Brett Adcock

Binh Le

2/18/19

Webdev Proj. 1

Introduction and Game Description:

The aim of this project is to implement and deploy an online multiplayer version of the classic board game Othello for users to enjoy. The project will allow for multiple simultaneous games to be played in different game rooms, with the state of each unique game being saved throughout, thus allowing players to reconnect to open games. The game logic will be written server-side in Elixir with Phoenix being used as a web framework to serve the game. The front end will be rendered using React.js with milligram css styling. The game will then be publicly hosted online at othello.foolsparadice.com for anyone to play.

Othello is a 2-player board game that is played on an 8x8 board with 64 game pieces, called discs, which are white on one side and black on the other. The initial board setup contains two black and two white discs in a checker pattern on the center 4 spaces of the board. Players then alternate turns placing discs on the board with their respective colors facing up. During a turn, any disks of the opponent's color that are in a straight line and bounded by the disk just placed and another disk of the current player's color are turned over to the current player's color. If a player is not able to make a valid move, ie a move that will flip at least 1 of the opponent’s discs, then that player forfeits their turn until they are able to make a valid move. This process continues until neither player can make a move or the board is full, at which point the game is over. The discs are then counted by color, and the player with who has the most discs of their color facing upward is declared the winner.

UI Design:

To create an application that allows for multiple simultaneous games to be played, a landing page was created to first route players to an appropriate game room. In this landing, or index page, a user can create a new game room or view the list of games currently being played. From here, users can either enter into a newly created game or join a game that has another player waiting to play.

When a user joins a new game, they find themselves on a page displaying a game board with 4 discs (two black, two white) in a checkered pattern in the center 4 cells. The page was modeled off of a traditional Othello game board. The board itself is a non-checkered 8x8 grid that is green in color with faint division lines between cells. The page background is brown to simulate the feel of the board sitting on a table between players. Users can then hover over any empty cell and see a disc of their color faintly appear, to show that they have the option to attempt to place a disc. This hover functionality does not check to see if the cell being hovered over is a valid cell for the user to place a disc, only that the cell is empty.

A header at the top of the page is also visible to users with a status about the state of the game. This status could be (1) waiting for opponent to join, (2) your turn, (3) opponent’s turn, or (4) displaying the winner and game score. This header was implemented to help keep the game flowing. Users are explicitly informed of actions they need to take, instead of relying on them to have awareness of the state of the game. This was an important choice for users reconnecting to a game already in progress.

UI to Server Protocol:

The UI to server protocol is handled by the use of both channels and a GenServer. The state of the game is stored server side as a in a GenServer called GameServer. This GameServer will create a registry of all games and be updated every time a move is made in any game. The front end (React.js component) communicates with this GameServer via channels. An action is taken on the user end that alters the state of the game, which sends a message on a particular channel. The specific channel depends on the exact user action; generally this is a user click, which sends information about the click to the GameChannel package. From here, the message routes the payload the appropriate handler, which is responsible for calling the appropriate game logic, storing the new game state in the registry, then broadcasting that state change back to all users connected to the game. The broadcast functionality is important as it allows all browsers connected to the game to be updated when the game state changes as opposed to a channel and backup agent approach which would only update the user that initiated the state change.

In this Othello implementation, channels are provided for (1) joining a game, (2) handling a browser click, and (3) broadcasting updates from the server. These communication protocols provide a complete interface for communicating with the server and synchronizing all sessions connected to a single game allowing multiple users to share a single game state over multiple browsers.

Challenges and Solutions: