Checkers

Software Design Document

GROUP 4

|  |  |
| --- | --- |
| **Group Members** | Omair Bhore, Phuc Ngo, Nhan Nguyen, Bruce Zhao |
| **Faculty Advisor** | Dr. Filippos Vokolos, Ph. D. |
| **Teaching Assistant** | Mr. Sean Grimes |

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Revision Information** | **Revision** |
| Omair Bhore  Nhan Nguyen  Phuc Ngo  Bruce Zhao | 08/05/2017 | First Draft - Created and planned out sections and subsections | 1.0 |
| Omair Bhore  Nhan Nguyen  Phuc Ngo  Bruce Zhao | 08/06/2017 | Second Draft - Begun fleshing out previous sections and subsections | 1.1 |
| Omair Bhore  Nhan Nguyen  Phuc Ngo  Bruce Zhao | 08/11/2017 | Third Draft - | 1.2 |

Table of Contents

1. **Document Introduction**
2. Purpose
3. Scope
4. Definitions, Acronyms, Abbreviations
   1. Game State
   2. Player
   3. Game Environment
5. **System Overview**
6. Description of Software
7. Technologies Used
8. **System Architecture**
9. Architectural Design Components
10. Design Rationale
11. **Component Design**
    1. Overview
    2. Front-End Design
    3. Back-End Design
12. **Human Interface Design**
    1. Overview of User Interface
    2. Screen Objects and Actions
    3. Server Menu Flow
    4. Client Menu Flow
13. **References**

**1. Document Introduction**

**1.1 Purpose**

This document is to describe the implementation of a multiplayer Checkers game as described in the requirements document. The multiplayer Checkers game is designed to be played between two players on different systems and allows for multiple game sessions to occur at the same time.

**1.2 Scope**

This document describes the implementation details of the multiplayer

Checkers game. This software shall consist of several components. All components fall under one of two major categories called the **Front-End** or the **Back-End**. This document is not covering the testing of the various software components.

**1.3 Definitions, Acronyms, Abbreviations**

**1.3.1 Game State**

**Lobby Page**

Refers to the landing page all users shall use to create a username when they first accesses the multiplayer Checkers game’s website. The user shall use this screen again before starting a new game or when quitting a game during or after a session.

**Game Session**

Refers to the page all users shall face after they have been connected. This page allows both users to play a game of Checkers.

**End of Game Session**

Refers to the modal page all users will face after a game has ended, either due to a winning move or due to a player timing out/quitting.

**1.3.2 Player**

**Username**

A user’s name entered by the user in the lobby when they enter the it for the first time.

**Objectives**

The goal is for a player to eliminate all of the opponent's pieces. This would grant that player the win. A player could also win by cornering their opponent, leading to a player's inability to make any moves.

**1.3.3 Game Environment**

**Checkers Board**

An 8 by 8 board with alternating black and white tiles. The game pieces are placed on the black squares.

**Checkers Pieces**

12 black or red checkers pieces assigned to a player. A player can click and hold onto a game piece and this will show a player all the legal moves for the selected game piece by highlighting various squares across the checkers board. The player can then drag the game piece and drop it on a highlighted square while any other space would be an illegal move and not accepted.

**2. System Overview**

**2.1 Description of the Web Application**

Checkers is a one-on-one game designed to be casually played with a minimal ruleset so that people of all ages, creeds, and births can play. Each player is pitted against each other, one side being red and the other being black, on an 8 by 8 board, not so different than a checkers board. Players will interact with the game through dragging and dropping the checkers pieces to the desired tiles.

**2.2 Technologies used**

Checkers will use NodeJS as the network environment and Express as the server framework. The application will also use Socket.io to implement the real-time, bidirectional communication between players and the server in a game session for real-time UI updating.

ReactJS will be used to build the front-end, and ReactDnD to support the drag and drop feature for ReactJS. The front-end shall use Redux for UI state management.

The target browser for Checkers on a desktop computer or a laptop is Google Chrome, and Android for smartphones in future development cycles with further funding. The development environment is NodeJS with the WebStorm IDE. Github and Git are being used for version control.

**3. System Architecture**

**3.1 Architecture Design Component**

The program allows two systems, known as clients, to connect via sockets to a server. The networking system consists of three main components: the Client(s), the Socket, and the Server.

**3.1.2 Client(s)**

A client is a remote system which connects to a server via sockets.

**3.1.3 Server**

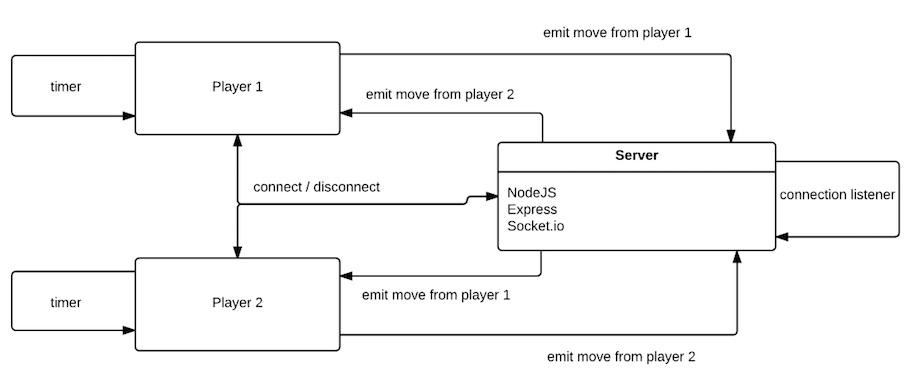
The server starts a game session when two clients have been connected via sockets. The server displays a webpage with a checkers board, twelve red pieces, twelve black pieces, an activity log, and a button to give up.

**3.1.4 Socket**

The socket is a way of connecting two clients to a server. It allows bi-directional communication layer which abstracts the program from cross-browser compatibility.

**3.1.7 Front-End**

The front-end contains multiple components spread across two pages and a modal page.

****

**Figure 1. Architecture Flow**

**3.2 Design Rationale**

**Why browser based rather than a standalone client?**

We went with a browser based version for the game because we wanted to keep true with the purpose of checkers. With the game designed for a browser, we allow for a wider audience. A standalone client would have required more from the users that could deter people from playing. The other reason for focusing our effort on a browser based game over a standalone copy is that for the purposes of playing checkers, the less steps necessary between a user and a game, the closer we get to the reason for checkers: its simplicity and easy of access.

**Why focus on Google Chrome?**

According to StatCounters, the current market share for browsers has google chrome leading the pack at 54.27% and is the dominating browser for most countries. Keeping with the purpose of checkers, making it accessible to the most amount of people is our goal. For that reason, if we optimize it for Google Chrome, we optimize it for a majority of browsers. Google Chrome also supports the new version of HTML5 which would allow for support for ReactJS, which we use to build our front-end.

|  |
| --- |
|  |
| Figure 2. Browser Market Share. (Chrome is in green) |

**4. Component Design**

**4.1 Overview**

In this section, the details for each component shall be given. There shall be one section for the front-end components, and one section for the back-end components. For each component, a UML and description are provided.

**4.2 Front-end**

**4.2.1 Lobby Page Component**

|  |
| --- |
| **Screen Shot 2017-08-12 at 8.14.48 PM.png** |
| **Figure 3. Lobby Page Component Layout** |

The Lobby Page component is the first page the user visits. Figure 3 displays the layout and the positions of its three sub components**.** The three sub components includes: Game Title component, Input component, and the Start Button Component.

|  |
| --- |
| Screen Shot 2017-08-12 at 9.24.40 PM.png |
| **Figure 3.1. Lobby Page Component UML** |

**4.2.1.a Methods**

|  |  |
| --- | --- |
| ReactComponent render() |  |
| Input: | void |
| Output: | ReactComponent |
| Description | Renders the Lobby page React Component to HTML DOM Node |

**4.2.2. Game Title Component**

|  |
| --- |
| **Screen Shot 2017-08-12 at 9.26.29 PM.png** |
| **Figure 3.2. Game Title Component UML** |

**4.2.2.a Attributes**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| gametitle | string | The title of the game |

**4.2.2.b Methods**

|  |  |
| --- | --- |
| ReactComponent render() |  |
| Input: | void |
| Output: | ReactComponent |
| Description | Renders the Game Title Component to HTML DOM Node |

**4.2.3 Input Component**

|  |
| --- |
| **Screen Shot 2017-08-12 at 10.20.22 PM.png** |
| **Figure 3.3. Input Component UML** |

**4.2.3.a Attributes**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| username | string | The username |

**4.2.3.b Methods**

|  |  |
| --- | --- |
| ReactComponent render() |  |
| Input: | void |
| Output: | ReactComponent |
| Description | Renders the Input Component to HTML DOM Node |

|  |  |
| --- | --- |
| boolean checkInput(string) |  |
| Input: | string |
| Output: | boolean |
| Description | Checks whether the username contains invalid characters and whether the username has already been taken |

**4.2.4 Start Button Component**

|  |
| --- |
| **Screen Shot 2017-08-13 at 5.32.23 PM.png** |
| **Figure 3.4. Start Button Component UML** |

**4.2.4.a Methods**

|  |  |
| --- | --- |
| ReactComponent render() |  |
| Input: | void |
| Output: | ReactComponent |
| Description | Renders the Start Button Component to HTML DOM Node |

|  |  |
| --- | --- |
| void handleClick(event) |  |
| Input: | event |
| Output: | void |
| Description | An event handler for the clicking event when the button is clicked. This shall notify the server to add the player to the matching queue and start waiting. Once finish, it will redirect the player to a game page. |

**4.2.5 Game Page Component**

|  |
| --- |
| **Screen Shot 2017-08-12 at 8.55.59 PM.png** |
| **Figure 4. Game Page Component Layout** |

The Game Page component is the page where the players play a game of checkers. Figure 4 displays the layout and positions its three sub components**.** The three sub components include: the Checkers Board component, Log component, and Quit (Give up) Button Component.

|  |
| --- |
| **Screen Shot 2017-08-12 at 9.57.29 PM.png** |
| **Figure 4.1 Game Page UML** |

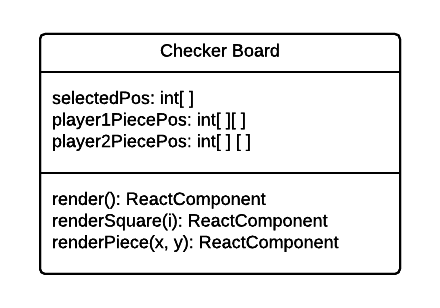
**4.2.5.a Methods**

|  |  |
| --- | --- |
| ReactComponent render() |  |
| Input: | void |
| Output: | ReactComponent |
| Description | Renders the Game Page Component to HTML DOM Node |

**4.2.6 Checkers Board Component**

|  |
| --- |
| **Screen Shot 2017-08-13 at 2.52.22 PM.png** |
| **Figure 5. Checkers board Layout** |

The Checkers Board component displays the 8 by 8 board with checkers pieces. Figure 4 shows the layout for this component. The squares in the board are represented by the Square Container component and Square component, and the checkers pieces are represented by the Piece component. The Square Container component shall handle the logic for the move validation and the Square component shall handle the rendering. Piece components will not render for the Square components where there are not any checkers piece. To keep with the simplicity of the layout, only one Square component with a Checkers piece component instead of all 64 squares shall be presented in the layout above.

****

**4.2.6.a Attributes**

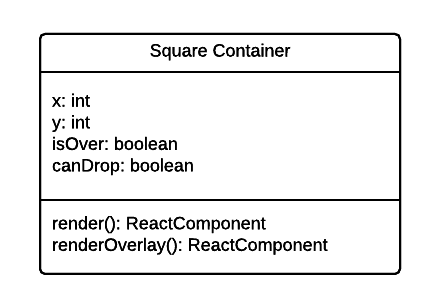
|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| selectedPos | int[ ] | The position of the clicked piece in the form of [x, y] |
| player1PiecePos | int[ ][ ] | An array of all positions in the [x, y] form of player 1’s pieces |
| player2PiecePos | int[ ][ ] | An array of all positions in the [x, y] form of player 1’s pieces |

**4.2.6.b Methods**

|  |  |
| --- | --- |
| ReactComponent render() |  |
| Input: | void |
| Output: | ReactComponent |
| Description | Renders the Checker Board Component to HTML DOM Node |

|  |  |
| --- | --- |
| ReactComponent renderSquare(i) |  |
| Input: | int |
| Output: | ReactComponent |
| Description | Creates 64 squares by rendering multiple Square Containers |

|  |  |
| --- | --- |
| ReactComponent renderPiece(x, y) |  |
| Input: | int, int |
| Output: | ReactComponent |
| Description | Renders each piece and assigns the starting positions |

****

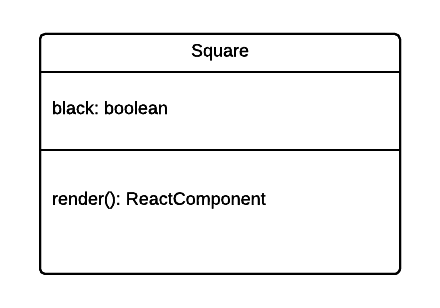
**4.2.6.a Attributes**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| x | int | The x coordinate of the piece |
| y | int | The y coordinate of the piece |
| isOver | boolean | Checks whether a piece is hovering on top of a Square |
| canDrop | boolean | Checks whether a piece can be dropped on the Square |

**4.2.6.b Methods**

|  |  |
| --- | --- |
| ReactComponent render() |  |
| Input: | void |
| Output: | ReactComponent |
| Description | Renders the Square Container Component to the HTML DOM Node |

|  |  |
| --- | --- |
| ReactComponent renderOverlay() |  |
| Input: | void |
| Output: | ReactComponent |
| Description | Renders the Square style when a piece is dragged over it. The Square changes color to show whether the move is valid. |

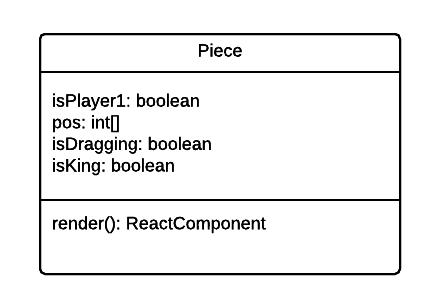
****

**4.2.6.a Attributes**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| black | boolean | Flag to check whether the square is black or white |

**4.2.6.b Methods**

|  |  |
| --- | --- |
| ReactComponent render() |  |
| Input: | void |
| Output: | ReactComponent |
| Description | Renders the Square Component to HTML DOM Node |

****

**4.2.6.a Attributes**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| isPlayer1 | boolean | Flag to check whether the piece is owned by player 1 or 2 |
| pos | int[ ] | Position of the piece in the form of [x, y] |
| isDragging | boolean | Flag to check whether the piece is being dragged |
| isKing | boolean | Flag to check whether the piece is king |

**4.2.6.b Methods**

|  |  |
| --- | --- |
| ReactComponent render() |  |
| Input: | void |
| Output: | ReactComponent |
| Description | Renders the Piece Component to HTML DOM Node |

**4.2.? Log Component**

|  |
| --- |
| **Screen Shot 2017-08-13 at 3.14.01 PM.png** |
| **Figure 6. Log Component Layout** |

The Log component displays which moves each player make during a game. The moves shall be listed in a vertical list. Figure 6 shows the layout for the Log component.

|  |
| --- |
|  |
| **Figure 6.1. Log Component UML** |

**4.2.4.a Methods**

|  |  |
| --- | --- |
| ReactComponent render() |  |
| Input: | void |
| Output: | ReactComponent |
| Description | Renders the Log Component to HTML DOM Node. |

**4.2.? Log List Component**

|  |
| --- |
|  |
| **Figure 6.2. Log List Component UML**  **4.2.6.a Attributes**   |  |  |  | | --- | --- | --- | | Name | Type | Description | | LogItems | string[] | The description of all logs |   **4.2.4.a Methods**   |  |  | | --- | --- | | ReactComponent render() |  | | Input: | void | | Output: | ReactComponent | | Description | Renders the Log List Component to HTML DOM Node | |

|  |  |
| --- | --- |
| ReactComponent renderContent(string []) |  |
| Input: | string[] |
| Output: | ReactComponent[] |
| Description | Creates a list of Log Item components with corresponding description from the parameters |

**4.2.? Log Item Component**

|  |
| --- |
|  |
| **Figure whatever. Log Item Component UML** |

**4.2.6.a Attributes**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| description | string | The description of a log |

**4.2.4.a Methods**

|  |  |
| --- | --- |
| ReactComponent render() |  |
| Input: | void |
| Output: | ReactComponent |
| Description | Renders the Log Item with the description to the HTML DOM Node |

**4.2.3 Modal Component**

|  |
| --- |
| **Screen Shot 2017-08-12 at 10.39.22 PM.png** |
| **Figure 6. Modal Component Layout** |

The Modal component is a popup modal which displays a button to redirect the players back to the Lobby page. It has one sub component, which is Back To Lobby Button component. Figure 6 shows the layout for the Modal component.

|  |
| --- |
| **Screen Shot 2017-08-12 at 10.39.16 PM.png** |
| **Figure 6.1. Modal Component UML** |

**4.2. .a Attributes**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| display | boolean | Flag to check whether to render the modal and its content |

**4.2. .b Methods**

|  |  |
| --- | --- |
| ReactComponent render() |  |
| Input: | void |
| Output: | ReactComponent |
| Description | A method to render a React element into a root DOM node |

|  |  |
| --- | --- |
| ReactComponent renderContent(boolean) |  |
| Input | boolean |
| Output | ReactComponent |
| Description | Helper function for render() to render the modal content based on the ‘display’ flag |

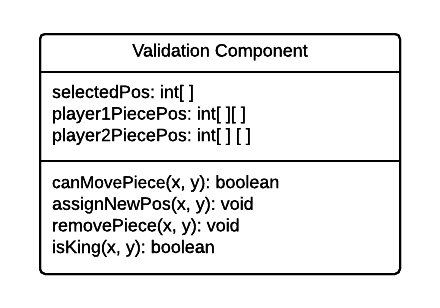
**4.2.4 Back To Lobby Button Component**

|  |
| --- |
| **Screen Shot 2017-08-13 at 5.32.43 PM.png** |
| **Figure 6.2. Back To Lobby Button Component UML** |

|  |  |
| --- | --- |
| ReactComponent render() |  |
| Input: | void |
| Output: | ReactComponent |
| Description | A method to render a React element into a root DOM node |

|  |  |
| --- | --- |
| void handleClick(event) |  |
| Input: | event |
| Output: | void |
| Description | An event handler for the clicking event when the button is clicked. This shall redirect the players back to the Lobby page. |

**4.2.5 Validation Component**

****

**4.2. .a Attributes**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| selectedPos | int[ ] | The position of the clicked piece in the form of [x, y] |
| player1PiecePos | int[ ][ ] | An array of all positions in the [x, y] form of player 1’s pieces |
| player2PiecePos | int[ ][ ] | An array of all positions in the [x, y] form of player 1’s pieces |

**4.2. .b Methods**

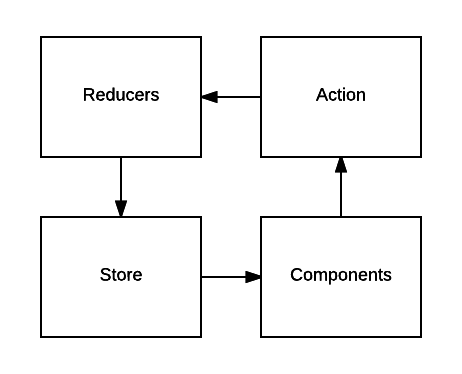
|  |  |
| --- | --- |
| boolean canMovePiece(x, y) |  |
| Input: | int, int |
| Output: | boolean |
| Description | Checks which locations are valid for moving the piece from the current [x, y] |

|  |  |
| --- | --- |
| void assignNewPos(x, y) |  |
| Input | int, int |
| Output | void |
| Description | Assigns new position for the checker that has been moved |

|  |  |
| --- | --- |
| void removePiece(x, y) |  |
| Input | int, int |
| Output | void |
| Description | Removes the piece that is jumped over by another piece |

|  |  |
| --- | --- |
| boolean isKing(x, y) |  |
| Input | int, int |
| Output | boolean |
| Description | Checks whether the piece is moved to a “King” location |

**4.2.6 State Management Component**

****

**Figure ??? Redux State Management Diagram**

Redux is used as the state management for the front-end. Each components can dispatch an action which shall be resolved by one of the reducers. Each action shall have a unique type with the data it want to change. They shall perform the necessary server call to update the data over socket. The reducers shall perform necessary procedures corresponding to the type and the data of the issued action to update the desirable properties in the front-end state stored in a global store that can be accessed globally by all components. Below are tables which describe what are needed for the state management. Table 1 lists the necessary properties for the front-end state.Table 2 lists the necessary actions to update the state for the front-end. Table 3 lists the corresponding resolver of the reducer for some of the actions to update to front-end state.

|  |  |
| --- | --- |
| **Name** | **Description** |
| socketID | The id of the socket room |
| username | The username of the player |
| wait | Flag to trigger waiting |
| isTurn | Flag to check whether it is |
| color | Checker piece color |
| numPieces | The number of a player’s checker pieces |
| win | Flag to check whether a player is a winner |
| lose | Flag to check whether a player is a loser |
| shouldModalDisplay | Flag to check whether to display the modal popup |
| board | A 2D array representing the Checker board |
| error | Error object of the state |

**Table 1. Properties of the front-end’s state**

|  |  |  |
| --- | --- | --- |
| **Type** | **Data** | **Description** |
| **INITIALIZE\_SOCKET** | socketID: string | Action to initialize socket with the server |
| **INITIALIZE\_SOCKET\_SUCCESS** | void | Action to handle success of socket connection |
| **INITIALIZE\_SOCKET\_ERROR** | error: obj | Action to handle error of socket connection |
| **SET\_USERNAME** | username: string | Action to set the username |
| **SET\_USERNAME\_SUCCESS** | username: string | Action to handle success of setting username |
| **SET\_USERNAME\_ERROR** | error: obj | Action to handle error of setting username |
| **TRIGGER\_WAIT** | void | Action to trigger waiting |
| **SET\_TURN** | void | Action to set player’s turn |
| **SET\_COLOR** | color: string | Action to set player’s checker piece color |
| **SET\_NUMBER\_CHECKER\_PIECES** | num: int | Action to set player’s number of checker pieces |
| **SET\_WIN** | void | Action to set player to winner |
| **SET\_LOSE** | void | Action to set player to loser |
| **TRIGGER\_MODAL\_DISPLAY** | void | Action to trigger modal display |
| **UPDATE\_CHECKER\_PIECE\_LOCATION** | pieceID: int  oldLocation: {x: int, x: int}  newLocation: {x: int, x: int} | Action to update a checker piece’s location |
| **UPDATE\_CHECKER\_PIECE\_LOCATION\_SUCCESS** | oldLocation: {x: int, x: int}  newLocation: {x: int, x: int} | Action to handle success of updating a checker piece’s location |
| **UPDATE\_CHECKER\_PIECE\_LOCATION\_ERROR** | Error | Action to handle error of updating a checker piece’s location |
| **SET\_KING** | pieceID: int | Action to set a piece to king |
| **SET\_KING\_SUCCESS** | pieceID: int | Action to handle success of setting a checker piece to king |
| **SET\_KING\_ERROR** | Error: obj | Action to handle error of setting a checker piece to king |

**Table 2. Action Type**

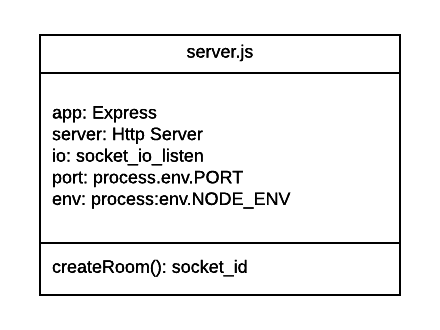
|  |  |  |  |
| --- | --- | --- | --- |
| **Resolver Name** | **Corresponding Action Type** | **Properties to update** | **Description** |
| initializeSocketError(obj) | **INITIALIZE\_SOCKET\_ERROR** | error | Set the error for socket initialization |
| setName(string) | **SET\_USERNAME\_SUCCESS** | username | Set the username |
| setNameError(obj) | **SET\_USERNAME\_ERROR** | error | Set the error for setting username |
| triggerWait() | **TRIGGER\_WAIT** | wait | Set the wait flag |
| setTurn() | **SET\_TURN** | isTurn | Set the turn flag |
| setColor(string) | **SET\_COLOR** | color | Set the color property |
| setNumPieces(int) | **SET\_NUMBER\_CHECKER\_PIECES** | numPieces | Set the number of pieces of the player |
| setWin | **SET\_WIN** | win | Set the win flag |
| setLose | **SET\_LOSE** | lose | Set the lose flag |
| triggerModalDisplay() | **TRIGGER\_MODAL\_DISPLAY** | shouldModalDisplay | Set the shouldModalDisplay flag |
| updateLocation(int, obj, obj) | **UPDATE\_CHECKER\_PIECE\_LOCATION\_SUCCESS** | board | Set new location for a checker piece in the board |
| updateLocationError(obj) | **UPDATE\_CHECKER\_PIECE\_LOCATION\_ERROR** | error | Set the error for updating location |
| setKing(int) | **SET\_KING\_SUCCESS** | board | Set a checker piece’s king flag to true |
| setKingError(obj) | **SET\_KING\_ERROR** | error | Set the error for setting a checker piece to king |

**Table 3. Reducer resolver**

**4.3 Back-end**

**4.3.1 Server Component**

The back-end server listens to people connecting to it and creates sockets for people to use when they get moved to the game room. This is always running and does most of the hard work when it comes to setting up and managing the games.



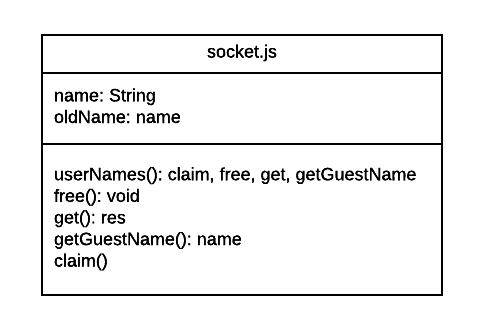
**a. Attribute**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| app | Express | Attribute for importing Express module |
| server | Http Server | Attribute for using http server object |
| io | socket\_io\_listen | Attribute for socket listener |
| port | process.env.PORT | Attribute for setting server port |
| env | process.env.NODE\_ENV | Attribute for setting production or development environment variable |

**b. Method**

|  |  |
| --- | --- |
| createRoom() |  |
| Input: | void |
| Output: | socket\_id |
| Description | Used to connect two users to a game session. This will populate the gameplay page. |

**4.3.2 Socket Component**



**a. Method**

|  |  |
| --- | --- |
| userNames() |  |
| Input: | void |
| Output: | claim, free, get, getGuestName |
| Description | Used for initiating and maintaining users. |

|  |  |
| --- | --- |
| free() |  |
| Input: | void |
| Output: | void |
| Description | Disconnects a user from the server |

|  |  |
| --- | --- |
| claim() |  |
| Input: | void |
| Output: | boolean |
| Description | After userName duplication check passes, the userName is claimed for the appropriate user and stored amongst a list of users. |

|  |  |
| --- | --- |
| get() |  |
| Input: | void |
| Output: | res |
| Description | returns serialized claimed names as an array |

|  |  |
| --- | --- |
| getGuestName() |  |
| Input: | void |
| Output: | String |
| Description | Finds the lowest unused “guest” name and claims it. |

**b. Attribute**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| name | String | Current user name |
| oldName | String | Used to validate user name change |

**5. Human Interface Design**

**5.1 Overview of User Interface**

The front-end shall contain a lobby page which allows users to start a game. The player would then get redirected to the gameplay page where two players can play a match. At the end of a match, the players would face a popup modal window which would allow the players to play another match or go to the lobby page.

**5.2 Screen Objects and Actions**

**5.2.1 Lobby Page**

The lobby page will have the game title, an ‘About’ link to navigate to the

‘About’ page, an input box to set the username and a ‘Start’ button to start

the game.

**5.2.2 About Page**

The ‘About’ page has version and patch notes, credits, game tutorial link,

and ‘Back to lobby’ button.

**5.2.3 Gameplay Page**

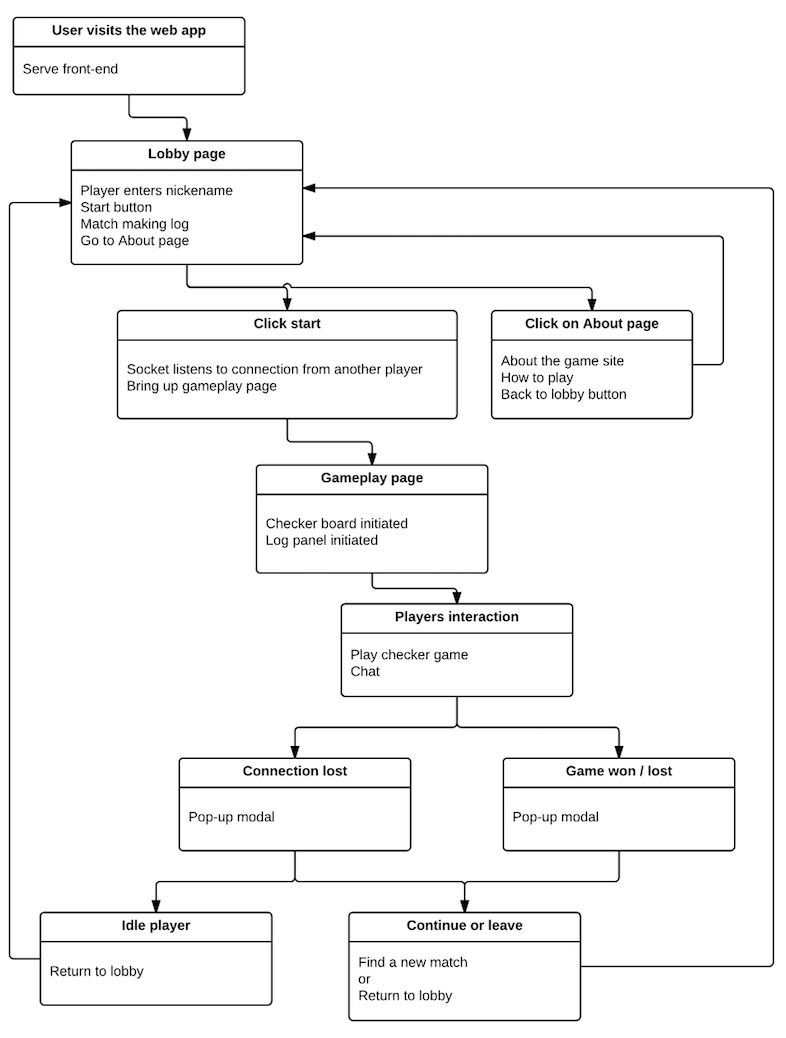
The Gameplay page has the checkers board with checkers pieces, a move log section, a ‘Give up’ button, and a popup modal after a game ends.

**5.2.4 Popup Modal**

The popup modal has two buttons: ‘Return to lobby’ and ‘Find a new

Match’.

**5.3 User Interface Flow**

****

**6. References**

http://gs.statcounter.com/browser-market-share#monthly-201707-201707-map