6Holes

Authors

- Anagha Bhosale
- Ruchit Urunkar

Introduction and Game Description

6Holes is a two player game. The game starts with 2 players, each having 6 cards, 3 cards in each row respectively. The players are able to see only two cards at the start of the game. The column two cards are opened initially for both the players.

The goal of each player is to get minimum possible score. The score is calculated based on each column. If two cards in a column match, the score evaluates to zero, the sum of scores of each column contributes to an overall score.The player can device stratergies to discard a card which is of no use to the opponent. The middle card can be swapped at any moment unless the columns does not count to a score of zero.

Thus at each consecutive turn the player tries to get minimum possible card, also the side cards belonging to first and second column can be swapped only if they are not opened, once the cards are unveiled they cannot be swapped.

The game can be joined by multiple people. Incases if any such additional players joining the game, they can join and chat in the game, but wont be able to play the game. The person who joins the game first becomes the first player, and the latter becomes the second player.

The game ends when all the side cards of one player are unveiled. The other player gets one turn to draw a card and decide, whether he wants to swap with any unveiled cards or discard. If the player wishes to discard the card, then the all the unveiled cards are opened and score is calculated and new round is started with the accumulated score.

UI Design

The start page is a form that takes below input from the user. * Name -> Name of the user * Table name -> The table which the user wishes to join.

The users must enter same table names, to join the same table. The field is case sensitive, thus it is mandatory to have the exact same table name. On clicking the submit button the link to join the game is enabled. The user can now join the game by clicking on the link.

On each move user can either open a card, swap a card from the deck, or use the previously discarded card. Incase if the user wants to open a card he can simply click on open the card button, followed by the card he wishes to open.

The user can draw a card from the deck and swap it with any of the hidden cards by simply clicking the deck to draw a card and then click on the card he wants to swap it with. The user can discard the card without swapping it by clicking on discard and the turn shifts to the next user.

Once all the side cards are opened they cannot be swapped and the game ends. Once such situation occurs for a player the next player has only one move in which he can open the card or pick a card from deck and swap it.

Once the round ends both the users are required to reload/refresh the page to enter in the next round. The game board is now visible to the user, with initial score based on column two cards that are initially open. The second user can join existing game only if the table name entered by him is correct, else he will be directed to a new game.

The game UI is desgined using images which are created usings tools such as Microsoft Office and paint. The image of the joker card has been derived from https://hexdocs.pm/phoenix/channels.html . The images are rendered correctly using file-loader, url-loader.

UI to Server Protocol

The interaction from the user are interpreted as get request, for each consequent action a respective route is defined in the router file that directs it to the correct location.

In our implementation the initial screen is a form which expects two values from the user. The user name and table name once input, creates a link which redirectes to xholes application. The routing is done with the help of router.ex file. The router directs it to the page conroller which lands us to the application.

The application renders and set the state using channel join. Once a channel is joined any action on the screen is captured and sent on the channel. From the channel it is forwarded to Game_server(implementation of genserver) which takes the appropriate action and sends updated state to the backup agent.

The different actions such as cardDrawn, changeTurn, resetDiscarded, resetDrawn, updateScore, swap, cvOpened take care of the change in the events and are sent to the gensever and then to elixir which updates the state and send back the updated state which is backed-up in the backup agent and then broadcasted so that the change in the action is visible to all the users connected to the table.

The implementation also uses Dynamic Supervisor, so that its starts and monitors the processes and restarts them whenever there is a occurence of error or issue in the execution, this allows the userscreen being handled in the crash state.

Data structures on server

The server side codding has been done in elixir. The data structures used in the context of the game are mainly for maintaining the state and performing

manipulation on the state so that the corresponding changes are reflected on the ui.

The state of the game is stored using maps, the reason for same is that the maps allow us to access the state of the game using key: value pair. Maps allow us to store list, integers, strings together thus keeping the state intact. Elixir does not allow mutation thus change of state through unfair means will not be allowed here and will be done only through channels.

The state stores the name of player1, player2. The values associated with each card and the sequence list is maintained and shuffled in elixir. It also saves the cards drawn, the turn of a player which is binary flag, list of values associated with cards, p1score, p2score, the prev holds the reference to drawn card which is not discarded, the discarded holds reference to drawn card which is discarded.

As the deck comprises of images, we have list to store the images representing the cards which are imported in the react. Enums are also used in implementation as they allow to work with large number of datatypes. Enums have helped in easy traversing of list for shuffling the deck of cards, maps.

Implementation of Game Rules

Below are the set of rules which allow the game to played in the correct sequence.

- The player who joins the game first gets the first turn to play, and the user two is not allowed to play or do any action.
- The player can draw a card from the deck, or open a card from the deck.
- If the player draws a card then he is left if two options, he can either swap the card with the cards that are hidden or he may discard the card.
- The player1 should not be able to open player2 cards, or play his turn.
- In scenario such as player 1 discards a card, player 2 opens a card then the player 1 is not allowed to use his previously discarded card.
- The cards present in the column two can be swapped at any time in the game, unless there is atleast one card hidden.
- The side cards i.e cards present in column 1 and column 3 can be swapped only if they are hidden.
- Once all the side cards of one player are opened the other player is allowed to perform only one turn and then starts the next round.

We have implemented the rules using channels for each action. The user who joins the table first becomes the 1st player, and the second person who joins the table latter becomes the 2nd player.

The rest of the player become the observer and can only send messages on chat. The player1 is allowed to play his turn and the player2's board is disabled. The player1 can then perform his set of actions and on each corresponding open card button click the player needs to click on the card which he intends to open. This triggers a handlein cvOpen which opens the cards and updates the state.

Once the card is opened the player1's view is disabled and player2 is allowed to perform his set of action. For player2, player once cards are disabled, now he can either open the card or draw a card and swap. If player2 draws a card he can swap it by clicking on the card he wishes to swap and this is handled using handlein swap. Once the card is swaped the turn shifts through handlin using turnChange.

Challenges and Solutions

The challenges faced during the implementation were mainly rendering of the images in react. We solved the problem by importing the images and using url-loader and file-loader.

The area where the user can join a game was tricky for us. The reason being we were unsure about how one should pass the name of the user within the channel and set it in the state. We solved this issue by refering Professor Nats recorded lectures. Professor suggested us that we could use a form for collecting the data and send it in the channel. After reading online documentation on Phoenix we realised a way and sent the username in payload and initialized it in the new game.

In the initial phase, understanding the action and its side-effect where difficult to implement, trial and error method helped us to understand how things can be done.

We faced challenge even when we were trying to move the score to next round. We resolved the same using a handlein and thus solved the issue.

In the chat implementation we faced issue where in the message was getting broadcasted multiple times and we were not able to understand the issue with the multiple calls. We thought it of implementing it using a click event rather than keypress.