**Notes for Project**

Yes, you can design your backend API to manipulate game results based on certain conditions, even if the underlying random number generator produces different values. Here's a basic approach to achieve this:

1. \*\*Generate Random Number\*\*:

- When a game round starts, your backend generates a random number using your chosen random number generation algorithm.

2. \*\*Modify Result\*\*:

- After generating the random number, your backend can apply custom logic to modify the result based on predefined conditions. For example:

- If the random number is 3 and you want the result to be 2, you can check for this condition and adjust the result accordingly.

- You can implement this logic in your backend code using conditional statements or lookup tables.

3. \*\*Return Modified Result\*\*:

- Once the result is modified, your backend API returns the modified result to the frontend, which then displays it to the user.

Here's a simplified example in pseudo-code:

```python

# Backend API endpoint to handle game logic

@app.route('/play\_game', methods=['POST'])

def play\_game():

# Generate random number

random\_number = generate\_random\_number()

# Apply custom logic to modify result

if random\_number == 3:

result = 2

else:

result = random\_number

# Return modified result

return jsonify({'result': result})

```

In this example, if the random number generated is 3, the backend API modifies the result to be 2 before returning it to the frontend. Otherwise, it returns the original random number as the result.

You can expand upon this approach to implement more complex game logic and result modifications based on your specific requirements and conditions.

To achieve this, you need to consider both the number of users joining each option (e.g., green or red) and the total amount of points wagered on each option. Here's a general approach:

1. \*\*Track User Selections and Wagers\*\*: Store information about each user's selection (e.g., green or red) and the amount they wagered. You can maintain this information in your backend database.

2. \*\*Calculate Total Wagers\*\*: Calculate the total amount of points wagered on each option (green, red, etc.) by summing up the wagers of all users who selected that option.

3. \*\*Determine the Winning Option\*\*: Compare the total wagers on each option to determine the winning option. In your case, if the total wager on red is higher than green, but green has more users, you may need to adjust the logic to prioritize the winning option based on either total wagers or the number of users.

4. \*\*Return the Result\*\*: Once you determine the winning option, return this information to the frontend to display to the users.

Here's a simplified example in pseudo-code:

```python

# Backend API endpoint to handle game logic

@app.route('/calculate\_result', methods=['GET'])

def calculate\_result():

# Retrieve user selections and wagers from the database

green\_users = get\_users\_selection('green')

red\_users = get\_users\_selection('red')

# Calculate total wagers on each option

total\_green\_wager = sum(user['wager'] for user in green\_users)

total\_red\_wager = sum(user['wager'] for user in red\_users)

# Determine the winning option

if total\_green\_wager > total\_red\_wager:

winning\_option = 'green'

elif total\_red\_wager > total\_green\_wager:

winning\_option = 'red'

else:

# Handle tie scenario

winning\_option = 'tie'

# Return the result to the frontend

return jsonify({'winning\_option': winning\_option})

```

In this example, the backend calculates the total wagers on each option (green and red) and determines the winning option based on the total wager amounts. You can adjust the logic as needed to prioritize the winning option based on your specific requirements, such as considering the number of users as a secondary factor.