DragAndDropComponent.js

```
import React, { useState } from "react";
import { View, Text, StyleSheet, Dimensions, Image } from "react-
native";
import Draggable from "react-native-draggable";

const tarotDeck = [
  { id: 1, content: "The Fool", image:
require("./img/fool.png") },
  { id: 2, content: "The Magician", image:
require("./img/magician.png") },
  {
    id: 3,
    content: "The High Priestess",
    image: require("./img/priestess.png"),
  },
  // 하드코딩으로 더미 데이터 삽입, 타로 카드들의 정보를 담은 tarotDeck 배열 생
성
];

export default function DragAndDropComponent() {
  // droppedCard 상태는 드래그 앤 드롭 기능을 통해 어떤 타로 카드가 초록 박스에
드롭되었는지를 나타냄
  const [droppedCard, setDroppedCard] = useState(null);
  // 반응형 디자인을 위해서 화면의 크기를 가져옴
  const windowWidth = Dimensions.get("window").width; // 현재 창의 너
비
  const windowHeight = Dimensions.get("window").height; // 현재 창의
높이

  // 초록 박스 크기 조절 : 카드를 드래그 앤 드롭할 영역
  const dropZoneWidth = windowWidth * 0.2; //초록 박스의 너비
  const dropZoneHeight = windowHeight * 0.25; //초록 박스의 높이

  //드래그 앤 드롭 동작 완료될 때 호출되는 함수
  /*
 여러 매개 변수를 받아 해당 동작이 초록 박스 영역 안에서 이루어진 경우에만
 setDroppedCard 함수를 사용하여 droppedCard 상태를 업데이트함
  */
  const onDragRelease = (event, gesture, component, card) => {
    // dropZoneX : 초록 박스 영역에서의 가로좌표
    // (windowWidth - dropZoneWidth) / 2 는 초록 박스를 수평으로 중앙에 위
```

```
치시키는 역할을 함
    // dropZoneX는 초록 박스의 왼쪽 가장자리의 X좌표가 됨
    const dropZoneY = windowHeight * 0.5;

    // dropZoneY : 초록 박스 영역에서의 세로 좌표
    // windowHeight * 0.5는 현재 창의 높이의 절반 지점에 해당하는 Y 좌표로, 초
록 박스를 수직으로 중앙에 위치시키는 역할을 함
    // dropZoneY는 초록 박스의 상단 가장자리의 Y 좌표가 됨
    const dropZoneX = (windowWidth - dropZoneWidth) / 2;

    // 조건문: 드래그가 완료된 지점이 초록 박스 영역 안에 있는지 확인
    /*
  조건문 내용
  gesture.moveX > dropZoneX : 드래그가 완료된 지점의 가로 좌표가 초록 박스
의 왼쪽 가장자리를 넘어서는지 확인
  gesture.moveX < dropZoneX + dropZoneWidth : 드래그가 완료된 지점의 가
로 좌표가 초록 박스의 오른쪽 가장자리를 넘어서는지 확인
  gesture.moveY > dropZoneY : 드래그가 완료된 지점의 세로 좌표가 초록 박스
의 상단 가장자리를 넘어서는지 확인
  gesture.moveY < dropZoneY + dropZoneHeight : 드래그가 완료된 지점의
세로 좌표가 초록 박스의 하단 가장자리를 넘어서는지 확인
  모든 조건이 만족된다면, 즉 드래그한 카드가 초록 박스 안으로 드롭되었다면
setDroppedCard(card)를 호출하여 drooppedCard 상태를 업데이트함
    */
    if (
      gesture.moveX > dropZoneX &&
      gesture.moveX < dropZoneX + dropZoneWidth &&
      gesture.moveY > dropZoneY &&
      gesture.moveY < dropZoneY + dropZoneHeight
    ) {
      setDroppedCard(card);
    }
  };

  return (
    <View style={styles.container}>
      <View
        style={[
          styles.dropZone,
          { width: dropZoneWidth, height: dropZoneHeight },
        ]}
```

```
      >
        <Text style={styles.dropText}>Drag here</Text>
        {droppedCard && (
          <View style={styles.droppedContent}>
            <Image source={droppedCard.image}
style={styles.cardImage} />
            <Text style={styles.cardText}>{droppedCard.content}
</Text>
          </View>
        )}
      </View>
      {tarotDeck.map((card) => (
        <Draggable
          key={card.id}
          x={(windowWidth - dropZoneWidth) / 2}
          y={windowHeight * 0.3 + (card.id - 1) * 60}
          renderSize={80}
          isCircle={false}
          onDragRelease={(event, gesture, component) =>
            onDragRelease(event, gesture, component, card)
          }
          draggableArea={{
            left: 0,
            top: 0,
            right: windowWidth - 80,
            bottom: windowHeight - 80,
          }}
        >
          <View style={styles.draggableCard}>
            <Image
              source={require("./img/card-back.png")}
              style={styles.cardImage}
            />
          </View>
        </Draggable>
      ))}
    </View>
  );
}

const styles = StyleSheet.create({
```

```
  container: {
    flex: 1,
    justifyContent: "center",
    alignItems: "center",
  },
  dropZone: {
    backgroundColor: "#4CAF50",
    marginTop: 20,
    justifyContent: "center",
    alignItems: "center",
  },
  dropText: {
    color: "white",
  },
  droppedContent: {
    marginTop: 20,
    alignItems: "center",
  },
  cardImage: {
    width: 80,
    height: 120,
    resizeMode: "contain",
  },
  cardText: {
    color: "white",
    marginTop: 10,
    fontSize: 16,
  },
  draggableCard: {
    borderRadius: 10,
    padding: 10,
  },
});
```

위 코드 간단 정리

- 라이브러리 및 상태 선언:
  - React, useState를 import하여 사용합니다.
  - react-native 라이브러리에서는 View, Text, StyleSheet, Dimensions, Image 등의 컴포넌트 및 스타일링 요소를 import합니다.

    - react-native-draggable 라이브러리에서 Draggable 컴포넌트를 import 합니다.
    - tarotDeck라는 배열을 선언하여 타로 카드의 정보를 담고 있습니다.
-
- 상태 관리:
    - useState를 사용하여 droppedCard라는 상태를 선언합니다. 이 상태는 드래그 앤 드롭으로 초록 박스에 드롭된 타로 카드의 정보를 담고 있습니다.
- 화면 크기 가져오기:
    - Dimensions.get("window").width와 Dimensions.get("window").height를 사용하여 현재 창의 너비와 높이를 가져옵니다.
- 드래그 앤 드롭 영역 설정:
    - 초록 박스의 크기를 조절하고, dropZoneWidth 및 dropZoneHeight 변수에 저장합니다.
- 드래그 앤 드롭 완료 처리 함수:
    - onDragRelease 함수는 드래그 앤 드롭이 완료될 때 호출됩니다.
    - 드래그된 카드가 초록 박스 안에 드롭되었는지를 조건문을 통해 확인하고, 드롭된 경우 setDroppedCard를 호출하여 droppedCard 상태를 업데이트합니다.
- 컴포넌트 렌더링:
    - View 컴포넌트로 전체 컴포넌트를 감싸고, 초록 박스 및 드래그 가능한 타로 카드를 렌더링합니다.
    - 초록 박스 영역에는 "Drag here"라는 텍스트와 드롭된 카드의 정보를 표시합니다.
- 드래그 가능한 타로 카드 렌더링:
    - tarotDeck 배열을 매핑하여 각 타로 카드에 대해 Draggable 컴포넌트를 생성합니다.
    - 각 카드는 초기 위치, 크기, 드래그가 완료될 때 호출될 함수 등의 속성을 설정합니다.
- 스타일링:
    - StyleSheet.create를 사용하여 스타일 객체를 만들어 각 컴포넌트에 적용합니다.
    - 초록 박스, 드롭된 카드의 디자인 및 스타일 속성을 정의합니다.

자료조사

https://froggydisk.github.io/tenth-post/

https://velog.io/@horang-e/React-Native-%EB%93%9C%EB%9E%98%EA%B7%B8%EC%95%A4-%EB%93%9C%EB%A1%AD-FlatList-%EB%A7%8C%EB%93%A4%EA%B8%B0react-native-draggable-flatlist

https://www.npmjs.com/package/react-native-draggable

https://velog.io/@horang-e/React-Native-%EB%93%9C%EB%9E%98%EA%

B7%B8%EC%95%A4- %EB%93%9C%EB%A1%AD- FlatList- %EB%A7%8C%EB%93%A4%EA%B8%B0react- native- draggable- flatlist

https://www.npmjs.com/package/react- native- draggable- flatlist

https://blog.reactnativecoach.com/creating- draggable- component- with- react- native- 132d30c27cb0

https://www.geeksforgeeks.org/how- to- implement- drag- and- drop- functionality- in- react- native/