Subject Name: **Front End Engineering**

Subject Code: CS186

Cluster: iGamma Group: **G19**

Department: **DCSE**

Submitted By:
Himanshu Krishna
2110992061
G19

Submitted To:
Ms. Pritpal Kaur

CHITKARA
UNIVERSITY

Project Name: Simon Game

React Simon Game

Introduction

Welcome to the documentation of our React-based Simon Game App. In this documentation, we will explore the key components that make up our application and their respective functionalities.

Our Simon Game is designed to help users capture and organize their thoughts, ideas, and important notes in a simple and intuitive manner. It offers a user-friendly interface and a range of features for managing notes efficiently.

Technologies Used:

- HTML
- CSS
- JavaScript
- React

Key Components

1. App Component

- The central component of our application that help to play game.

- Allows users to check memory level by remembering colours.

- Coordinates the interactions between other components.

2. Heading Component

- Represents the application's header, displaying the app title with an icon for visual appeal.

3. Infolcon Component

- Displays a dynamic color that automatically updates with the current color with transition.

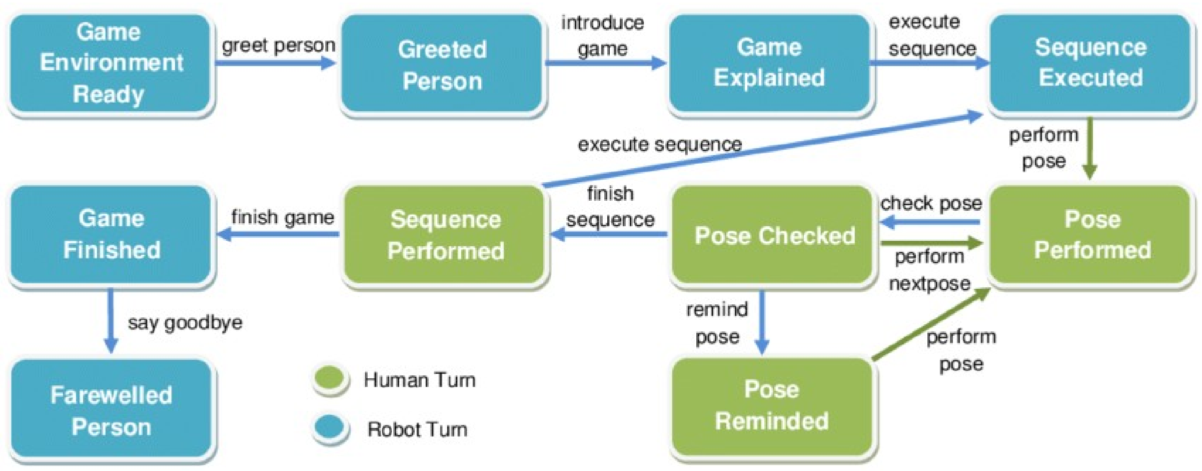
4. Index Component

- Represents an individual color within the application.

Usage

The Simon game is **the exciting electronic game of lights and sounds in which players must repeat random sequences of lights by pressing the colored pads in the correct order**. It's fast-paced play, with lights and sounds that can challenge you. Experience the fun as you repeat the patterns and advance to higher levels.

Flow Chart:



------------------------------Components Documentation--------------------------------

Index.js

It appears that you've provided the root file of your React application, which sets up the application and renders the main `App` component using React Concurrent Mode. This is a standard entry point for a React application. I'll provide a brief documentation for this file:

File: index.js

Description: This is the entry point for your React application, responsible for setting up the application and rendering the main `App` component.

Dependencies:

- `React`: The core library for building user interfaces in React.

- `ReactDOM`: The library for rendering React components in the DOM.

- `./index.css`: The stylesheet for styling the application.

- `App`: The main application component.

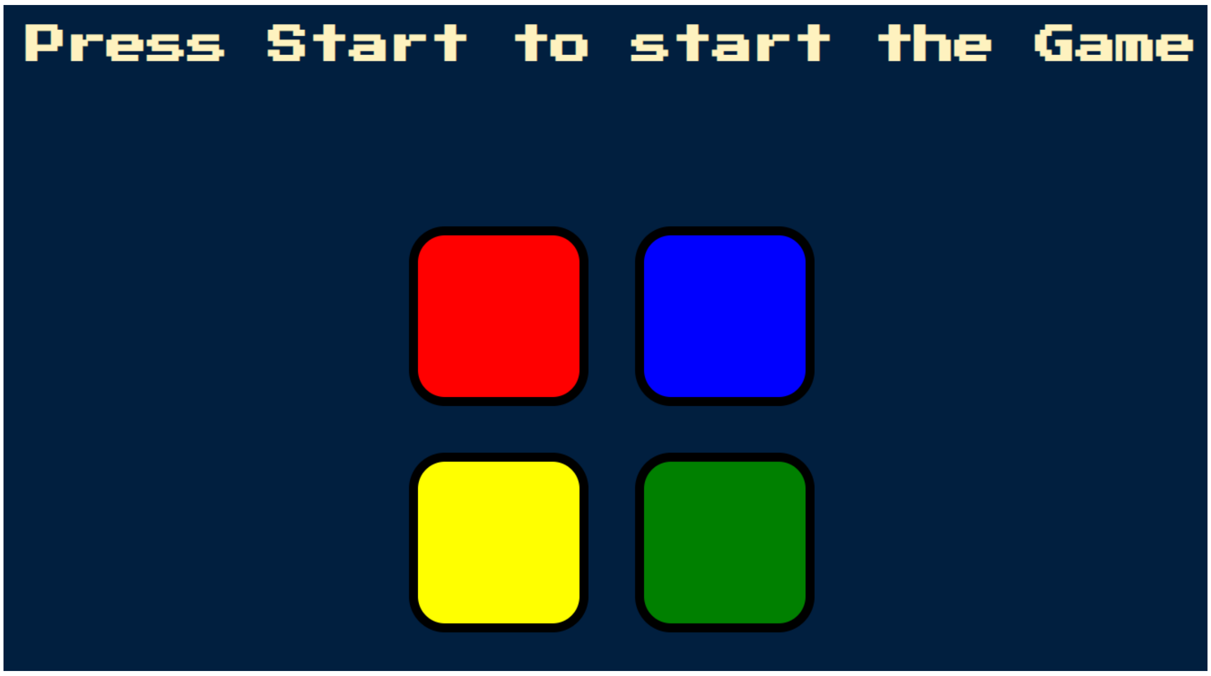
Rendering:

- `ReactDOM.createRoot`: Creates a concurrent mode root, which is used for rendering components in a concurrent and asynchronous manner.

- `root.render(...)`: Renders the `App` component inside the concurrent mode root.

Example Usage:

```javascript
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```



1. App.jsx

Here's a documented breakdown of your React component, `App`, which appears to be the main component for your Simon game application:

Component Name: App

Description: The `App` component is the main component for your Simon game. It handles the game logic, user interactions, and rendering various elements on the screen.

States:

- `gamePattern`: An array representing the computer's generated sequence of colors.

- `userClickedPattern`: An array representing the user's clicked colors.

- `level`: An integer representing the current level of the game.

- `heading`: A string representing the game's status or instructions.

- `randomChosenColour`: A string representing the currently selected random color.

- `wrongAnswer`: A boolean indicating if the user's answer is wrong.

- `isStarted`: A boolean indicating whether the game has started.

- `showInfo`: A boolean to control the visibility of the game info.

- `buttonColour`: An array containing the color options.

- `wrongSound`: An `Audio` object for the wrong answer sound effect.

References:

- `ref`: A reference to a DOM element, used for handling the game info modal.

Functions:

- `handleInfoClick`: Function to open the game info modal.

- `userClick`: Function to handle user color button clicks.

- `nextSequence`: Function to generate the next color sequence.

- `checkAnswer`: Function to check the user's answer against the game pattern.

Effects:

- The first `useEffect` watches for changes in `userClickedPattern` and checks the user's input for correctness.

- The second `useEffect` adds a new random color to the game pattern when `randomChosenColour` changes.

- The third `useEffect` does not have an associated function, but it seems to update when `wrongAnswer`, `level`, or `gamePattern` change.

- The fourth `useEffect` updates the heading when the game starts (`isStarted` changes).

Component Structure:

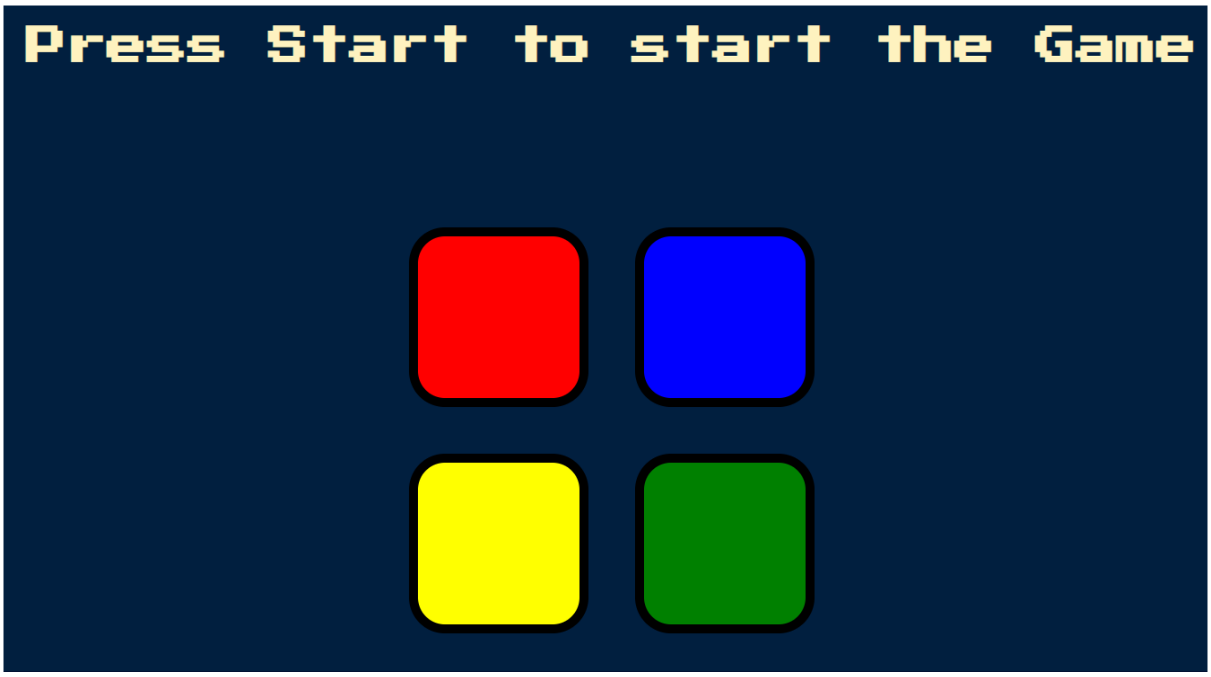
- It renders the `GameInfo` component to show game information.

- It renders the game elements, including the `Heading`, `BoxContainer`, and `StartButton`.

Example Usage:

```jsx
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

This documentation outlines the key aspects of your `App` component, including its states, functions, effects, and how it fits into your overall application.



2. Box.jsx

Here's a documented breakdown of your `Box` component, which appears to represent the colored boxes in your Simon game:

Component Name: Box

Description: The `Box` component represents a colored box in your Simon game. It handles user interactions, displays colored boxes, and provides feedback on user clicks and game progression.

Props:

- `color`: A string representing the color of the box.

- `sound`: An `Audio` object to play a sound when the box is clicked or highlighted as the next in the sequence.

- `next`: A string representing the color that is next in the game sequence.

- `userClick`: A callback function to handle user clicks on the box.

States:

- `isClicked`: A boolean indicating whether the box has been clicked.

- `isNext`: A string representing the color that is next in the game sequence.

Functions:

- `handleClick`: Function to handle user clicks on the box. It sets `isClicked` to true, plays a sound, and invokes the `userClick` callback. After a brief delay, `isClicked` is set back to false.

Effects:

- The `useEffect` listens for changes in `next`, `sound`, and `color` to determine if the box should be highlighted as the next in the sequence. It sets `isNext` to the color and plays a sound effect temporarily.

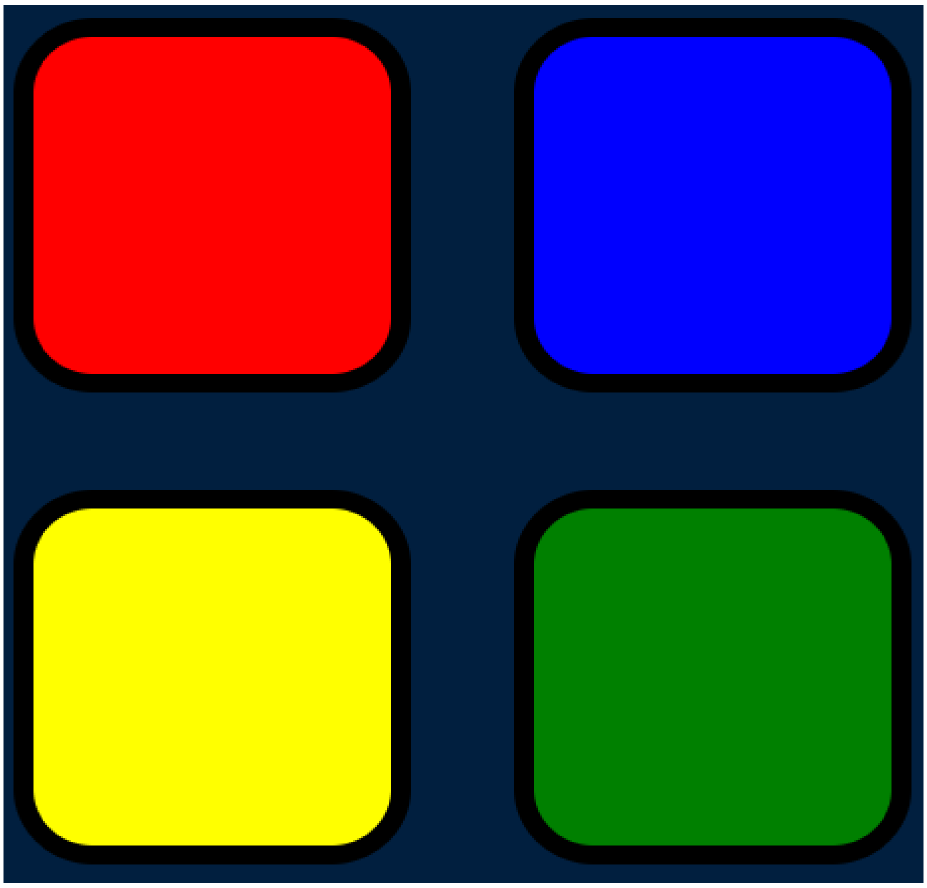
Component Structure:

- It renders a button element with a dynamic class that changes based on the state of `isClicked` and `isNext`. It also calls the `handleClick` function when clicked.

Example Usage:

```jsx

<Box

  color="red"

  sound={audioObject}

  next="red"

  userClick={handleUserClick}

/>
```

This documentation outlines the key aspects of your `Box` component, including its props, states, functions, and effects. It provides users with a clear understanding of the component's purpose and usage within your Simon game.



3. BoxContainer.js

Here's a documented breakdown of your `BoxContainer` component, which seems to represent a container of colored boxes in your Simon game:

Component Name: BoxContainer

Description: The `BoxContainer` component represents a container for the colored boxes in your Simon game. It organizes the layout and presentation of the colored boxes, as well as their associated sounds and user interaction.

Props:

- `randomChosenColour`: A string representing the randomly chosen color in the game sequence.

- `userClick`: A callback function to handle user clicks on the boxes.

Audio Imports:

- It imports audio files (`red.mp3`, `green.mp3`, `blue.mp3`, and `yellow.mp3`) and creates `Audio` objects for each color's sound.

Component Structure:

- It organizes colored boxes in a 2x2 grid, each represented by the `Box` component.

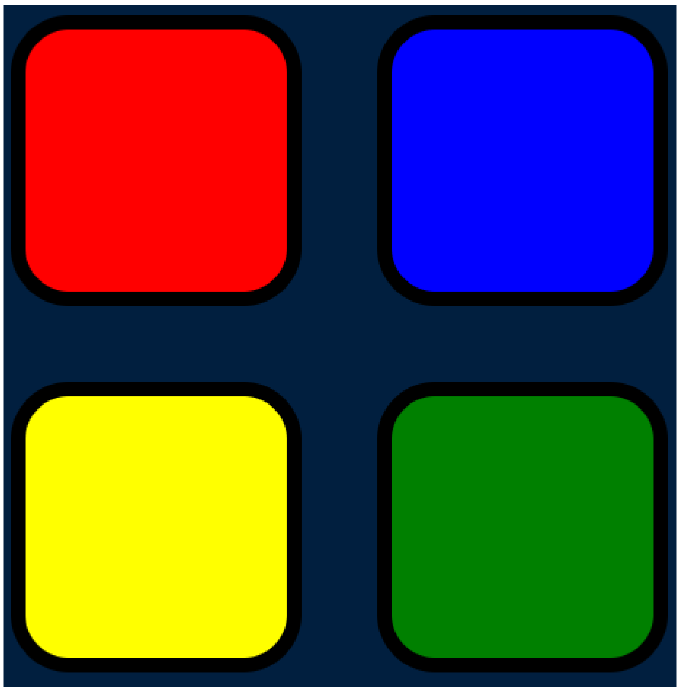
- Each `Box` component receives the respective color, the next color in the game sequence, the user click callback, and the corresponding sound.

Example Usage:

```jsx
```

```
<BoxContainer

  randomChosenColour="red"

  userClick={handleUserClick}

/>
```

This documentation outlines the key aspects of your `BoxContainer` component, including its props, the organization of colored boxes, and their associations with sounds and user interaction. It helps users understand the purpose and usage of this component within your Simon game. If you have more components or specific parts of your code to document, please let me know.



4. Heading.js

Here's a documented breakdown of your `Heading` component, which appears to be responsible for displaying the game's heading or level information:

Component Name: Heading

Description: The `Heading` component is responsible for displaying the heading or level information for your Simon game.

Props:

- `level`: An integer representing the current level of the game.

- `heading`: A string representing the game's status or instructions.

Component Structure:

- It renders an `<h1>` element with dynamic classes that control its appearance.

- The text content of the `<h1>` element is determined based on the presence of a `heading` and the `level` prop.

Example Usage:

```jsx

<Heading level={5} heading="Game in Progress" />

<Heading level={6} heading="" />

```

This documentation provides an overview of your `Heading` component, its props, and how it's used to display game information or level details. It helps users understand how to use this component within your Simon game.

5. StartButton.js

Here's a documented breakdown of your `StartButton` component, which appears to be responsible for rendering the "Start" button in your Simon game:

Component Name: StartButton

Description: The `StartButton` component renders the "Start" button, which initiates the game when clicked.

Props:

- `nextSequence`: A function to initiate the next sequence in the game.

- `isStarted`: A boolean indicating whether the game has already started.

- `setIsStarted`: A function to update the `isStarted` state when the button is clicked.

Component Structure:

- It renders a `<button>` element with dynamic classes that control its appearance.

- The button is conditionally rendered based on the `isStarted` prop. It is displayed only if the game has not started (`!isStarted`).

- When the button is clicked, it sets `isStarted` to `true` and initiates the `nextSequence` function after a brief delay.

Example Usage:

```jsx

<StartButton nextSequence={startGame} isStarted={isGameStarted} setIsStarted={setIsGameStarted} />

This documentation provides an overview of your `StartButton` component, its props, and how it's used to start the game. It helps users understand how to use this component within your Simon game.

```