WEB GAME – CHECKERS

PROJECT REPORT

Course: Web Development

Term: Spring 2018

Submitted By

Elavazhagan Sethuraman
Rachna Reddy Melacheruvu

Contents

1. INTRODUCTION AND DESCRITPTION

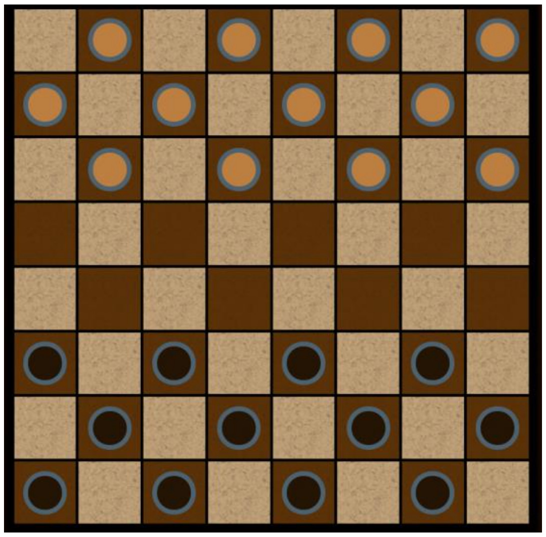
1.1 Introduction

This web game project aims to develop a multi-player game with multiple spectators on the web browser. It will include two opponents whose aim will be to diagonally jump over the opponent’s pawns and eliminate those pawns from the game. This project explores a new dimension in the traditional checkers game by allowing spectators to watch the on-going game.

The major objective of this game was to implement a server-side game with multi-players connecting on the same channel to play the game. The server-side logic is written in Elixir and the client-side logic is written in React.js. The client to server communication is done through Phoenix channels to set state and monitor player actions.

1.2 Description

The checkers board has 8 rows and 8 columns which have squares colored with 2 colors alternatively. There are 64 squares in total which are 32 light colored squares and 32 dark colored squares. Colors used in our project are dark brown and off-white squares. There are 12 pawns of each color, black and mustard. 3 rows on each side of the board are occupied by placing the pawns of each color. The pawns are placed on the dark colored squares. And since the game involves moving the pawns only diagonally, the pawns are always on the dark colored squares. The board at the start of the game looks as below:



This game allows 2 players to play against each other when both the players join the same game instance. The player to join the game first, is the first one to move the pawn. Each player can move the pawn diagonally one step ahead if the square is empty. If the diagonally adjacent square has the pawn of the same color, then the that pawn cannot move. If the diagonally adjacent square has a pawn of opposite color and the diagonally adjacent square to it is empty, then the pawn can jump-over the opponent's pawn and capture it. These pawns can only move diagonally forward in direction.

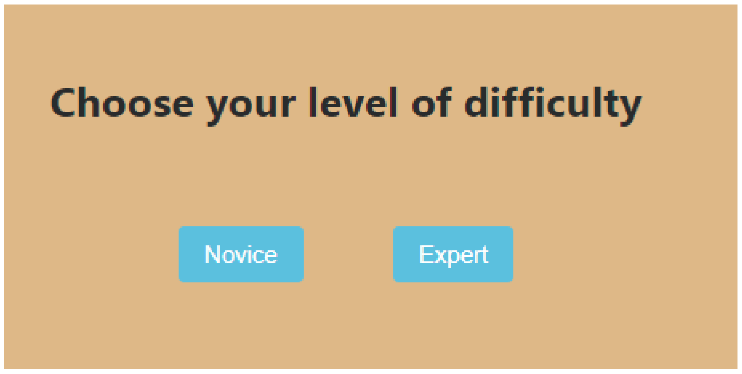
When a pawn reached the last row of the opponent's side, then the pawn gains "king" power. Which means, that from then on, king pawn may move diagonally forward and backward in direction. The king pawn is distinguished by other pawns with a green border around it.

1.3 FEATURES IMPLEMENTED IN THE GAME

- Choosing the difficulty level
- Chat room
- Unlimited audience (supervisors) for a game
- Reset on mutual consent

1.3.1 Difficulty Levels

There are two difficulty levels: novice and expert. In novice level, the players are guided on the pawn's anticipated moves. That is, on clicking a pawn, when it is valid, the expected squares that the clicked pawn can move to are highlighted. This way, players can make better decisions and erroneous moves are prevented. In Expert mode, the squares are not highlighted. Hence, players must think through on where to move the pawn. This would be more crucial when the pawns are kings. Because a king can move in all four sides diagonally and over the opponent pawns, the player can get distracted and hence needs to be more careful in making moves.



2 UI DESIGN

There are two main pages rendered in the front end. First, when the player goes in to the game website, an index page is shown up. Here, the user is prompted for a game name and user name. One entering the user name and game name, the player can click the submit button. Once the button is clicked, the game page is rendered. Since, checkers is a board game, we chose to develop UI with a rustic theme. Hence the game page has the following components:

- Logo
- Gameboard
- Scorecard
- How to Play? Button
- Chat box
- Game Status box
- Play music button
- Reset Button
- Exit button
- Special cases

Logo

The logo for checkers occupies the entire width of the page as a header. The label CHECKERS is displayed in between the two icons.

Gameboard

The gameboard is displayed as in the center of the page. This is a dominant component of the page. The gameboard is a rectangle shaped structure which contains 64 small squares in it. Each square refers to one of the actual 64 squares on the board. The squares are colored alternatively in burleywood and brown to construct an aesthetic checkers board.

Pawns

Each pawn is a filled circle. There are two types of pawns: red and black. The red pawns are placed on top half of the board and black pawns are placed on the bottom half of the screen. Initially, pawns are just filled circle. When a player clicks his pawn during his turn, the pawns are embossed and highlighted. When a pawn becomes a king, the pawn is filled with a crown inside.

Squares

When a valid pawn is clicked in novice mode, the squares that the pawn can be moved on to, are highlighted with a black square around. In an expert mode, the expected squares are not highlighted.

Scorecard

The left side of the page is the score card that displays the player names. A red pawn and black pawn image is placed inside the scorecard

Chat box and play music button

The chat box appears on the right side of the screen. It is a scrollable box. The play music button on click toggles the background music option. This works best when used in a google chrome or safari browser.

Game Status Box

The live status of game is displayed in the game status box placed at the bottom of the page. This is deliberately kept at the bottom so that the player can see the chat box and game status box in a closed vicinity.

Reset, Exit, how to play? Button

Clicking this button pops up the instruction description that guides the players about the game rules. Reset button enables the players to reset the game upon arriving a mutual consent. Exit button on clicking concedes the game and makes the other player winner.

Special cases

Initially, the board is replaced by an option allowing the user to select the difficulty level of the game. That is shown as buttons and on clicking one of the buttons, the players are entered in to the game with the game board rendered. Similarly, after the game is over, the board is replaced with winning status message and shows a button giving an option to restart the game.

3 UI TO SERVER PROTOCOL

The communication between UI and server happens through the phoenix channels. In this architecture, javascript (mainly ReactJS flavor) renders the game in the front-end. Backend server is the Genserver whose logic is written in elixir. Middleware component is the Game channel written in Elixir.

Identifiers for communication between UI and server

When there are multiple game instances, each instance is uniquely identified by the game name that the user has entered in the index page. These names are stored as an attribute in socket. So, when a call from UI to the server goes, the server rightly identifies the relevant game by picking up the socket that matches the invoked UI instance's game name from the pool of sockets.

Understanding sockets and channels

There can be multiple instances of games and many instances of a same game and in each of the game instances, there can be multiple instances (players and spectators of a game are the game instances). Each unique game instance has a socket associated to it. Any player entering a game establishes a channel through which she is connected to the socket of the game.

Broadcasting

The UI propagates information from the state maintained by Reactjs. Once the changes are successfully carried to the server from ReactJs layer to server layer through phoenix channels, they should be reflected on all instances of a particular game. This is achieved by **broadcasting** information through all the channels of the game socket. Once broadcasting is done, each instance of the game renders the updated game state of ReactJs. Hence, a change made in one player's UI is propagated to all the spectators and the other player.

For example, when a player moves a pawn on the board, reactJs invokes the game channel written in Elixir to handle this action. The game channel propagates this information to the server. This propagation is done though a simple elixir invocation as both layers are elixir implementations. Elixir updates the game state in the form of a map and send the updated map back to the channel. At this point, elixir broadcasts the updated state information to all the channels. As a result of broadcast, all the channels are updated and thereby updating the respective states maintained in the ReactJs layer. When the ReactJs states are updated with the new pawn positions which gets rendered on the board.

Saving the game state

In the middleware phoenix channel, whenever the server responds for the UI requests, the state of the game is saved. And the game is loaded from the saved game instance. This way, the players and spectators are ensured that each of their moves are safely backed up. So, when a player clicks a pawn and suddenly closes the browser by mistake, he can simply reopen the page with the credentials to get back his control on game. During this downtime of the player, no other player or spectators are disrupted of their rendering in the front-end. The last made change happened before downtime will also be broadcasted from the saved game state.

The Handshake protocol

In addition to the underlying protocols maintained between Genserver, phoenix channels and the javascript layers, we have enforced handshake protocol between the players to ensure there is a good level of agreement maintained. For instance, the difficulty level can be chosen only after the second player joins the game. The players can use chat option to decide upon the difficulty level and then they can choose to play either as an expert or novice. Similarly, the reset button is not meant for a hard reset. Instead, a reset button when clicked for the first time by a player, the other player should then respond by clicking reset button too. Only then, a reset will happen. Such constraints are enforced as per our choice of design.

4 DATA STRUCTURES ON SERVER

The data structure that is prevalently used on the server side is a hash map. Each game instance is stored in the GenServer in the form of Elixir map, where game name serves as the key. The game state is stored in the form of a map. It contains fields such as pawns, highlightedSquaresMap, first, second, nextTurn, redPoints, blackPoints, winner, clickedPawnObj, gameStatusMsg and resetStatusCode. Each serves its on purpose for rendering the game states in the front end appropriately.

Initial Pawn positions

To store the initial positions of the pawns, list is used. There are two lists one for red pawn and the other for black. These lists are a collection of integers, which maintains the position of that pawn on the checkerboard. **Example:** If the positions list of for black is [1,5] , then the initial state of the board has black pawns at 1st and 5th square of the checkerboard.

Pawns, ClickedPawnObj

The field pawns in the game map is itself another map containing two fields – red, black. As the name suggests, red is the inner map that stores all the information about the red pawns present in the board and black indicates all the information about the black pawns in the board. The field clickedPawnObj stores the information about the pawn currently clicked by the player. Information about a pawn is also stored in form of a map, where the key for the map is the square position at which the pawn is placed in the board. Each pawn information contains type of the pawn (red or black), position at which the pawn is placed on the board, whether the pawn is a king or not and its own id. The pawns are identified uniquely by the identifiers. Initially, on the board, there are 12 pawns on both sides and hence red pawns have ids from 0 to 11. Similarly, black pawns have ids 0 to 11. Thus, each pawn in the entire board is uniquely represented by its type and the id.

First, second, nextTurn and winner

The first player entering the game is assigned to the field first and the second player is assigned to second. The first player gets his turn to play the game first. So, in a fresh game, the first player always get accessibility to move the pawn. The field “nextTurn” can have two possible values – “red” and “black”. This tracks which type of pawns can be clicked. The field “winner” can have the following possible values – an empty string, name of first player, second player name, “draw”. As long as the winner field is an empty string, the game has not ended. When the value is “draw”, it means, the game has ended in a draw state where both the players are winners. In other cases, the player whose name is associated in the winner field is declared to be the winner.

Highlighted Squares Map

Another important choice of data structure is the highlightedSquaresMap. As soon as the pawn is clicked by the user, in the server, all the possible positions for the clicked pawn is computed. This information has to be retained for the player to move this pawn to one of the valid squares. Hence, this is stored in the highlightedSquaresMap of the game map. Each key in this map represents the next possible square to where the clicked pawn is stored. The value for each key represents the intermediate pawns present diagonally. These intermediate opponent pawns are taken out of the board when the clicked pawn is moved to the expected square position.

Points and Messages

The points for each player is stored as blackPoints and redPoints. RedPoints tracks how many black pawns have been removed by the red pawns thus far and blackPoints represents the vice versa. The field gameStatusMsg is of type String and is used to broadcast the information about the game at present. Server modifies this field appropriately and broadcast it to all the front-end instances of a given game.

Reset Status code

This field takes care of the handshake protocol implemented to reset the game on both the players arriving at a mutual consent. This field is an integer which can hold two values – 0 and 1. The default status code is 0. When the field is 0, the server does noting with regards to resetting. When the reset button is clicked by one player during her turn, the reset status code is updated to 1. So, now when the other player again clicks the reset button, the server handles the reset condition. The game state is set to the initial state and reset button is appropriately updated back to 0.

5 IMPLEMENTATION OF GAME RULES

Checkers is a multi-player game, where two players actively play and get to know about each other's progress. Hence the game rules are implemented accordingly. In our version of checkers, a luxury of the players choosing what kind of difficulty level they want to play in. So, the board is not exposed to the player unless another player also joins the game. When both the players join the game, one of the players can then choose the level of difficulty. There are two levels of difficulty – **novice and expert.**

Choosing Difficulty Levels

On choosing a difficulty level by any of the two players, the game board of that difficulty level is displayed on the screen, wherein the first player (red pawn) gets her turn. Spectators are not allowed to choose the difficulty level. The chat option allows the players to communicate between each other. That would serve as a medium of acknowledgement for the players to get a mutual consent on the difficulty level.

Moving the pawns

Once the difficulty level is chosen, the players and spectators are shown the checkers board, wherein the game is to be played. The first player always gets to start the game. The first player is associated with red pawn and the second player is associated with black pawn. First player cannot click and move the black pawns. Similarly, second player cannot move the red pawns. Spectators cannot move any of the pawns as they are not supposed to do so.

The pawns can only be moved diagonally. When there is an immediate diagonal square available for the pawns in the opponent's direction, the pawns can be moved to it. When the immediate diagonal square is occupied with the same kind of player's pawn, the pawn is said to be locked and cannot be moved in that direction any further. These are the base rules.

Scoring points

When there is an opponent pawn placed in the immediate diagonal square and the next immediate diagonal from that square is free (in the same direction), then the pawn can jump over the opponent pawn. When this happens, the opponent pawn is removed and the player whose pawn jumped over gets a point.

king power

When a player's pawn reaches the farthest row on the board, the pawn acquires a king power. The king pawns are distinct from other pawns visually by having a crown imposed on it. The significance is that king pawns have the power to move all four directions diagonally with the locking conditions same as discussed above for normal pawns.

Reset Control

At any point of time a player can choose to reset the game. Resetting the game is allowed only when the player has his turn. And when the reset is clicked for the first time, it is not a hard reset. Instead it will raise a reset request. The other player now has the option to acknowledge the reset request by clicking the reset button. The other player can ignore the reset request by simply clicking on his pawns as if his turn is normally played. Note in this case, when a player declines the reset request from other player, the other player gets penalized with a turn of him being lost.

End state for the game

There are three different win conditions possible – player1 wins, player2 wins or the match is drawn.

Win conditions

When a player has no more pawns to play, then she loses. Another condition is, when a player cannot move any more pawns of her while the other player can still make moves, she loses. Optionally, a player clicks a button called “Give up”, she chose to lose. Eventually, the other player wins. When both players cannot make moves (all pawns on the board are locked), then player with the higher number of kings wins the game.

Draw condition

When both players cannot make moves (all pawns on the board are locked) and they have equal number of kings, then the game ends in a draw.

6 CHALLENGES AND SOLUTIONS

This project posed us nice challenges to be dealt with both on the technical and decisive front. Design choices were crucial is constructing the system design, that we had to foresee many technical issues that would have occurred otherwise.

6.1 IMPLEMETATION CHALLENGES

Broadcasting the changes

Broadcasting the changes from one of the player's window to all the other players' window was a major challenge that we faced in this project. The initial intuition was to use a separate game supervisor that takes care of all these. However, after reading through multiple forums and phoenix documentations, it was understood that channels themselves can be used to broadcast the information between the game instances. A lot of understanding was required to get this function properly in our system. One interesting finding of this challenge was – loading game instance from the saved game state of server is more reliable than loading game from the socket directly.

Controlled access to players and spectators

Once communication was established between the instances of a game, the next challenge was to decide upon how the access control is enforced among the players. Also, spectators should just be passive contributors of the game and should not be given any controls except for joining the game. The solution was to associate the current window's player id with the player information in the current state of the game (first player, second player and next turn). And features such as reset, give up are permitted to the player only if he has his turn on board at that instance.

Computing End game state

Deciding if the game has ended or not was a beautiful challenge that we dealt with. When the opponent has no more pawns, then he loses. This is a straightforward case. But often, 60 percent of the checkers games ends when the opponent still has pawns but all the pawns have no position to hop on the board. Since our game state only holds information about clicked pawn and the possible squares of the clicked pawn, computing this condition was an interesting task. The solution was to proactively compute all the possible moves of all the pawns of the opponent once a move is made by a player. While computing this, if it is found that not even a single pawn has a valid position to move on the board, then the player holding the pawn loses.

Implementing Background music

The implementation of background music seemed straightforward. We spent time on choosing a background music that gets along well with the context and pace of the game. Once it is implemented and tested, we found that the audio option was working only in google chrome and not in mozilla firefox web browser. The reason is that firefox does not support mp3 audio files rendered through plain html css. When analysed further, the audio format OGG was found to be a ubiquitous audio format that worked in different browsers. However, we could not find a good OGG format music that served our need. So, we chose to settle with the mp3 file itself, though it does not work on firefox. Another concern with audio files was raised when we had to decide on how to invoke the audio file. We had two options – download the mp3 file and put it inside static folder and locally invoke it or refer to the URL of the mp3 fie from the web. We preferred the latter as the former suffers from overloading the phoenix project with extra space.

6.2 DESIGN CHOICE CHALLENGES

The reset functionality

Implementing game reset was simple once the concept of broadcasting between channels was understood. As per out initial design, just as a player clicking reset was resetting the board. But when examined the practical scenario, any two-player game would like to be informed before a reset happens. Based on the second player's consent, the reset was either enforced or discarded. So, a two-way handshake protocol was implemented for reset. As per this, when a player gets a turn, he can choose to reset. On reset button click, the control of the board moves on to the other player. When the other player also presses reset button, it means they have a mutual consent on resetting the game and hence the board is reset. If the second player is not interested in reset, he can simply ignore the request by choosing to click a pawn. This way the conundrum of reset was resolved.

The Exit Button

The consequence of reset button being acknowledgeable is that a player can never raise a new game request. That would make it uninteresting as the players are forced to continue the game they are no longer interested in. This problem led to the "exit" feature. On clicking this button during a player's turn, a confirmation popup will appear. On confirming give up, the player has conceded the game thereby leading to the game being ended with the other player declared as the winner. After this, the players are given an option to restart a new game.

Choosing the player id

The initial idea was to choose mail id as a player id as that would appear unique. But this is a game where players would likely be interested in playing the game without having to enter much of information. So, we chose the player name as the player id for simplicity purpose. Anyhow the game is unique from one another by its name. So, player's name on top of the already unique game name gives a good extent of uniqueness.