기초입문편

부록

01

내일은 플러터

기초 개발 예제

01 스플래시 화면

✓핵심 키워드

스플래시, Duration, 로딩바, CircularProgressIndicator

여기서는 무얼 배울까

앱을 처음 실행할 때 나오는 스플래시 화면을 만들어 보고자 한다. 스플래시 화면을 사용하는 이유에 대해서 알아보고, 어플리케이션 로고와 함께 앱 이름과 버전을 애니메이션을 이용하여 출력하는 법을 배운다. 자신만의 스플래시 화면을 디자인해 보도록 하자.

스플래시 화면(Splash Screen)

게임이나 프로그램이 시작되는 동안 나타나는 화면을 스플래시 화면이라고 한다. 스플래시 화면을 사용하는 이유는 다음과 같다.

- 브랜딩 및 앱 아이덴티티 강화: 스플래시 화면은 앱을 실행할 때 첫 화면으로 나타나는데, 이를 통해 앱의 로고, 이름 및 디자인 요소를 강조하여 브랜딩과 앱의 아이덴티티를 강화할 수 있다.
- 초기화 시간 감추기: 앱이 초기화되는 동안 스플래시 화면을 보여줌으로써 사용자에게 초기화 시간이 필요한 것처럼 느껴지지 않게 한다. 이는 사용자에게 앱이 빠르게 로드되는 것처럼 느껴지게 해서 사용자 경험을 개선할 수 있다.
- 로고 및 앱 정보 표시: 스플래시 화면을 통해 앱의 로고, 이름, 버전 정보 등을 표시하여 사용자에게 앱의 신뢰성을 전달하고 업데이트 여부를 알리는 데 도움을 줄 수 있다.

로딩 화면의 스플래시 애니메이션을 구현하기 위해 다음과 같은 단계를 따를 수 있다.

① 프로젝트에 애니메이션을 위한 이미지 또는 로고를 추가한다.

② SplashScreen 클래스를 생성하고 StatefulWidget으로 만든다.

③ initState 메서드에서 로고 애니메이션을 초기화하고 실행한다.

④ 로고 애니메이션을 위한 컨트롤러와 애니메이션을 설정한다.

⑤ build 메서드에서 로고 애니메이션을 포함한 위젯을 반환한다.

⑥ 필요한 시간만큼 로고가 나타난 후, 다음 화면으로 전환한다.

다음은 위 단계를 반영한 스플래시 애니메이션 코드다.

```
import 'package:flutter/material.dart';

class SplashScreen extends StatefulWidget {
  @override
  _SplashScreenState createState() => _SplashScreenState();
}

class _SplashScreenState extends State<SplashScreen>
    with SingleTickerProviderStateMixin {
  AnimationController? _animationController;
  Animation<double>? _animation;

  @override
  void initState() {
    super.initState();
    _animationController = AnimationController(
      vsync: this,
      duration: Duration(milliseconds: 1000),
    );
    _animation = CurvedAnimation(
      parent: _animationController!,
      curve: Curves.easeInOut,
    );
    _animationController!.forward();
    _animationController!.addStatusListener((status) {
      if (status == AnimationStatus.completed) {
        // 로딩 애니메이션이 완료되면 다음 화면으로 이동
        Navigator.pushReplacementNamed(context, '/home');
      }
```

```
    });
  }

  @override
  void dispose() {
    _animationController?.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.white,
      body: Center(
        child: FadeTransition(
          opacity: _animation!,
          child: FlutterLogo(
            size: 200,
          ),
        ),
      ),
    );
  }
}
```

위 코드에서는 SplashScreen 클래스를 만들고 initState에서 애니메이션 컨트롤러와 애니메이션을 초기화한다. build 메서드에서는 FadeTransition 위젯을 사용하여 애니메이션을 적용한 로고를 화면에 표시한다. 로딩 애니메이션이 완료되면 addStatusListener를 통해 다음 화면으로 이동하도록 설정한다.

위 코드는 SplashScreen을 앱의 첫 화면으로 사용하는 경우를 가정하여 작성되었다. 필요에 따라 Navigator.pushReplacementNamed를 사용하여 다음 화면으로 전환하는 코드를 수정해야 할 수도 있다.

애니메이션

로고나 다른 그래픽 요소에 애니메이션 효과를 적용하여 동적이고 흥미로운 스플래시 화면을 만들 수 있다. 아래는 플러터에서 애니메이션을 사용하여 로고를 서서히 나타내는 효과를 구현하는 코드다.

```
import 'package:flutter/material.dart';

class SplashScreen extends StatefulWidget {
  @override
  _SplashScreenState createState() => _SplashScreenState();
}

class _SplashScreenState extends State<SplashScreen>
    with SingleTickerProviderStateMixin {
  AnimationController? _animationController;
  Animation<double>? _animation;

  @override
  void initState() {
    super.initState();
    _animationController = AnimationController(
      vsync: this,
      duration: Duration(seconds: 2),
    );

    _animation = CurvedAnimation(
      parent: _animationController!,
      curve: Curves.easeInOut,
    );

    _animationController!.forward();
  }

  @override
  void dispose() {
    _animationController!.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
```

```
      body: Center(
        child: FadeTransition(
          opacity: _animation!,
          child: FlutterLogo(size: 200),
        ),
      ),
    );
  }
}
```

위의 코드에서는 AnimationController와 CurvedAnimation을 사용하여 애니메이션을 제어한다. AnimationController는 애니메이션을 제어하는 컨트롤러이며, CurvedAnimation은 애니메이션을 부드럽게 변화시키는 곡선을 정의한다. 애니메이션이 시작되면 _animationController.forward() 메서드를 호출한다. 애니메이션은 FadeTransition 위젯을 사용하여 로고가 서서히 나타나도록 구현되었다. opacity 속성에 _animation을 연결하여 애니메이션을 적용하고, 나타나는 로고는 FlutterLogo 위젯을 사용하였다. 이 코드를 사용하면 스플래시 화면에서 로고가 서서히 나타나는 효과를 구현할 수 있다. 애니메이션 컨트롤러의 duration을 조정하거나 Curves 클래스에서 제공하는 다양한 곡선을 사용하여 원하는 애니메이션 효과를 조정할 수 있다.

앱 이름 및 버전

스플래시 화면에 앱의 이름과 버전 정보를 표시할 수 있다. 이는 사용자에게 앱의 신뢰성과 업데이트 여부를 알리는 데 도움이 된다.

```
import 'package:flutter/material.dart';
import 'package:package_info/package_info.dart';

class SplashScreen extends StatefulWidget {
  @override
  _SplashScreenState createState() => _SplashScreenState();
}

class _SplashScreenState extends State<SplashScreen> {
  String _appName = '';
  String _version = '';

  @override
  void initState() {
```

```
    super.initState();
    _getAppInfo();
  }

  Future<void> _getAppInfo() async {
    PackageInfo packageInfo = await PackageInfo.fromPlatform();
    setState(() {
      _appName = packageInfo.appName;
      _version = packageInfo.version;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Text(
              _appName,
              style: TextStyle(fontSize: 24, fontWeight: FontWeight.bold),
            ),
            SizedBox(height: 16),
            Text(
              'Version $_version',
              style: TextStyle(fontSize: 16),
            ),
          ],
        ),
      ),
    );
  }
}
```

위의 코드에서는 package_info 패키지를 사용하여 앱의 이름과 버전 정보를 가져온다. initState() 메서드에서 _getAppInfo() 함수를 호출하여 앱 정보를 가져오고, setState()를 사용하여 상태를 업데이트한다. 그리고 화면에 앱 이름과 버전 정보를 표시한다. Column 위젯을 사용하여 앱 이름과 버전 정보를 수직으로 정렬하고 중앙에 배치한다. 텍스트 스타일을 사용하여 앱 이름과 버전 정보의 모양과 크기를 조정할 수 있다. 이 코드를 사용하면 스플래시 화면에

앱의 이름과 버전 정보를 표시할 수 있다. 사용자는 앱이 어떤 버전인지 확인하고 업데이트 여부를 파악할 수 있다.

로딩바

로딩 바를 별도로 구현하기 위해 다음과 같은 방법을 사용할 수 있다.

① 필요한 패키지를 가져 온다.

```
import 'package:flutter/material.dart';
```

② 로딩 바를 표시할 위젯 내부에서 CircularProgressIndicator 위젯을 사용한다.

```
CircularProgressIndicator(
  valueColor: AlwaysStoppedAnimation<Color>(Colors.blue),
)
```

③ valueColor 속성을 사용하여 로딩 바의 색상을 설정한다. AlwaysStoppedAnimation을 사용하여 로딩 바의 색상이 항상 일정하게 유지되도록 설정한다. 코드에서는 파란색으로 설정되어 있다. 원하는 색상으로 변경할 수 있다.

④ 필요에 따라 다른 속성을 사용하여 로딩 바의 크기, 두께 등을 조정할 수 있다. CircularProgressIndicator 위젯에는 다양한 속성이 있으며, 필요에 따라 사용자 정의할 수 있다.

아래는 로딩 바를 표시하는 코드다.

```
import 'package:flutter/material.dart';

class MyLoadingScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: CircularProgressIndicator(
          valueColor: AlwaysStoppedAnimation<Color>(Colors.blue),
        ),
      ),
    );
```

```
  }
}
```

위의 코드에서는 MyLoadingScreen이라는 위젯을 만들어 로딩 바를 화면에 표시하고 있다. Center 위젯을 사용하여 로딩 바를 화면 중앙에 정렬하고 있다. CircularProgressIndicator의 valueColor 속성을 사용하여 파란색으로 로딩 바를 설정하고 있다. 이렇게 구현된 로딩 바는 로딩 상태를 시각적으로 나타내는 데 사용될 수 있다.

02 랜덤게임 앱

✓핵심 키워드

팝업, showDialog, CustomPaint, AnimationController, 로고

여기서는 무얼 배울까

챕터 4와 5에서 배운 것들을 활용하여 게임 앱을 만들어 보자. 여러 개의 게임을 해볼 수 있도록 여러 페이지로 구성된 앱을 구현한다. 앱 내부에서 사용하는 팝업에 대해서 배워보고, 팝업에서 데이터를 전달하는 법을 배운다. 앱을 만들고 난 뒤 앱 이름과 로고를 수정하여 나만의 어플리케이션을 만들어 보자.

아이디어 및 기획

어플리케이션의 주요 아이디어를 정리하고 기획 단계를 시작한다. 여러 개의 랜덤게임을 할 수 있는 서비스를 만들어 보자. 다음은 만들고자 하는 게임의 종류다.

사다리타기 게임

'사다리타기' 또는 고스트 레그(Ghost Leg,유령다리, 일본 Amidakuji)는 제비뽑기의 일종으로 세로줄 사이에 가로줄을 겹치지 않게 무작위로 그은 다음 위에서 아래로 선을 따라 내려가면서 최종적으로 도달하여 결과를 정하는 게임이다. 가장 처음 사다리 게임에 참여할 인원 수를 설정하고, 제일 위와 제일 아래에 사용자 입력 기능을 추가하며, 제일 위에 입력된 값들과 제일 아래 입력된 값들을 매칭시키는 것으로 구현해 보자.

돌림판 게임

돌아가는 원판에 원하는 것들 적고 판을 돌렸을 때, 화살표가 가리키고 있는 위치가 당첨이 되는 추첨 게임이다. 사용자가 원하는 값들을 원판에 입력할 수 있도록 하고, 원판을 돌릴 수 있게 해 보자. 최종적으로 화살표가 가리키는 위치에 있는 값이 결과가 될 수 있도록 구현해 보자.

몬티홀 딜레마 게임

몬티홀 문제란 캐나다-미국 TV 프로그램 사회자가 진행하던 미국 오락 프로그램에서 유래한 확률 문제다. 닫혀 있는 문 3개 뒤에 자동차 한 대와 염소 두 마리 중 하나씩 있을 때, 자동차가 있는 문을 고르는 것이다. 몬티홀 딜레마는 참가자가 첫 번째 선택을 했을 때, 염소가 있는 문을 하나 열어 주고, 선택을 바꿀 수 있는 기회를 제공한다. 이때 선택을 바꾸는 것이 유리할지가 몬티홀 딜레마 문제라고 할 수 있다. 문 뒤에 상품을 하나 숨겨 놓고 자동차를 찾아보는 몬티홀 딜레마 문제를 만들어 보자.

로또 게임

국내 로또 확률을 체감할 수 있게 돕는 게임이다. 로또 1등 당첨의 확률은 1 / 8,145,060로 대략 2^{23}이라고 할 수 있다. 이지선다를 연속으로 23번 맞추게 되면 1등의 확률과 비슷하다고 할 수 있다. 2등은 20번, 3등은 15번, 4등은 9번과 10번 사이, 5등은 5번과 6번 사이의 확률이라고 할 수 있다. 2개의 버튼을 만들어서 랜덤하게 통과할 수 있도록 해 보자.

게임 선택화면

우선 메인 페이지에서 게임 선택을 할 수 있는 게임 선택화면을 작성해 보자. 선택지는 5가지로 사다리타기, 몬티홀 딜레마 게임, 로또 게임, 종료 버튼으로 이루어져 있다.

```
import 'package:flutter/material.dart';
import 'dart:math';

void main() {
  runApp(GameSelectionApp());
}

class GameSelectionApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Game Selection',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: GameSelectionScreen(),
    );
```

```
  }
}

class GameSelectionScreen extends StatefulWidget {
  @override
  _GameSelectionScreenState createState() => _GameSelectionScreenState();
}

class _GameSelectionScreenState extends State<GameSelectionScreen> {
  int _playerCount = 2;
  int _questionCount = 3;
  int _lottoRank = 1;
  List<String> _participants = [];

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Game Selection'),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            ElevatedButton(
              onPressed: () {
                showPlayerCountDialog();
              },
              child: Text('1. 사다리타기'),
            ),
            ElevatedButton(
              onPressed: () {
                showRouletteCountDialog();
              },
              child: Text('2. 돌림판 게임'),
            ),
            ElevatedButton(
              onPressed: () {
                showQuestionCountDialog();
              },
              child: Text('3. 몬티홀 딜레마 문제'),
            ),
            ElevatedButton(
```

```
            onPressed: () {
              showLottoRankDialog();
            },
            child: Text('4. 로또 랜덤 게임'),
          ),
          ElevatedButton(
            onPressed: () {
              showExitConfirmationDialog();
            },
            child: Text('5. 종료하기'),
          ),
        ],
      ),
    ),
  );
}
```

위의 코드는 각 게임 선택지를 버튼으로 나타내는 게임 선택화면을 구성한다. 각 버튼의 onPressed 콜백에서는 해당 게임을 실행하도록 코드를 추가하면 된다.

팝업(Pop-up)

게임 선택화면에서 게임 선택 시 팝업 창을 통해 인원 수나 게임 종류를 선택할 수 있도록 해 보자. 기능을 추가하려면 showDialog 메서드를 사용하여 팝업 창을 띄우고, 선택에 따라 게임 동작을 조정하면 된다. 각 게임에 대한 세부적인 선택 기능을 추가하는 부분은 각 게임에 따라 다르기 때문에 별도의 구현이 필요하다. 각 선택 기능은 다음과 같다.

사다리 타기 게임: 인원 수 선택

```
void showPlayerCountDialog() async {
  int? newPlayerCount = await showDialog(
    context: context,
    builder: (BuildContext context) {
      int playerCount = _playerCount;

      return StatefulBuilder(
        builder: (BuildContext context, StateSetter setState) {
```

```
            return AlertDialog(
              title: Text('인원 수 선택'),
              content: Column(
                mainAxisSize: MainAxisSize.min,
                children: [
                  Text('인원 수를 선택하세요'),
                  Row(
                    mainAxisAlignment: MainAxisAlignment.center,
                    children: [
                      IconButton(
                        onPressed: () {
                          setState(() {
                            playerCount = max(2, playerCount - 1);
                          });
                        },
                        icon: Icon(Icons.remove),
                      ),
                      Text('$playerCount'),
                      IconButton(
                        onPressed: () {
                          setState(() {
                            playerCount = min(10, playerCount + 1);
                          });
                        },
                        icon: Icon(Icons.add),
                      ),
                    ],
                  ),
                ],
              ),
              actions: [
                TextButton(
                  onPressed: () {
                    Navigator.of(context).pop(playerCount);
                    Navigator.push(
                      context,
                      MaterialPageRoute(
                        builder: (context) =>
                            LadderGamePage(playerCount: playerCount),
                      ),
                    );
                  },
                  child: Text('시작'),
```

```
            ),
            TextButton(
              onPressed: () {
                Navigator.of(context).pop();
              },
              child: Text('취소'),
            ),
          ],
        );
      },
    );
  },
);

if (newPlayerCount != null) {
  setState(() {
    _playerCount = newPlayerCount; // 값이 반환되면 화면 상태 업데이트
  });
}
}
```

돌림판 게임: 돌림판 칸 수 선택 및 입력

```
void showRouletteCountDialog() async {
  List<String> participants = [];

  List<String>? result = await showDialog(
    context: context,
    builder: (BuildContext context) {
      return StatefulBuilder(
        builder: (BuildContext context, StateSetter setState) {
          return AlertDialog(
            title: Text('인원 수 선택'),
            content: Column(
              mainAxisSize: MainAxisSize.min,
              children: [
                Text('인원 수를 선택하세요'),
                ListView.builder(
                  shrinkWrap: true,
                  itemCount: participants.length,
```

```
                itemBuilder: (BuildContext context, int index) {
                  return TextFormField(
                    onChanged: (value) {
                      participants[index] = value;
                    },
                    decoration: InputDecoration(
                      hintText: '참가자 ${index + 1}',
                    ),
                  );
                },
              ),
              Row(
                mainAxisAlignment: MainAxisAlignment.center,
                children: [
                  IconButton(
                    onPressed: () {
                      setState(() {
                        if (participants.length > 2) {
                          participants.removeLast();
                        }
                      });
                    },
                    icon: Icon(Icons.remove),
                  ),
                  Text('${participants.length}'),
                  IconButton(
                    onPressed: () {
                      setState(() {
                        participants.add('');
                      });
                    },
                    icon: Icon(Icons.add),
                  ),
                ],
              ),
            ],
          ),
          actions: [
            TextButton(
              onPressed: () {
                Navigator.of(context).pop(participants);
                Navigator.push(
                  context,
```

```
                  MaterialPageRoute(
                    builder: (context) =>
                        RouletteGamePage(participants: participants),
                  ),
                );
              },
              child: Text('시작'),
            ),
            TextButton(
              onPressed: () {
                Navigator.of(context).pop();
              },
              child: Text('취소'),
            ),
          ],
        );
      },
    );
  },
 );
 if (result != null) {
   setState(() {
     // 반환된 값이 있으면 화면 상태 업데이트
     _participants = result;
   });
 }
}
```

몬티홀 딜레마 게임: 문의 수 선택

```
void showQuestionCountDialog() async {
  int? newQuestionCount = await showDialog(
    context: context,
    builder: (BuildContext context) {
      int questionCount = _questionCount;

      return StatefulBuilder(
        builder: (BuildContext context, StateSetter setState) {
          return AlertDialog(
            title: Text('문의 개수 선택'),
```

```
            content: Column(
              mainAxisSize: MainAxisSize.min,
              children: [
                Text('문의 개수를 선택하세요'),
                Row(
                  mainAxisAlignment: MainAxisAlignment.center,
                  children: [
                    IconButton(
                      onPressed: () {
                        setState(() {
                          questionCount = max(3, questionCount - 1);
                        });
                      },
                      icon: Icon(Icons.remove),
                    ),
                    Text('$questionCount'),
                    IconButton(
                      onPressed: () {
                        setState(() {
                          questionCount = min(10, questionCount + 1);
                        });
                      },
                      icon: Icon(Icons.add),
                    ),
                  ],
                ),
              ],
            ),
            actions: [
              TextButton(
                onPressed: () {
                  Navigator.of(context).pop(questionCount);
                  navigateToMontyHallGamePage(questionCount);
                },
                child: Text('시작'),
              ),
              TextButton(
                onPressed: () {
                  Navigator.of(context).pop();
                },
                child: Text('취소'),
              ),
            ],
```

```
          );
        },
      );
    },
  );

  if (newQuestionCount != null) {
    setState(() {
      _questionCount = newQuestionCount;
    });
  }
}
```

로또 게임: 등수 선택

```
void navigateToMontyHallGamePage(int questionCount) {
  Navigator.push(
    context,
    MaterialPageRoute(
      builder: (context) => MontyHallGameScreen(questionCount: questionCount),
    ),
  );
}

void showLottoRankDialog() async {
  int? newLottoRank = await showDialog(
    context: context,
    builder: (BuildContext context) {
      int lottoRank = _lottoRank;

      return StatefulBuilder(
        builder: (BuildContext context, StateSetter setState) {
          return AlertDialog(
            title: Text('로또 등급 선택'),
            content: Column(
              mainAxisSize: MainAxisSize.min,
              children: [
                Text('로또 등급을 선택하세요'),
                DropdownButton<int>(
                  value: lottoRank,
```

```
                onChanged: (value) {
                  setState(() {
                    lottoRank = value!;
                  });
                },
                items: List.generate(
                  5,
                  (index) => DropdownMenuItem(
                    value: index + 1,
                    child: Text('${index + 1}등'),
                  ),
                ),
              ),
            ],
          ),
          actions: [
            TextButton(
              onPressed: () {
                Navigator.of(context).pop(lottoRank);
                Navigator.push(
                  context,
                  MaterialPageRoute(
                    builder: (context) =>
                        LottoGameScreen(lottoRank: lottoRank),
                  ),
                );
              },
              child: Text('시작'),
            ),
            TextButton(
              onPressed: () {
                Navigator.of(context).pop();
              },
              child: Text('닫기'),
            ),
          ],
        );
      },
    );
  },
);

if (newLottoRank != null) {
```

```
      setState(() {
        _lottoRank = newLottoRank;
      });
    }
  }
```

종료 버튼: 선택 확인

```
  void showExitConfirmationDialog() {
    showDialog(
      context: context,
      builder: (BuildContext context) {
        return AlertDialog(
          title: Text('종료 확인'),
          content: Text('정말로 종료하시겠습니까?'),
          actions: [
            TextButton(
              onPressed: () {
                Navigator.of(context).pop();
              },
              child: Text('아니오'),
            ),
            TextButton(
              onPressed: () {
                Navigator.of(context).pop();
                Navigator.of(context).pop();
              },
              child: Text('예'),
            ),
          ],
        );
      },
    );
  }
}
```

위의 코드는 팝업 창을 통해 인원 수, 문의 개수, 로또 등급을 선택할 수 있도록 구현한다. 각 버튼을 누르면 해당 팝업 창이 나타나며, 버튼의 콜백 함수에서는 각 상황에 맞는 동작을 추가하면 된다. 종료 버튼의 경우, 정말 종료할 것인지를 묻는 확인 팝업 창이 나타나며, 확인 버튼을 누르면 앱이 종료된다. 각 코드에서 선택한 값을 사용하기 위해 GameSelectionScreenState 클래스에서 지역 변수를 선언해야 한다. 지역 변수는 다음과 같다.

```
int playerCount = 2;
int questionCount = 3;
int lottoRank = 1;
List<String> _participants = [];
```

사다리 타기 게임

```
class LadderGameInputPage extends StatefulWidget {
  @override
  _LadderGameInputPageState createState() => _LadderGameInputPageState();
}

class _LadderGameInputPageState extends State<LadderGameInputPage> {
  TextEditingController playerCountController = TextEditingController();

  @override
  void dispose() {
    playerCountController.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('사다리 게임'),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Text('플레이어 수 입력:'),
            SizedBox(height: 16),
```

```
            TextField(
              controller: playerCountController,
              keyboardType: TextInputType.number,
              decoration: InputDecoration(hintText: '플레이어 수'),
            ),
            SizedBox(height: 16),
            ElevatedButton(
              onPressed: () {
                int playerCount = int.parse(playerCountController.text);
                Navigator.push(
                  context,
                  MaterialPageRoute(
                    builder: (context) => LadderGamePage(
                      playerCount: playerCount,
                    ),
                  ),
                );
              },
              child: Text('다음'),
            ),
          ],
        ),
      ),
    );
  }
}

class LadderGamePage extends StatefulWidget {
  final int playerCount;

  LadderGamePage({required this.playerCount});

  @override
  _LadderGamePageState createState() => _LadderGamePageState();
}

class _LadderGamePageState extends State<LadderGamePage> {
  List<String> startPoints = [];
  List<String> endPoints = [];

  @override
  void initState() {
    super.initState();
```

```
    for (int i = 0; i < widget.playerCount; ++i) {
      startPoints.add('');
      endPoints.add('');
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('사다리 게임'),
      ),
      body: ListView.builder(
        itemCount: widget.playerCount,
        itemBuilder: (context, index) {
          return ListTile(
            title: Padding(
              padding: const EdgeInsets.all(8.0),
              child: Row(
                children: [
                  Expanded(
                    child: TextField(
                      onChanged: (value) {
                        setState(() {
                          startPoints[index] = value;
                        });
                      },
                      decoration: InputDecoration(hintText: '시작 지점 입력'),
                    ),
                  ),
                  SizedBox(width: 8),
                  Expanded(
                    child: TextField(
                      onChanged: (value) {
                        setState(() {
                          endPoints[index] = value;
                        });
                      },
                      decoration: InputDecoration(hintText: '도착 지점 입력'),
                    ),
                  ),
                ],
              ),
```

```
          ),
        );
      },
    ),
    floatingActionButton: FloatingActionButton(
      onPressed: () {
        // 필요한 경우, 사용자에게 입력이 완료지 않은 셀이 있는지 확인하고 경고 메시지
        를 표시할 수 있습니다.
        for (int i = 0; i < widget.playerCount; ++i) {
          if (startPoints[i].isEmpty) {
            ScaffoldMessenger.of(context).showSnackBar(
              SnackBar(content: Text('모든 시작 지점을 입력해 주세요.')),
            );
            return;
          }
        }

        // 도착 지점을 무작위로 섞습니다.
        List<String> shuffledEndPoints = List.from(endPoints);
        shuffledEndPoints.shuffle();

        Map<String, String> resultMap =
            Map.fromIterables(startPoints, shuffledEndPoints);

        Navigator.push(
          context,
          MaterialPageRoute(
            builder: (context) => LadderGameResultPage(
              playerCount: widget.playerCount,
              resultMap: resultMap,
            ),
          ),
        );
      },
      child: Icon(Icons.check),
    ),
  );
 }
}
```

위의 코드에서는 LadderGamePage 위젯이 playerCount를 인자로 받는다. startPoints와 endPoints는 각각 시작 지점과 도착 지점의 입력값을 저장하는 리스트다. initState에서는 시작 지점과 도착 지점의 초기값을 설정한다. 모든 항목이 채워지면 FloatingActionButton을 눌러 다음 단계를 시작한다. startGame 함수는 시작 버튼을 누르면 입력된 값들이 텍스트로 변경되도록 처리한다. FloatingActionButton에서 'check' 아이콘을 누르면, 도착 지점들은 무작위로 섞여 결과 맵에 저장된다. 그 후, 결과 맵과 플레이어 수를 가지고 LadderGameResultPage로 이동한다.

```
class LadderGameCell extends StatelessWidget {
  final String startPoint;
  final String endPoint;
  final bool isSelected;
  final VoidCallback onTap;
  final ValueChanged<String> onStartPointChanged;
  final ValueChanged<String> onEndPointChanged;

  LadderGameCell({
    required this.startPoint,
    required this.endPoint,
    required this.isSelected,
    required this.onTap,
    required this.onStartPointChanged,
    required this.onEndPointChanged,
  });

  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: onTap,
      child: Container(
        decoration: BoxDecoration(
          border: Border.all(color: Colors.grey),
          color:
              isSelected ? Colors.green.withOpacity(0.3) : Colors.transparent,
        ),
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            TextField(
              onChanged: onStartPointChanged,
              decoration: InputDecoration(hintText: '시작 지점 입력'),
```

```
          ),
          SizedBox(height: 8),
          TextField(
            onChanged: onEndPointChanged,
            decoration: InputDecoration(hintText: '도착 지점 입력'),
          ),
        ],
      ),
    ),
  );
 }
}
```

TextField 위젯에서 onChanged 속성으로 onStartPointChanged와 onEndPointChanged 함수를 전달하여 사용자 입력에 따라 해당 값을 업데이트 할 수 있도록 하였다. 이 코드에서는 LadderGameCell 객체 하나하나가 사다리 게임에서 한 플레이어의 시작 점과 종료 점 정보를 입력받아 저장하는 역할까지 수행한다.

```
class LadderGameResultPage extends StatelessWidget {
  final int playerCount;
  final Map<String, String> resultMap;

  const LadderGameResultPage({
    Key? key,
    required this.playerCount,
    required this.resultMap,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('사다리 게임 결과'),
      ),
      body: ListView.builder(
        itemCount: playerCount,
        itemBuilder: (context, index) {
          String startPoint = resultMap.keys.elementAt(index);
          String endPoint = resultMap[startPoint]!;
          return ListTile(
```

```
          title: Text('$startPoint → $endPoint'),
        );
      },
    ),
  );
 }
}
```

위 코드에서 LadderGameResultPage 객체가 사다리 게임에서 각 플레이어가 어디서 시작해서 어디로 도착했는지 결과 정보를 사용자에게 보여주게 된다.

돌림판게임

```
class RouletteGamePage extends StatefulWidget {
  final List<String> participants;

  RouletteGamePage({required this.participants});

  @override
  _RouletteGamePageState createState() => _RouletteGamePageState();
}

class _RouletteGamePageState extends State<RouletteGamePage>
    with SingleTickerProviderStateMixin {
  AnimationController _animationController;
  double _angle = 0.0;
  int _winnerIndex = -1;
  bool _isSpinning = false;

  @override
  void initState() {
    super.initState();
    _animationController = AnimationController(
      vsync: this,
      duration: Duration(seconds: 5),
    );
    _animationController.addListener(() {
      setState(() {
        _angle = _animationController.value * 2 * pi;
```

```
    });
  });
}

@override
void dispose() {
  _animationController.dispose();
  super.dispose();
}

void _startSpinning() {
  setState(() {
    _isSpinning = true;
  });
  _animationController.forward(from: 0.0).then((_) {
    setState(() {
      _isSpinning = false;
      _winnerIndex = Random().nextInt(widget.participants.length);
    });
  });
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Roulette Game'),
    ),
    body: Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Container(
            width: 200,
            height: 200,
            decoration: BoxDecoration(
              shape: BoxShape.circle,
              color: Colors.red,
            ),
            child: Stack(
              alignment: Alignment.center,
              children: [
                Transform.rotate(
```

```
                angle: _angle,
                child: Container(
                  width: 10,
                  height: 100,
                  decoration: BoxDecoration(
                    color: Colors.white,
                    borderRadius: BorderRadius.circular(5),
                  ),
                ),
              ),
              GestureDetector(
                onTap: !_isSpinning ? _startSpinning : null,
                child: Container(
                  width: 20,
                  height: 20,
                  decoration: BoxDecoration(
                    shape: BoxShape.circle,
                    color: Colors.black,
                  ),
                ),
              ),
            ],
          ),
        ),
        SizedBox(height: 20),
        Text(
          _isSpinning
          ? 'Spinning...'
          : 'Winner: ${widget.participants[_winnerIndex]}',
          style: TextStyle(fontSize: 20),
        ),
      ],
    ),
  ),
  floatingActionButton: FloatingActionButton(
    onPressed: () {
      // 다시 게임을 시작하려면 초기화 필요
      setState(() {
        _winnerIndex = -1;
      });
      _startSpinning();
    },
    child: Icon(Icons.play_arrow),
```

```
      ),
    );
  }
}
```

위의 코드는 돌림판 게임 페이지를 구현한 것이다. 페이지가 생성될 때 입력받은 participants 리스트를 활용하여 원판에 세팅한다. 돌림판을 돌리기 위해 _startSpinning 함수를 사용하고, _animationController를 이용하여 원판이 돌아가는 애니메이션을 구현한다. 원판이 멈추면 랜덤하게 당첨자를 결정하고 결과를 표시한다. 게임을 다시 시작하려면 초기화를 위해 _winnerIndex를 -1로 설정하고 _startSpinning 함수를 호출한다. 이렇게 구현하면 원판이 빙글빙글 돌면서 랜덤한 결과가 나올 수 있게 된다.

몬티홀 딜레마 문제

```
class MontyHallGameScreen extends StatefulWidget {
  final int questionCount;

  MontyHallGameScreen({required this.questionCount});

  @override
  _MontyHallGameScreenState createState() => _MontyHallGameScreenState();
}

class _MontyHallGameScreenState extends State<MontyHallGameScreen> {
  Random _random = Random();
  late List<bool> _doors;
  int? _userSelectedDoor;
  int? _openedDoor;
  int? _finalChoice;
  bool _gameOver = false;

  @override
  void initState() {
    super.initState();
    _initializeDoors();
  }

  void _initializeDoors() {
```

```
    _doors = List.generate(widget.questionCount, (index) => false);
    int randomCarIndex = _random.nextInt(widget.questionCount);
    _doors[randomCarIndex] = true;
  }

  void _selectDoor(int doorIndex) {
    setState(() {
      _userSelectedDoor = doorIndex;
      _openOtherDoor();
    });
  }

  void _openOtherDoor() {
    int carIndex = _doors.indexOf(true);
    int otherDoorIndex;
    do {
      otherDoorIndex = _random.nextInt(widget.questionCount);
    } while (otherDoorIndex == _userSelectedDoor || otherDoorIndex == carIndex);

    setState(() {
      _openedDoor = otherDoorIndex;
    });
  }

  void _makeFinalChoice(int doorIndex) {
    setState(() {
      _finalChoice = doorIndex;
      _gameOver = true;
    });
  }

  Widget _buildDoor(int index) {
    bool isUserSelected = _userSelectedDoor == index;
    bool isCar = _doors[index];
    bool isOpened = _openedDoor == index;
    bool isFinalChoice = _finalChoice == index;

    return GestureDetector(
      onTap: () {
        if (!_gameOver && _userSelectedDoor == null) {
          _selectDoor(index);
        } else if (!_gameOver && _openedDoor != index) {
          _makeFinalChoice(index);
```

```
      }
    },
    child: Container(
      width: 100,
      height: 200,
      margin: EdgeInsets.all(8),
      decoration: BoxDecoration(
        border: Border.all(width: 2),
        color: isOpened ? Colors.red : Colors.white,
      ),
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          if (isOpened || isFinalChoice)
            Icon(
              isOpened
                  ? Icons.cancel
                  : isCar
                      ? Icons.car_rental
                      : Icons.sentiment_very_satisfied,
              size: 60,
              color: isUserSelected || isFinalChoice
                  ? Colors.blue
                  : Colors.black,
            ),
          SizedBox(height: 8),
          Text(
            isOpened
                ? isCar
                    ? '성공'
                    : '실패'
                : '문 ${index + 1}',
            style: TextStyle(
              fontWeight: FontWeight.bold,
              fontSize: 16,
            ),
          ),
        ],
      ),
    ),
  );
}
```

```
Widget _buildGameBoard() {
  return GridView.builder(
    shrinkWrap: true,
    physics: NeverScrollableScrollPhysics(),
    gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
      crossAxisCount: 2,
      crossAxisSpacing: 16,
      mainAxisSpacing: 16,
    ),
    itemCount: widget.questionCount,
    itemBuilder: (context, index) {
      return _buildDoor(index);
    },
  );
}

Widget _buildResult() {
  bool isCarBehindFinalChoice = _finalChoice != null && _doors[_finalChoice!];

  return Column(
    children: [
      SizedBox(height: 16),
      Text(
        '게임 결과',
        style: TextStyle(
          fontSize: 20,
          fontWeight: FontWeight.bold,
        ),
      ),
      SizedBox(height: 16),
      Icon(
        isCarBehindFinalChoice
            ? Icons.sentiment_very_satisfied
            : Icons.cancel,
        size: 60,
        color: isCarBehindFinalChoice ? Colors.green : Colors.red,
      ),
      SizedBox(height: 8),
      Text(
        isCarBehindFinalChoice ? '성공' : '실패',
        style: TextStyle(
          fontWeight: FontWeight.bold,
          fontSize: 16,
```

```
        ),
      ),
    ],
  );
 }

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(
    title: Text('Monty Hall Dilemma'),
   ),
   body: Center(
    child: Column(
     mainAxisAlignment: MainAxisAlignment.center,
     children: [
      if (!_gameOver)
       Text(
        '문을 선택하세요',
        style: TextStyle(fontSize: 18),
       ),
      if (_gameOver) _buildResult(),
      _buildGameBoard(),
      if (_gameOver)
       ElevatedButton(
        onPressed: () {
         setState(() {
          _initializeDoors();
          _userSelectedDoor = null;
          _openedDoor = null;
          _finalChoice = null;
          _gameOver = false;
         });
        },
        child: Text('새 게임 시작'),
       ),
     ],
    ),
   ),
  );
 }
}
```

위의 코드는 MontyHallGameScreen 위젯을 구현한 것이다. _initializeDoors() 함수를 통해 각 문에 대한 설정을 하고, _selectDoor() 함수를 통해 사용자가 문을 선택하고 열리지 않은 다른 문을 열어 준다. 그리고 사용자가 최종 선택을 하면 게임 결과를 보여 준다. 게임이 종료된 후에는 새 게임을 시작할 수 있는 버튼이 나타난다.

로또 게임

```
class LottoGameScreen extends StatefulWidget {
  final int lottoRank;

  LottoGameScreen({required this.lottoRank});

  @override
  _LottoGameScreenState createState() => _LottoGameScreenState();
}

class _LottoGameScreenState extends State<LottoGameScreen> {
  Random _random = Random();
  int _consecutiveSuccess = 0;
  int _remainingAttempts = 0;
  bool _gameOver = false;

  @override
  void initState() {
    super.initState();
    _initializeGame();
  }

  void _initializeGame() {
    switch (widget.lottoRank) {
      case 1:
        _remainingAttempts = 23;
        break;
      case 2:
        _remainingAttempts = 20;
        break;
      case 3:
        _remainingAttempts = 15;
        break;
      case 4:
```

```
      _remainingAttempts = 9;
      break;
    case 5:
      _remainingAttempts = 5;
      break;
  }
}

void _checkAttempt(bool isSuccess) {
  setState(() {
    if (isSuccess) {
      _consecutiveSuccess++;
      _remainingAttempts--;
    } else {
      _gameOver = true;
    }

    if (_consecutiveSuccess == widget.lottoRank) {
      _gameOver = true;
    }
  });
}

Widget _buildGameBoard() {
  return Column(
    children: [
      Text(
        '연속 성공 횟수: $_consecutiveSuccess',
        style: TextStyle(fontSize: 18),
      ),
      SizedBox(height: 16),
      Text(
        '남은 클리어 횟수: $_remainingAttempts',
        style: TextStyle(fontSize: 18),
      ),
      SizedBox(height: 32),
      if (!_gameOver)
        Row(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            ElevatedButton(
              onPressed: () {
                bool isSuccess = _random.nextBool();
```

```
                _checkAttempt(isSuccess);
              },
              child: Text('O'),
            ),
            SizedBox(width: 16),
            ElevatedButton(
              onPressed: () {
                _checkAttempt(false);
              },
              child: Text('X'),
            ),
          ],
        ),
      if (_gameOver)
        Text(
          '게임 오버',
          style: TextStyle(fontSize: 20, fontWeight: FontWeight.bold),
        ),
    ],
  );
 }

 @override
 Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Lotto Game'),
    ),
    body: Center(
      child: Padding(
        padding: EdgeInsets.all(16),
        child: _buildGameBoard(),
      ),
    ),
  );
 }
}
```

- _initializeGame 메서드는 게임을 초기화한다. 주어진 로또 등급에 따라 연속 성공 횟수와 남은 클리어 횟수를 설정한다.
- _checkAttempt 메서드는 사용자의 시도 결과를 확인하고, 연속 성공 횟수와 남은 클리어 횟수를 업데이트한다. 만약 실패 버튼을 누른 경우나 클리어 횟수를 달성한 경우에는 게임 오버 처리된다.
- _buildGameBoard 메서드는 게임 보드를 구성하는 위젯을 반환한다. 연속 성공 횟수와 남은 클리어 횟수를 화면에 표시하고, O와 X 버튼을 눌러 시도를 수행할 수 있다. 게임 오버 상태일 때는 게임 오버 메시지를 표시한다.

로고 및 앱 이름 설정

로고 설정

로고 이미지를 사용하려면 해당 이미지 파일을 준비해야 한다. 로고 이미지를 앱에 적용하기 위해서는 이미지 파일을 프로젝트의 assets 폴더에 추가하고, pubspec.yaml 파일에도 해당 이미지를 등록해야 한다. pubspec.yaml 파일의 flutter 섹션에 assets 항목을 추가하고, 로고 이미지 파일의 경로를 지정한다. 다음은 이미지 파일의 경로를 지정하는 코드다.

```
flutter:
  assets:
    - assets/logo.png
```

앱 이름 설정

앱 이름은 앱의 main.dart 파일에서 설정된다. main.dart 파일의 main() 함수 내에서 runApp() 메서드를 호출할 때, MaterialApp 위젯의 title 속성을 설정하여 앱 이름을 지정한다. 다음은 확률 게임의 이름을 지정하는 코드다.

```
void main() {
  runApp(
    MaterialApp(
      title: '랜덤게임', // 앱 이름 설정
      home: MyHomePage(),
    ),
  );
}
```

로고 이미지와 앱 이름을 설정한 후, 해당 로고 이미지는 앱 아이콘이나 앱 화면 상단에 사용될 수 있으며, 앱 이름은 앱 런처 아이콘, 앱 스토어, 앱 화면 타이틀 등 다양한 곳에서 표시된다.