WordleBot - Implementation Choices

#Additional Functionalities

I have added a reset functionality to the WordleBot game. The reset button will only appear if one of the following conditions is met:

- The game is over, indicating that the user has won the game by correctly guessing the word.
- The user has lost the game by reaching the maximum number of chances without guessing the word correctly.
- An error has occurred during the game.

In any of these cases, the reset button will be displayed, allowing the user to reset the game and start again.

#WordleBot

In my implementation of the WordleBot component, I made several choices to ensure its functionality and user experience.

- First, I used the `useState` hook to manage the game's state variables. These variables include `currentChance`, `clueCode`, `apiResult`, `cardData`, `loading`, `error`, `isWinner`, and `initialLoader`. By using hooks, I was able to update and track these states throughout the game.

- I integrated an external API by importing the `fetchWordleResult` function from the "../api/api" module to interact with the Wordle game and fetch results. This allowed me to make API requests based on user inputs and obtain responses accordingly.

- To handle specific lifecycle events, I utilized the `useEffect` hook. For example, I used an effect to reset the `clueCode` state whenever the `apiResult` changed. Additionally, I created an effect that simulated an initial loading state by setting `initialLoader` to `true` initially and then updating it to `false` after a delay of 3000 milliseconds.

- I employed conditional rendering throughout the code To dynamically render components based on game conditions. This allowed me to display different elements based on factors like `isWinner`, `currentChance`, and `error`. For instance, I rendered the initial loader (`CircularProgress`) when `initialLoader` was `true`, and displayed game cards, the color palette, submit and reset buttons, winner and error messages based on the game's state.

- For modularity and code organization, I created separate components such as `CardDisplay` and `Palette`. The `CardDisplay` component handled the rendering of individual game cards,

while the `Palette` component managed the color selection interface. This approach improved code reusability and maintenance.

To achieve a visually appealing UI, I utilized components from Material-UI (`Button`, `Typography`, `CircularProgress`) and applied custom CSS styles. This ensured a consistent and visually pleasing design, with layout adjustments made using classes defined in the "WordleBot.css" file.

#Palette

While implementing the `Palette` component, I made some choices to ensure its functionality and visual representation which are mentioned below.

- First, I utilized the `useState` hook to manage the state of the component. The `selectedBoxes` state variable is an array that keeps track of the selected box index for each column. I initialized it with an array of `null` values.

- To handle the click event on a box, I implemented the `clickHandler` function. This function updates the `selectedBoxes` state with the new selected box index. However, this update is only allowed if the component is not in a loading state, indicated by the `isLoading` prop. If `isLoading` is `true`, the click event is ignored.

- To trigger the `onPaletteSelection` callback whenever the `selectedBoxes` state changes, I used the `useEffect` hook. This effect is dependent on the `selectedBoxes` and `onPaletteSelection` values. Whenever `selectedBoxes` changes, the `onPaletteSelection` callback is invoked with the updated `selectedBoxes` array as an argument.

- Additionally, I implemented another `useEffect` hook to reset the `selectedBoxes` state whenever the `isLoading` prop changes. This ensures that when the loading state is toggled from `true` to `false`, the `selectedBoxes` state is reset to an array of `null` values.

- In terms of the visual representation, I designed the palette to have three rows and five columns using nested `div` elements. Each box is represented by a `div` with specific styling. The background color of the boxes is determined based on their `rowIndex`, where `0` corresponds to "green", `1` corresponds to "yellow", and `2` corresponds to "white".

- To provide visual feedback for the selected boxes, I adjusted the opacity of the boxes based on their match with the `selectedBoxes` state. If a box's index matches the corresponding `rowIndex` in `selectedBoxes`, its opacity is set to `1`, indicating selection. Otherwise, its opacity is set to `0.5`, indicating deselection.

#CardDisplay

- The component utilizes the `Card`, `CardContent`, and `Typography` components from the `@mui/material` library for styling.

- The `Typography` component is used to display the chance number, word to guess, and response received.

- The `BoxDisplay` component is used to render the guess word and response in separate boxes.

- Inline CSS is applied to the `Card` component to customize its appearance, removing the default border and box shadow styles.

#BoxDisplay

- Each box is represented by a `div` element with specific dimensions, border, margin, text alignment, line height, and a default background color of white.

- The `getBackgroundColor` function determines the background color for each box based on the `backgroundColor` prop. It checks if the prop is a valid string of length 5 and maps the characters to corresponding colors. If the prop is invalid or the index is out of bounds, the default background color of white is used.

- The `word` prop is split into individual letters using the `split` method. Each letter is then rendered as a box with its corresponding background color determined by `getBackgroundColor`. The `map` function is used to iterate over the letters and generate the box elements.

In summary, my implementation of the WordleBot component focused on managing game state, integrating an external API, providing a user-friendly interface, and maintaining code modularity. These choices aimed to enhance the overall functionality and user experience of the WordleBot game.