实战 1.2 带简易 AI 的黑白棋人机对战游戏

一、实验目的

1. 了解 DevEco Studio 的使用
2. 学习 ArkTS 语言及 ArkUI 组件
3. 编写代码
4. 编译运行
5. 在模拟器上运行

二、实验原理

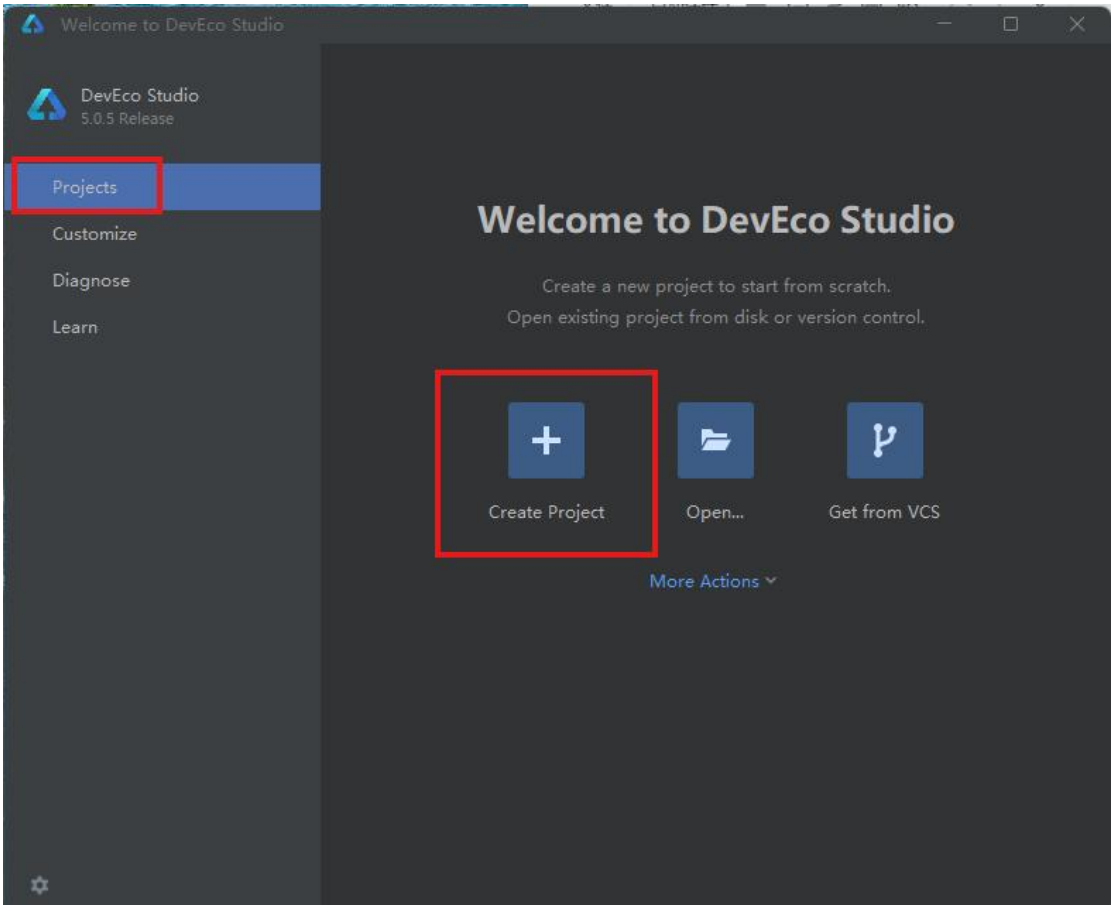
1. 鸿蒙开发原理
2. ArkTS，ArkUI 开发原理
3. 鸿蒙应用运行原理

三、实验仪器材料

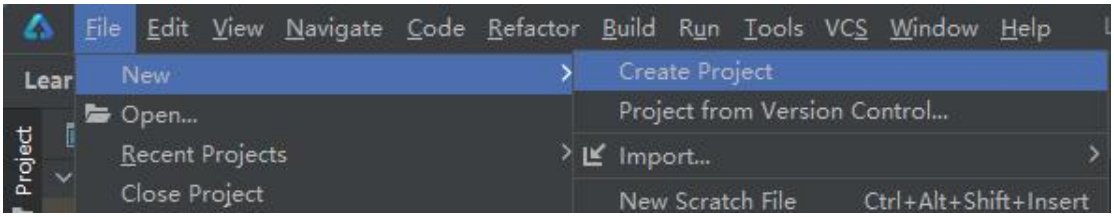
1. 计算机实训室电脑一台
2. DevEco Studio 开发环境及鸿蒙手机模拟器

四、实验步骤

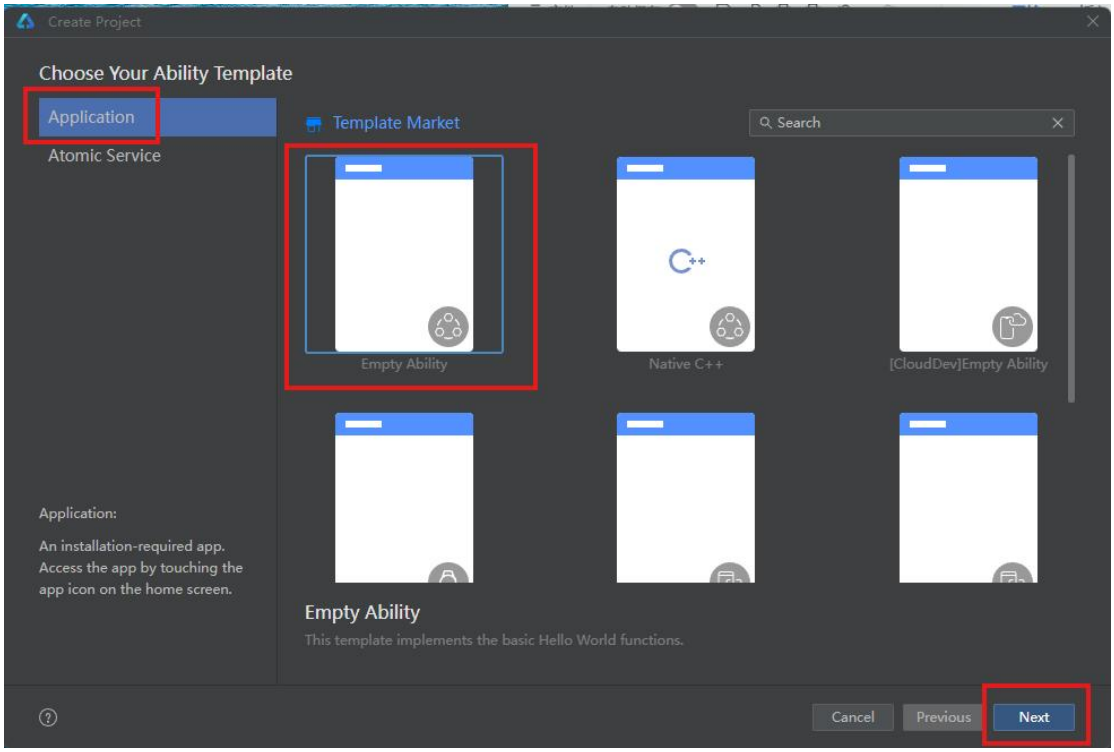
1. 打开 DevEco Studio，点击 Create Project 创建工程。



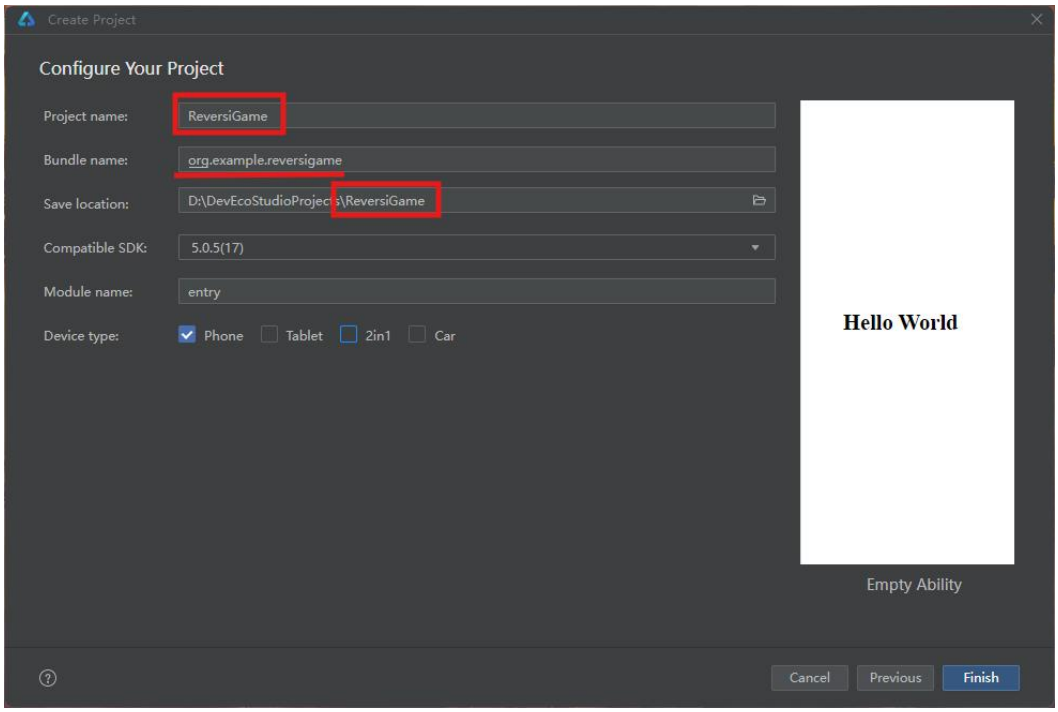
如果已经打开了一个工程，请在菜单栏选择 File > New > Create Project 来创建一个新工程。



点击“Create Project”，进入：



选择 Application，然后选择“Empty Ability”，点击 Next 按钮，进入：



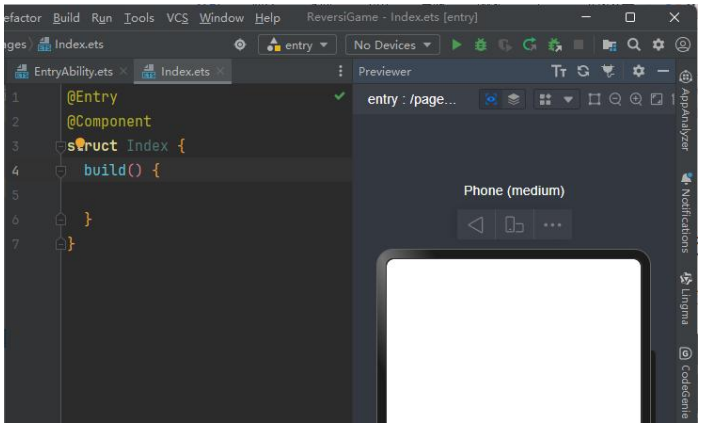
设置项目名称为 ReversiGame（你可以取自己喜欢的名称），存放地址等，然后点击 Finish。

2. 理解 @State 注解

对 Index.ets 进行一些修改：

- 清理界面

在 Index.ets 文件的 Index 结构中，首先删除掉 message 这个成员变量，清理掉 build() 函数中原来绘制 “Hello World”的代码，此时刷新 Previewer,界面会变成空白。



在 Index 结构中 build 函数前创建一个 @State 注解的变量 counter，代码如下：

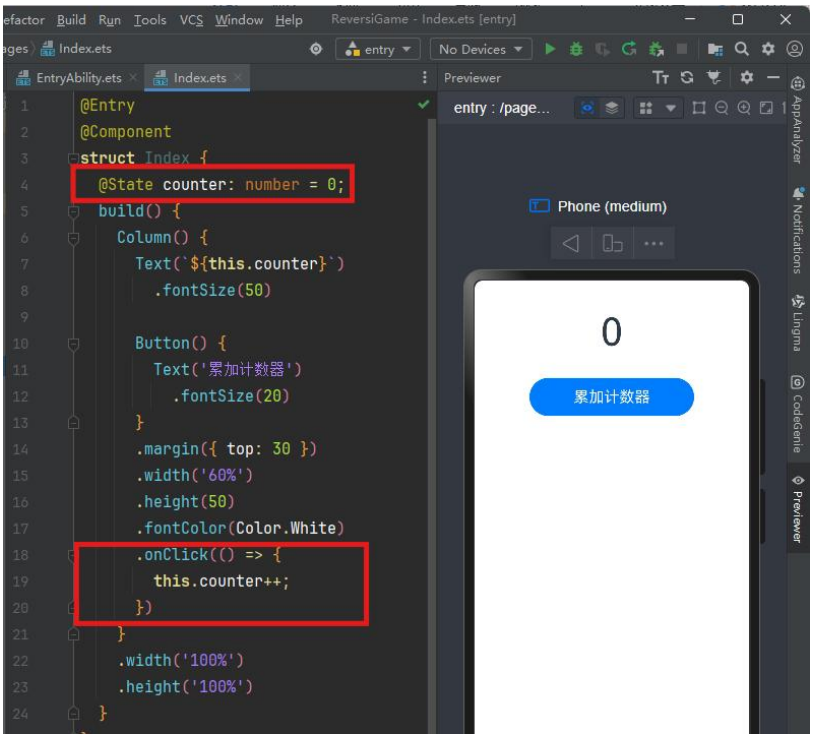
```
  @State counter: number = 0;
```

然后在界面构建函数 build() 中绘制一个 Text 控件、一个 Button 控件，并在 Button 后面添加按钮点击以后的事件响应代码，让 counter 计数器自增加一。完整的代码如下：

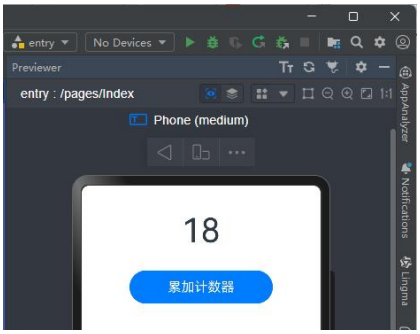
```
@Entry
@Component
struct Index {
  @State counter: number = 0;
  build() {
    Column() {
      Text(`${this.counter}`)
        .fontSize(50)

      Button() {
        Text('累加计数器')
          .fontSize(20)
      }
      .margin({ top: 30 })
      .width('60%')
      .height(50)
      .fontColor(Color.White)
      .onClick(() => {
        this.counter++;
      })
    }
    .width('100%')
    .height('100%')
```

```
  }
}
```

完成后截图如下：



此时，我们点击按钮，会发现界面上显示的数字会随着按钮点击次数增加，每点击一次按钮，数字加一。如图：



这段 ArkTS 代码展示了一个基础的计数器应用，是 @State 装饰器含义和用法的经典案例。

在 ArkUI 框架中，**@State**（状态变量）是实现**声明式 UI** 的核心机制之一。它的主要作用是建立数据与视图之间的单向或双向绑定关系，从而让 UI 能够响应数据的变化。

1. @State 的含义

(1) @State 装饰器用于修饰组件内部的变量，将其标记为一个状态变量（State Variable）。

(2) 状态（State）：指组件在某一时刻的数据或配置。核心作用是：一旦这个被 @State 装饰的变量的值发生变化，框架会自动检测到这个变化，并只重新渲染（刷新）视图中依赖于

该变量的部分，保持数据与 UI 的同步。

2. 解读：

(1) 当用户点击 Button 时，执行回调函数，将 this.counter 的值加 1。

(2) 重点：一旦 this.counter 被修改，ArkUI 的状态管理机制就会被触发，自动执行依赖更新，顺序如下：

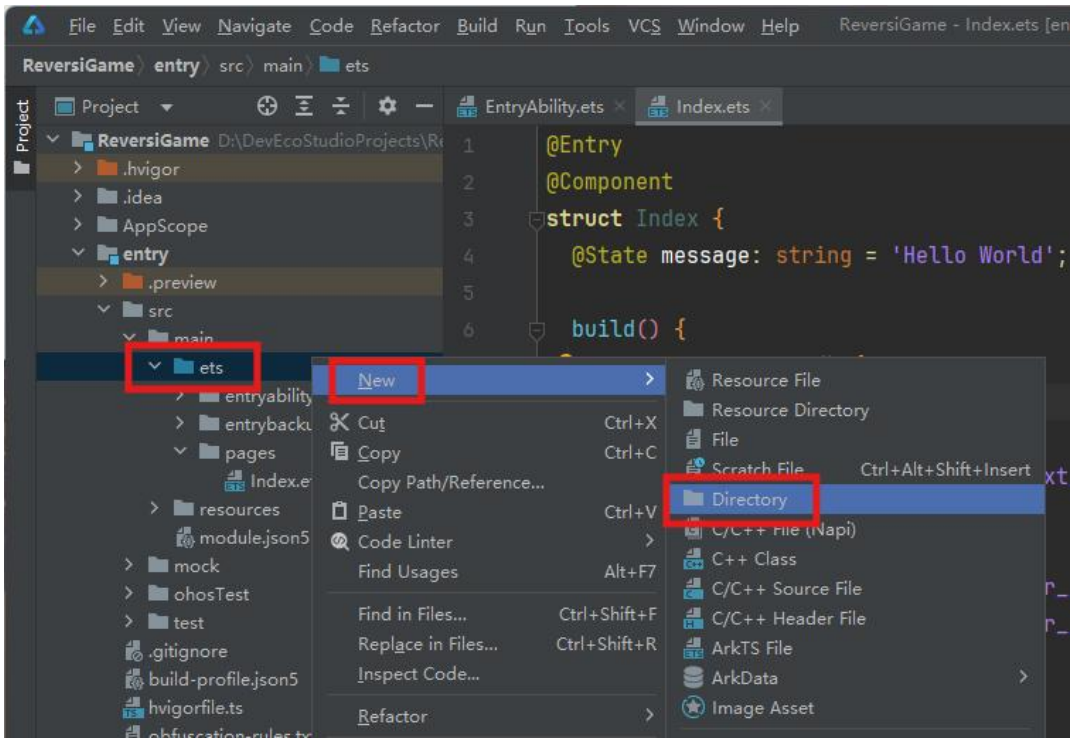
① 检测到 this.counter 的值变化。

② 识别到 Text 组件依赖于 this.counter。

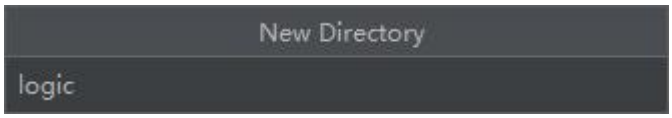
③ 只重新渲染 Text 组件，将其显示更新为新的数值，从而实现了 UI 的自动刷新。

依赖于 ArkUI 的声明式 UI 特性，我们可以非常容易地将黑白棋的逻辑数据，绑定到界面上，通过直接修改被绑定的数据，完成界面元素的显示。

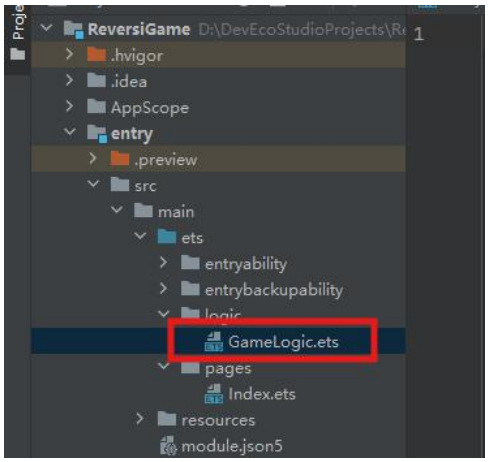
3. 创建与界面分离的游戏业务逻辑模块



在项目的“ets”目录上点击鼠标右键，在弹出菜单中选择“New”，“Directory”创建目录，目录名为“logic”。



然后在 logic 目录下创建游戏逻辑文件“GameLogic.ets”，注意扩展名是.ets。



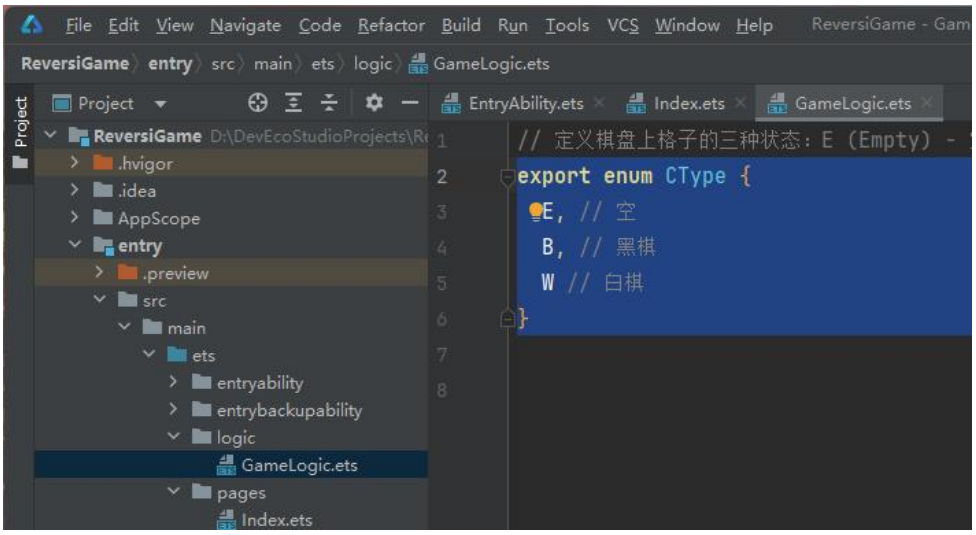
后续过程中，游戏的数据逻辑相关代码都保存在这个 GameLogic.ets 文件中。

- 创建数据类型

在 GameLogic.ets 中创建保存玩家角色和棋盘棋子类型的数据结构 CType，并且导出这个数据类型：

```
// 定义棋盘上格子的三种状态：E (Empty) - 空，B (Black) - 黑棋，W (White) - 白棋
export enum CType {
  E, // 空
  B, // 黑棋
  W // 白棋
}
```

添加后效果如下：



4. **绘制基本棋盘**

- 导入数据类型

在 Index.ets 的第一行，导入 GameLogic 中的 CType 类型

```
import { CType } from '../logic/GameLogic';
```

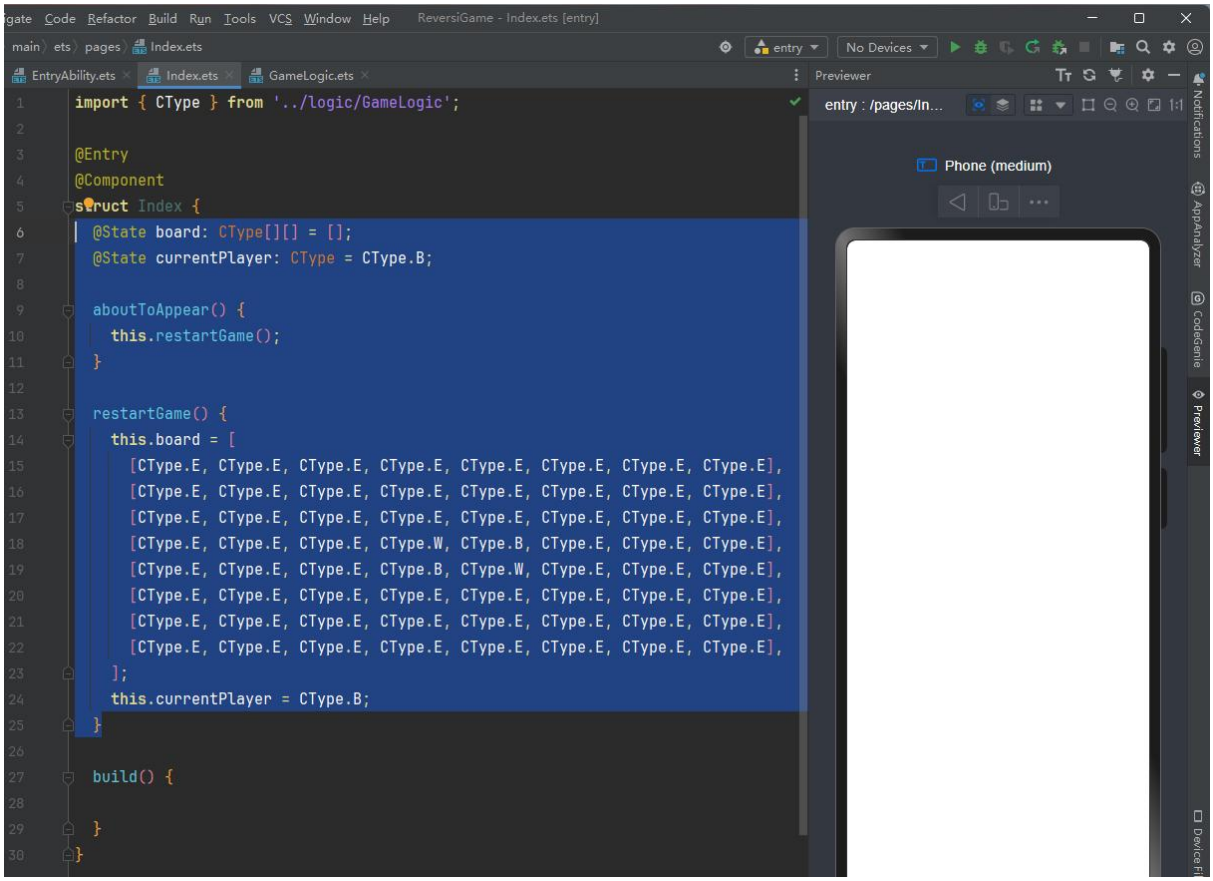
- 创建初始化棋盘数据的函数

在 Index 这个结构中，删除掉计数器 counter 这个变量，并且清理掉 build() 中我们绘制计数器的代码。

取而代之的是添加一个棋盘变量 board 以及当前玩家变量 currentPlayer。然后创建一个 restartGame 函数用来初始化棋盘的棋子数据以及当前玩家。接着在 aboutToAppear() 函数中调用这个函数。代码如下：

```
  @State board: CType[][] = [];
  @State currentPlayer: CType = CType.B;

  aboutToAppear() {
    this.restartGame();
  }

  restartGame() {
    this.board = [
      [CType.E, CType.E, CType.E, CType.E, CType.E, CType.E, CType.E, CType.E],
      [CType.E, CType.E, CType.E, CType.E, CType.E, CType.E, CType.E, CType.E],
      [CType.E, CType.E, CType.E, CType.E, CType.E, CType.E, CType.E, CType.E],
      [CType.E, CType.E, CType.E, CType.W, CType.B, CType.E, CType.E, CType.E],
      [CType.E, CType.E, CType.E, CType.B, CType.W, CType.E, CType.E, CType.E],
```

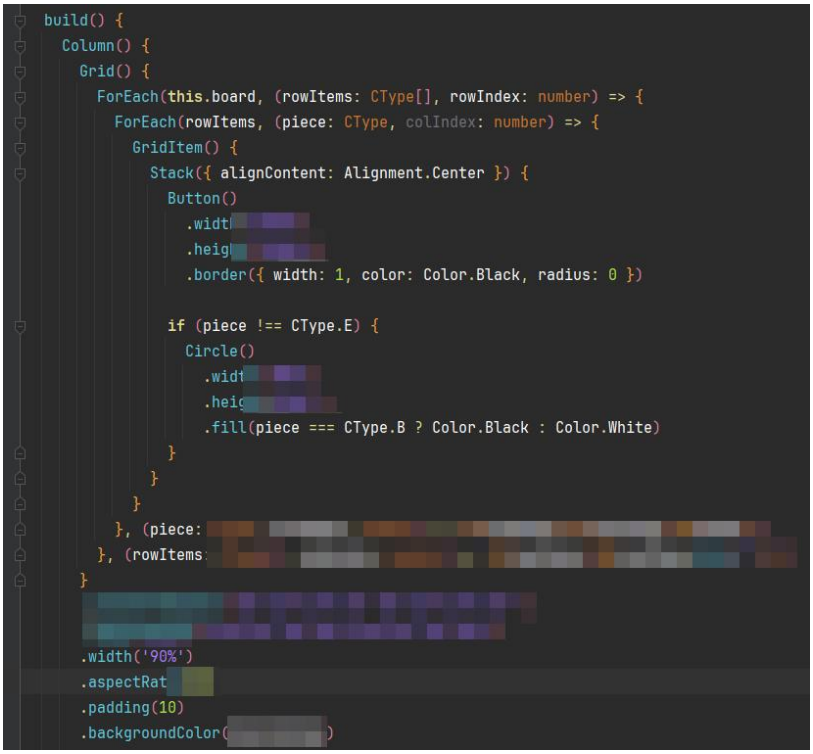
```
      [CType.E, CType.E, CType.E, CType.E, CType.E, CType.E, CType.E, CType.E],
      [CType.E, CType.E, CType.E, CType.E, CType.E, CType.E, CType.E, CType.E],
      [CType.E, CType.E, CType.E, CType.E, CType.E, CType.E, CType.E, CType.E],
    ];
    this.currentPlayer = CType.B;
  }
```



我们看到创建了一个 8*8 的二维数组 board 用来保存棋盘数据，并设定当前准备落子的玩家执黑子。

- 绘制棋盘

在 Index.ets 的 build 函数中，添加一个 Column 布局组件，其中添加 Grid 组件，使用 ForEach 双重循环遍历 board 数组中的数据，判断数据的类型并决定显示棋子的颜色。关键代码截图如下：



注意代码中用到了 Stack 界面组件，在同一个格子中使用堆叠技术根据棋盘数据显示不同颜色的棋子。

- 增加游戏信息

当前的棋盘界面有点单薄，所以我们在棋盘上方添加游戏信息，在棋盘下方显示当前棋手的信息以及得分。在界面中增加如下代码保存游戏名称：

```
  private gameName: string = '黑白棋';
```

效果如下：

然后在 Grid() 组件的上方，添加显示游戏名称的代码：

```
      Text(this.gameName)
        .fontSize(32)
```

```
        .fontWeight(FontWeight.Bold)
        .fontColor('#2C3E50')
        .padding({ top: 40, bottom: 20 })
```

紧接着添加统计双方分数的变量，记得用@State 注解来声明两个变量：whiteScore、blackScore。

紧接着在界面上合适的地方添加界面控件显示计分板数据，代码自行实现。

5. 为游戏添加玩法逻辑

- 添加棋盘点击处理逻辑

当玩家点击棋盘落子区域时候，我们要判断点击的位置是否是一个合法的落子点。根据黑白棋规则，只能将棋子落在可以反转对手棋子的位置（当无法落子的时候则跳过）。因此我们需要增加判断落子位置是否合法的检查逻辑，并且需要增加相关的支持数据结构。

我们在 GameLogic.ets 中添加描述坐标的类 Coordinate、checkValidMove 函数的返回类 MoveCheckResult、游戏的主逻辑类 GameLogic，以及并且导出这些类。

这些类的代码在“实战 1.1-命令行带 AI 的黑白棋游戏-实验指导书.pdf”中。

- 在界面上处理点击逻辑

修改 Index.ets 文件中的导入代码，增加导入 GameLogic 中新定义的类：

```
import { CType, GameLogic, Coordinate } from '../logic/GameLogic';
```

然后在 Index 结构中，增加一个 GameLogic 的私有成员变量：

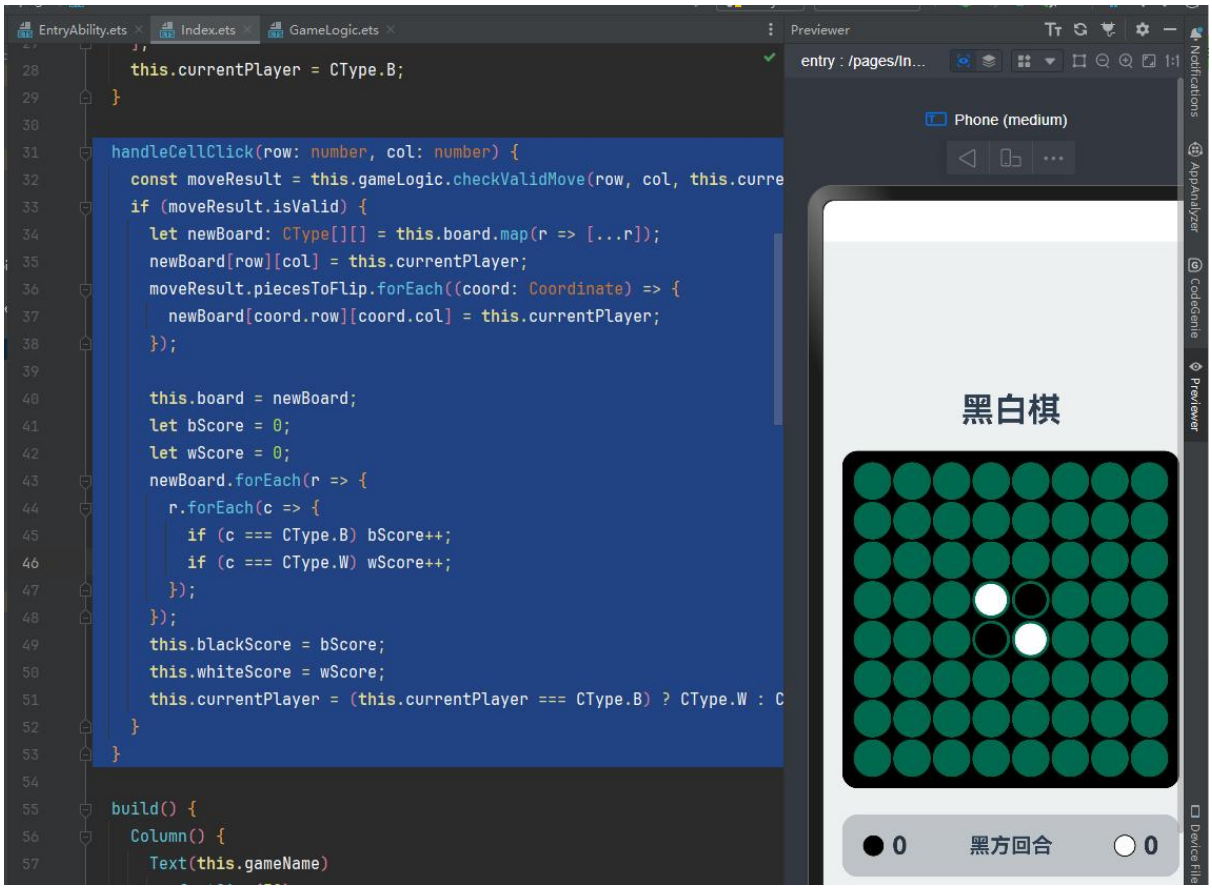
```
  private gameLogic: GameLogic = new GameLogic();
```

导入外部类，并添加了成员变量后的效果：

接着在 Index 增加一个 handleCellClick 函数用以响应每个 Grid 单元格里 Button 被点击后处理逻辑，在处理函数中完成数据棋盘上的棋子翻转、计分统计和玩家换手的功能。

```
  handleCellClick(row: number, col: number) {
    const moveResult = this.gameLogic.checkValidMove(row, col,
this.currentPlayer, this.board);
    if (moveResult.isValid) {
      let newBoard: CType[][] = this.board.map(r => [...r]);
      newBoard[row][col] = this.currentPlayer;
      moveResult.piecesToFlip.forEach((coord: Coordinate) => {
        newBoard[coord.row][coord.col] = this.currentPlayer;
      });

      this.board = newBoard;
      let bScore = 0;
      let wScore = 0;
      newBoard.forEach(r => {
        r.forEach(c => {
          if (c === CType.B) bScore++;
          if (c === CType.W) wScore++;
        });
      });
      this.blackScore = bScore;
      this.whiteScore = wScore;
      this.currentPlayer = (this.currentPlayer === CType.B) ? CType.W : CType.B;
    }
  }
```

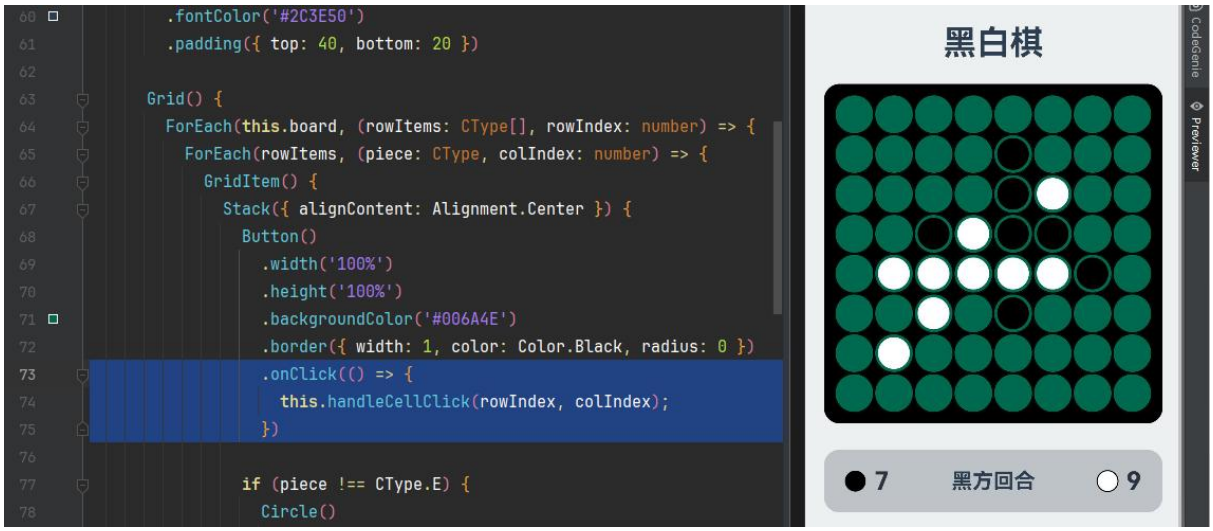
添加后的代码：



紧接着修改 build 函数，给 Grid 中的每个 Button 添加 onClick 事件的响应函数，相关代码如下：

```
                .onClick(() => {
                  this.handleCellClick(rowIndex, colIndex);
                })
```

添加后的效果：



刷新 Previewer 后，我们就可以点击棋盘的方格在棋盘落子，此时系统会自动切换玩家所持的颜色，并且自动计算黑白双方的分数，完成的是两个人类玩家对弈的规则。

至此，我们就完成了一个最基本的黑白棋游戏。但是当前这个简易版的黑白棋有一些缺点：

1. 没有游戏结束的判断逻辑。
2. 不带 AI 人机对战功能。

我们修改游戏的名字为：“学生姓名-黑白棋”：

6. 为游戏添加 AI 对手

为了让游戏更有趣，我们可以给代码添加简单的 AI 算法，并将游戏改为人机对战模式。如果想尝试更强的棋力，可以考虑自行研究其他算法，本实验指导不做过多讨论。

至此我们完成了一个有 AI 智能的黑白棋对战游戏。

五、实验注意事项

1. 注意教师的操作演示。

2. 学生机与教师机内网连通，能接收和提交实验结果。

3. 按实验要求输入测试数据，并查看输出结果是否符合实验结果。

六、思考题

1. 通过这个实验，你学到了什么？