Multiplayer Aeroplane Chess Design Report

Authors: Xinming Dong, Weihan Liu

1. Introduction and Game Description

This web application implements a multiplayer Aeroplane Chess game with a browser interface. This application allows the users to play the same game on different browsers and interact with other players in the same game room. The server-side logic of the game was implemented using Elixir and Phoenix framework, and the client-side logic was implemented through ReactJS/React-Konva. This application is deployed online, and it can be accessed here.

Aeroplane Chess is named by the unique airplane-shape icons on the board, and it was getting popular in China in the 1990s due to its easy-to-understand but full-of-interactions rules.

This game can be played by 2-4 players and each player has its color chosen from yellow, blue, red and green. The objective of the game for each player is to move all of the four pieces of their color from the "camp", which is located at the corner of the board, to the ending point at the center of the board. A piece can only be moved out of the "camp"(launch) by a roll of six, and after it is launched, it can be moved on the board clockwise around the track with the number of spaces indicated by the die. Also, any roll of six, whether it is used to "launch" or move a piece, gives the user another roll. However, if a player rolls three sixes in a row, the third roll can not be used to move pieces and the pieces moved by the previous two rolls will be sent back to the camp. These moved-back pieces should act the same as the other pieces in the camp, which means they need a roll of six to launch again [1].

The track around the board is colored by yellow, blue, red and green as well. When moving in the track, if a piece lands on a space of its color, it can "jump" to the next space of its color in the current round without additional roll. There are also four spaces on the tracked which are marked as "shortcut", and each color has one "shortcut". If a piece lands on the "shortcut" of its color, it can move forward by 12 spaces immediately(shortcut jump). A regular jump can be followed by a shortcut jump, but a shortcut jump cannot be followed by a regular jump.

There are some additional rules which add interactions between players. If a piece lands on a space currently occupied by the opponent's piece(s), all these opponent's pieces will be sent back to the camp immediately, and can only be launched again by a roll of six.

A piece does not need to be moved into the ending point on an exact roll. It can fly into the ending point by any roll equal or larger than the remaining number of spaces between the current position and the ending point. When the first player moves all the four pieces into the ending point, he/she become the winner of this game and ends the game.

In this report, we will illustrate the frontend and backend design of this web application in the following four sections, as well as the challenges we came across in the development process as well as the solutions.

2. UI Design

From the perspective of users, there are two pages. Each of the two pages has a head that indicates the game name, which is called "aeroplane". The color of the game interface is mainly blue and white.

Every user can join a game table by providing a table name and a player name on the main page. This will bring the user to a game board page, where users can either play a game or observe the game. On the first page, users can input any word as table names and player names. If a table or user in the specified game table does not exist, the game server will create it for the user, while the game will load history game status if it exists.

For the game board page, the interface consists of two parts. One is the chessboard, and the other is a chat room.

In the chessboard part, a user can join the game by clicking the "Join Game" button, and the first player who joined can start the game by clicking the "Start" button which shows up when two or more players have joined in. Furthermore, a "Restart" button will come out when one of the players wins. As an initial view of an active game, every piece is located at its camp, usernames are displayed at the camps according to the order that players joined, and a die is placed next to the yellow camp displaying "Roll a die", which means the user that occupied yellow camp is the first one to roll a die and make moves. As players take their turns, the die will move to the camp of the current player. If a user doesn't have a 6 on the die and cannot make any move, the die will keep staying at the user's camp for a while to notify the user of the number before moving to the next camp. Moreover, there is also a message area, which shows the necessary information for players. When the game is active, this area states which color should take the turn, while the winner's name will show up when the game is over.

In the chat room area, which is located on the right side of the game interface, users can see all the history messages in the text area on the top, and send their message by inputting text in the box at the bottom and click send next to the box. Not only the player that actually plays the game but also observers can send messages to the table's chat room. Messages will be shown with a format "username: message".

3. UI to Server Protocol

Each time when the game's interface gets a user click on components, the browser needs to send a message to the server through channels and get a response so as to display a new view for users.vIn an aeroplane game, the user interface

communicates with the game server when users click the join button, start button, the restart button, pieces, die, and the send message button.

3.1 Join Button

When a user actively joins a game, the server needs to know which player joined the game and tell the browser which color is assigned to the user. On the user interface side, the username that assigned to color should display near the camp of that color. Therefore, the user interface sends its table name and a user name that clicked join, and the server sends back an updated map of users who have already joined the game together with their joining order, in which order of user name indicates the color that user is assigned.

3.2 Start Button

Given that no users can join a game that is already started, the server has to disable the "can join" and "can start" flag in the browser, so that no user or observer can click these buttons. The browser sends the table name and game name to the server through a channel, and the channel makes sure the user that clicked the "Start" button is the first player joined, set "can join" and "can start" flag, and send the whole game state back.

3.3 Restart Button

With a click of the restart button, the user interface needs to be set to initial view, expect for history chat message. This operation expects the server to send an initial state back to the browser. Therefore, the browser makes a request to reset the game state through channel, and the server sends the reset state back.

3.4 Pieces

When a user intends to move a piece, there are multiple ways that the move displays, since a piece can jump at the first time it lands at a grid which color is the same as the piece, and it can fly through a bridge, or send another piece back to its camp. Therefore, the browser tells the server the identification of the piece that the user intends to move, and the server sends back a series of states, each update the piece's location by one step. And the channel sends these states back to the browser one by one with time delays between them. In that sense, users can clearly watch the piece make jumps.

3.5 Die

Clicking a die makes the number on the die change, and when a user who is not the player to take turns clicks the die, the user interface view should not have any change. In this response, the user interface implies the username that clicks the die, and the server updates the number on the die in the client view state.

3.6 Send Message Button

When users type in their message and click on send, the message should appear at the bottom of the message displaying area. Furthermore, when a user just joins the table, all the history messages will appear in that user's interface. Thus, the browser sends the username and the message sent to the server, and the server will send the updated game state with all messages as a list back.

4. Data structures on the server

On the server side, the game is represented by changing the location of pieces according to the game rules. There are four major elements in the game: color, the spaces on the board and their attributes, the pieces, and their locations, and the die. The data structures of these four elements are illustrated below. The structure of the game state, as well as the client view, integrated all these four elements, and will also be explained in the following sections.

4.1 Color

There are four colors in the game, which are yellow, blue, red and green. They are represented in the server as atom : :y, :b, :r and :g.

4.2 Die

The die is represented as a single integer in the server. When users roll a die, the server generates a random number from 1 to 6 and assigns it as the current die number.

Since users cannot move any pieces at the beginning until they roll a 6, we decide to increase the possibility of 6 at the beginning of the game for each user. At the beginning of the game, the possibility of rolling a 6 increase to 37.5%. When at least one piece is launched, the possibility for each number will be the same (16.67%).

4.3 Game Board

There are a total of 95 spaces on the board, including the camps, the track, the ending points as well as the bridge to the ending points. These spaces and their attributes are stored in the server as a Map for easy retrieval. Every time the user moves a piece required the server to check the type and color of the landing space. Therefore, using a map increases the efficiency of this application.

The keys of this map are the id of the spaces, and values are lists of color and type. It means the element in the map is represented as "id => [color, type]". For example, the top-left space of the yellow camp is defined as space number 0, and it is represented as "0 => [:y, 0]" in the board map.

We defined 6 types of the spaces in the game and label them with numerical type id 0 - 5:

Type 0: spaces of four camps. These spaces are located in the four corners of the game board, and they are where the pieces initially located. Pieces located in this type of space can only be moved on a roll of 6. There are a total of 16 spaces in this type and four for each color.

- Type 1: starting spaces. On a roll of 6, the player can choose to launch a piece that is currently in the camp, and the launched piece will locate in this starting point waiting for the next round to move. There are a total of 4 spaces in this type and one for each color.
- Type 2: turning spaces. After these spaces, pieces will turn into the bridge area and heading to the ending point. Only pieces of the same color can make turns in this type of space. There are a total of 4 spaces in this type and one for each color.
- Type 3: shortcut starting spaces. If pieces of the same color land on this type of spaces, it will immediately move 12 spaces forward without additional rolling of the die. There are a total of 4 spaces in this type and one for each color.
- Type 4: Bridge spaces. This type of space is the connection between the ending point and the track. Only pieces with the same color of these spaces can enter this area, and there is no jumping inside the bridge. This type includes the ending point. There are a total of 24 spaces in this type and 6 for each color.
- Type 5: spaces on the track that are not type-2 or type-3. Pieces lands on these spaces of its color can jump to the next space on track of its color. There are a total of 44 spaces in this type and 11 for each color. It is worth mentioning that the ending points of the shortcuts are also included in this type.

To locate the pieces in the UI, the server also stores a map of the space id and the corresponding x, y coordinate. The keys of the map are ids of the spaces, and the values are maps represents the coordinates. For example, space number 0 are represented as 0 => %{x: 198, y: 198} in the board coordinates map. This map is only be used when sending the client's view to the front end.

4.4 Pieces Location

There are a total of 16 pieces shown in the game. If there are less than 4 players in the game, the remaining pieces will stay in the camp for the entire game. The locations of the pieces are represented as a map. The keys are the Atoms represent the color. The values are a list of 4 integers, indicating the located space id for the four pieces of this color. For example, at the beginning of the game, the yellow pieces reside in the yellow camp, which has space id 0 - 3. Therefore, the 4 yellow pieces are represented as y: [0, 1, 2, 3] in the map. This structure allows us to easily located the clicked pieces, and update its location accordingly.

4.5 Game State

The game state is the data structure that represents the entire game. It includes all the changing elements in the game which are needed to check when implementing the game logic. The game state is represented as a map, which includes the following things as keys:

- Current die number: record the number of rolled dice, and to the piece accordingly.
- Last two rolls: this records the last two roll numbers for each player. It is needed for the special continuous-three-6s rules mentioned above.
- The last two moved: this logs the last two moved pieces for each player. It is also needed for the continuous-three-6s rules.
- Moveable Pieces: This is the list to store all the id of moveable pieces. It is set to empty when the user is supposed to click the die, or there is no moveable piece for the current player according to the game rules. If the current player rolled a 6, it is set to all the 4 pieces of this player. Otherwise, if the current player rolled another number, it is set to all the pieces of the current player that are neither in the camp nor at the ending spot.
- Current player: this keep check of who should be the current player
- Next player: this keep check of who should be the next player. If the current player rolled a 6, but not a there-six-in-a-roll, the next player will be set as the same as the current player. Otherwise, it will set as the next player in the player list, according to the number of players in the game.
- Die active or not: when the player is supposed to click a piece, this variable is set to 0, otherwise, if the player is supposed to click the die, this variable is set to 1.
- Winner: this variable is set to empty string if the game is still in process. When one player completes the game, this variable is set to that winner as a string of his/her color name.
- Game active or not: This variable is set to 0 when the user joined the game room but did not press the start button. After the start button is clicked, this variable is changed to 1. If the game is not active, none of the players can click the die or the pieces.
- Can start or not: This variable is set to 0 when there are less than 2 users join the game(by clicking the join button). When at least 2 users joined the game, it is set to 0.
- Users: This is a map which records all the user in this current game. The keys are the usernames and the values are the user index/id. When a user enters the room, his/her username is added to this list with the id starting from 4. When a user clicked the join button, his/her user id will be changed into the smallest, not occupied id from 0 to 3. Therefore, the user with index 0 - 3 are game players, and other users are observers in this game room.
- Message: This is a list of string which logs all the messages sent by the users in the game chat room.

4.6 Client View

The server sent data to the frontend through a map. This map including the following elements which are needed in the client view:

- Current die number: represented as an integer
- Current player: represented as an atom
- Piece locations: represented as a list of maps. The keys are piece id (rang 0 - 15) and the values are maps of x, y coordinates.
- Winner: empty string if there is no winner yet, color name of the winner if winner exists.
- Game_active: game active or not(0 or 1). Same as the game state.
- Can_start: can start the game or not (0 or 1). Same as the game state.
- User_map: map all the users who joined the game (that are users with index 0
  - 3). Keys are usernames, and values are the user id.
- User_name: the current user who is taking action. This allows the server to send back the username that is transferred into the server through the socket channel.
- Message: list of string representing all the messages sent in the game chat room. ### 5. Implementation of game rules

5.1 Functional implementation

5.1.1 Launch only on a roll of 6

Since the game state keeps checking a list of moveable pieces. This rule is implemented by changing the moveable list. After the user clicked the die, the server generates a die number. If the die number is 6, the moveable pieces are set to all the pieces of the current player which are not located in the ending points. If the die number is not 6, the moveable pieces are set to all the pieces of the current player that are not located in the ending points and not located in the camp. Therefore, although they can choose not to, the user can only move the pieces in the camp on a roll of 6.

5.1.2 Additional roll on a roll of 6

This rule is implemented by setting the next player variable in the game state. On a roll of 6, the next player is set to the same of the current player, otherwise, it will be set as the next player in the list. After each turn, the server switches the current player to the next player that set beforehand.

5.1.3 Return to base camp in the three-6s-in-a-roll situation

The game keeps track of the previous two rolls for each player. Whenever a play rolled a 6, the server will check if his/her previous two rolls are also 6. If it is, the server will go through the “last two moved” list for this player (stored

in-game state) and move the last two moved pieces back to its initial locations, which are the base camp.

5.1.4 Jumping forward when lands on its color and shortcuts

After the valid user clicks a moveable piece, there will be three location changing actions happen in the server:

Firstly, clicked piece location will be changed according to the current die number and become the moved location.

Second, the server will check the attributes of this moved location. If it has the same color of the clicked piece and has a type of 5(normal track), it will move the location 4 more steps forward. If it has the same color of the clicked piece and has a type of 3(shortcut), it will move the location 12 more steps forward. This new location is the jumped location.

Thirdly, since a regular jump can be followed by a shortcut jump, the server will furthermore check the attributes of the jumped location. If it has the same color of the clicked piece and has a type of 3(shortcut), it will move the location 12 more steps forward. This location will be the final location for this clicked piece in this round.

5.1.5 "Bring down" the opponent's pieces when lands on them

After the location changing actions mentioned above, there is one more action that will happen after a piece is clicked: The server will check the entire piece location list for all users. If there are any pieces already in this location and have a different color from the current player, their locations are changed back to its initial location, which is the base camp of its color.

5.1.6 Winning the game

After checking the opponent's pieces mentioned above, the server will check if all the pieces of the current player located at its corresponding ending point. If it is, the server will announce the winner(by setting the winner's name in the game state) and disable all the clicks. Otherwise, the server will change the moveable pieces list to empty and disable any click on the pieces. Also, it changes the die to active to start the next turn.

5.2 Non-functional implementation

5.2.1 validate user for each click on die/pieces

Since this is a multiplayer game with unified views for every player, we need to verify the user when receiving every on-click calls. This is implemented by taking in user name as input for both the click die and the click piece function. Before doing any actions, the server checks if the user who performs the click

action is the same as the current user in the game state. If it is not, then return the input game state without any changes. This can prevent users to do actions on other's turn.

A similar logic is applied to check if the game is active. If it is not, no one can click on the die or any pieces, to maintain the initial state before the game starts.

5.2.2 showing the intermediate states

According to the previous sections, there are several stages when a user clicked a valid piece. For a user-friendly purpose, instead of showing the final location of the clicked piece immediately after it is clicked, we would like to show the process of the move, jump, and battle before the pieces reach the final location in one turn.

Therefore, the click piece function in the server will return a list of game-state, each representing the location of the pieces from different processes: move, regular jump, shortcut jump, and pieces battle. These game states will be handled in the channel and shown to the UI one by one with a short timeout in between. This implementation provides a better user experience and allows the players to understand what actually happened before their pieces land on a specific location.

6. Challenges and Solutions

6.1 Preloading Images

Rolling a die in this aeroplane game is implemented by changing images. Each time the server updates the number on the die, an image is loaded accordingly. However, loading images in react needs time, so each time the game view loads an image that has never shown before there is a time delay, and the die does not show up.

To solve this, we used a preload function in the HTML page, making all the images preloaded to local, so that the React would not need any extra time for loading. In the preload function, we defined a list of image file paths and a list of image objects and set source attributes of these image objects by traversing all the file paths.

6.2 Changing Die Position

It is easy to keep a die stay at its previous location before it moves to the next player by using "setTimeOut" in the client's view. However, when clicking a die comes together with clicking pieces, the situation gets rather complicated. Clicking a die or a piece can result in different player to take the next turn, and if users can click pieces and dice at any time, the final location of the die is dependent on which update happens latter.

On the client-side, to solve this problem, we used a flag that indicates whether users can click components on the chessboard. When a die is clicked, the flag is set to false, which means the user can neither click pieces nor click the die. And after the time delay of the die position update, the flag is changed back to true, enabling users to click other components.

6.3 Passing User Names to Server

To make sure that only users who joined the game can click on the chessboard, the channel needs to identify which user is interacting with it and report the username to the server. In that sense, usernames should be recorded.

Our solution is to get usernames from users' test input in the main page and pass it to the channel as a payload when the client view and react components are initialized. The username is also assigned to the socket as an attribute. Therefore, each time when channel call functions through the game server, the channel can get usernames from sockets and pass the names as arguments.

6.4 Pieces overlap issue

During the testing, we find out that when more than one piece of the same color lands on the same space on the board, these pieces overlap with each other and can not be clicked. Also, since all the pieces are identical in size, when they overlap, it is visually impossible for the user to tell there is more than one piece at that spot. This situation happens very often since all pieces of the same color are starting from the same spot.

To solve this issue, the server implements a function that separates the overlapped pieces a little bit to allow visual identification. The server logic is using the square ID to record the locations, and it is unreasonable to apply several IDs to the same square on the board. Therefore, this function is applied at the point when converting the location ID to the actual coordinates before sending the client view.

This function check duplicates in the location id list for each color. If duplicates are found in the list of the same color, the coordinates will be set as the original coordinates of that space, plus or minus a small offset. This implement provides an easy solution to this issue.

7. Conclusion

This web application implements the basic rules of the traditional Aeroplane Chess board game. It also successfully supports multiple players to play together from different browsers and different locations at the same time.

More functionalities can be added to this application in the future. For example, some version of the game allows the player to stack their pieces if they land on pieces of their own color. The stacked pieces will move together on the board until they reached the ending point or they are "bring down" by other

players' pieces. Moreover, some game rules indicate that when a piece lands on opposing pieces, players roll a die and who got higher roll can set the other's pieces back to its base camp. [1] Due to the complexity of the game rules, future implementation of the game can allow the user to choose which rules to apply before starting the game, this can adapt the game to different versions and provide richer experiences to the users.

References

1. Wikipedia. (Sep 2019.), Aeroplane Chess [Online]. Available: https://en.wikipedia.org/wiki/Aeroplane_Chess [Accessed: Oct, 2019].