


sd Start Game
Player 1
UI
UI-Manager
Server-Manager
1: Click Start Button
1.1: Generate new game form
1.1.1: Game-ID and player 2 username
1.1.2: Success message: Game created
1.2: Confirmation message
1.3: Display New Game-ID

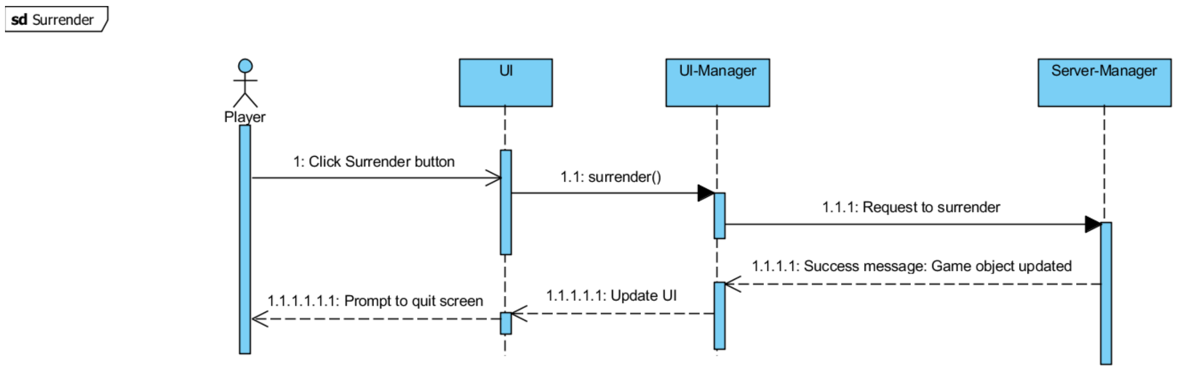
Join Game Sequence:


sd Join Game
Player 2
UI
UI-Manager
Server Manager
1: Click Join Button
1.1: Generate form and request games list
1.1.1: Get Games List
1.1.2: List of public games
1.2: List of public games in HTML
2: Enter username
3: Select Game
3.1: Record game ID and username
3.1.1: Send game ID and player username
alt
[Valid Game ID]
3.1.2: Success: Start Game
3.1.1.1: Success: Game-ID Validated and player 2 added
3.1.2.1: Checkerboard displayed
[Invalid Game ID]
3.1.1.2: Failure: Game-ID not validated
3.1.1.2.1: Failure: Prompt user to enter valid Game ID

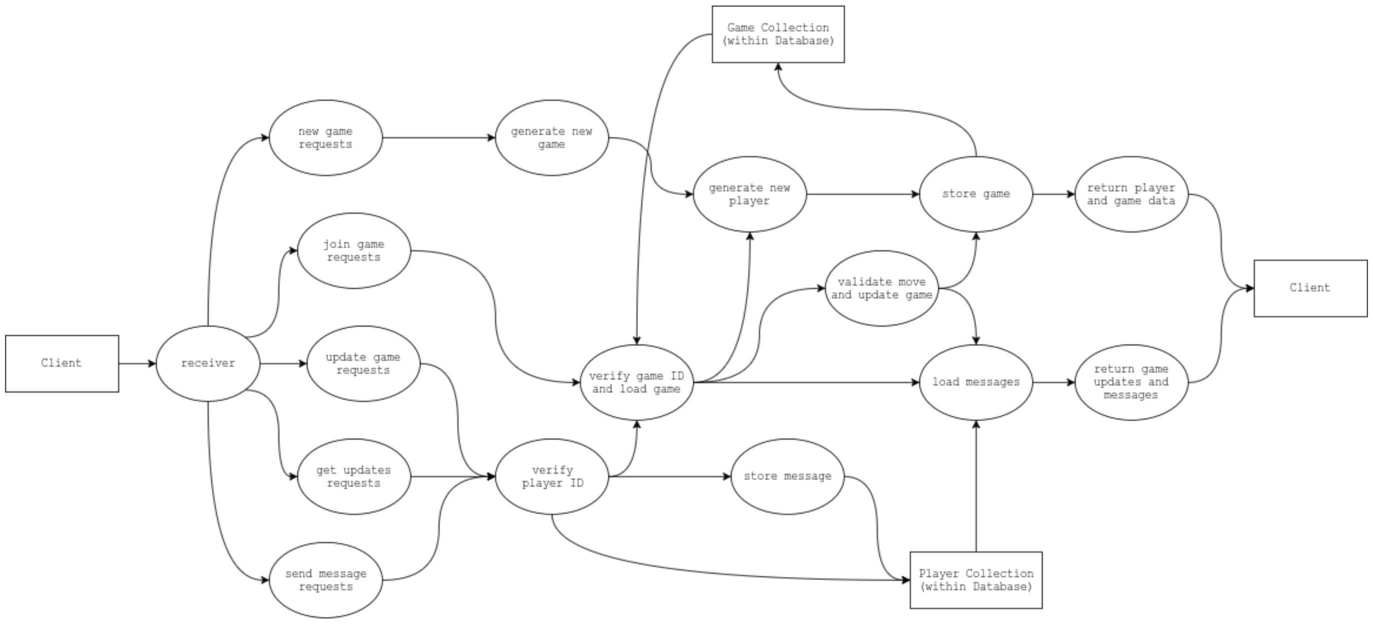
Make a Move Sequence:



Surrender Sequence:


sd Surrender
Player
UI
UI-Manager
Server-Manager
1: Click Surrender button
1.1: surrender()
1.1.1: Request to surrender
1.1.1.1: Success message: Game object updated
1.1.1.1.1: Update UI
1.1.1.1.1.1: Prompt to quit screen

Server Data Flow:



7. References

[1] Karoui, Nathanson, Bechtel, Soderstrom. (Remote Checkers Requirements Specifications Document).Drexel University. Philadelphia, PA, United States, 2017.