```
package com.example.appdev
import android.annotation.SuppressLint
import androidx.compose.foundation.border
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.layout.width
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.Color.Companion.Black
import androidx.compose.ui.graphics.Color.Companion.White
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.activity.enableEdgeToEdge
import androidx.compose.material3.Scaffold
import androidx.compose.foundation.layout.Spacer
import com.example.myapplication.ui.theme.MyApplicationTheme
import androidx.compose.ui.Alignment
import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.unit.TextUnit
import androidx.compose.ui.unit.sp


class MainActivity : ComponentActivity() {
    @SuppressLint("UnusedMaterial3ScaffoldPaddingParameter")
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            MyApplicationTheme {
                Scaffold(modifier = Modifier.fillMaxSize()){
                    GameplayScreen(
                        playerShips = listOf(Ship(listOf(11, 12, 13))),
                        opponentShips = listOf(Ship(listOf(44, 45))),
                        onAttack = { row, col -> println("Attacked ($row, $col)") },
                        onDefend = { println("Defend action triggered") },
                        attackedTiles = mapOf(11 to "hit", 55 to "miss"),
                        onReset = { println("Game Reset") },
                        backgroundColor = Color.Red,
```

```
                                isPlayerOneTurn = true
                            )
                        }
                    }
                }
            }
        }
// Define Ship class
data class Ship(val positions: List<Int> = emptyList())


private fun <ColumnScope> ColumnScope.Text(text: String, fontSize: TextUnit, fontWeight:
FontWeight) {

}

fun <Ship> GameGrid(ships: List<Ship>, attackedTiles: Map<Int, String>, onTileClick: (Int,
Int) -> Unit) {

}

@Composable
fun <Ship : Any, Color> GameplayScreen(
    playerShips: List<Ship>,              // List of your ships
    opponentShips: List<Ship>,            // List of opponent's ships (hidden)
    onAttack: (Int, Int) -> Unit,      // Function when an attack happens
    onDefend: () -> Unit,                 // Function for defense action
    attackedTiles: Map<Int, String>, // Tracks attack history
    onReset: () -> Unit,                  // Function to reset the game
    backgroundColor: Color,
    isPlayerOneTurn:Boolean,
)

{
    Column(
        modifier = with(Modifier) {
            fillMaxSize()                    // Fill entire screen //
                .background(Black) // Set background color based on turn
                .padding(16.dp)
        },                    // Apply padding around UI
        horizontalAlignment = Alignment.CenterHorizontally) {
        // Text showing whose turn it i
        Text(
            text = "Player ${if (backgroundColor == Color(0xFFFF6666.toInt())) 1 else 2}'s
Turn",
            fontSize=24.sp,               // Set text size
            fontWeight = FontWeight.Bold // Make it bold
            // Use white text for contrast
        )

        Spacer(modifier = Modifier.height(16.dp)) // Add space below text

        // Opponent's grid (for attack)
        GameGrid(
            ships =opponentShips,       // Show opponent's ships
            attackedTiles = attackedTiles, // Highlight attacked positions
            onTileClick = onAttack       // Attack happens when clicking tiles
```

```
            )

            Spacer(modifier = Modifier.height(16.dp)) // Add spacing

            // Player's own grid (just display ships)
            GameGrid(
                ships = playerShips,       // Show player's own ships
                attackedTiles = emptyMap(),    // No attack tracking here
                onTileClick = { _, _ -> } // Disable clicking on own board
            )

            Spacer(modifier = Modifier.height(16.dp)) // More spacing
            val selectedTiles by remember{mutableStateOf(mutableSetOf<Int>())}
            GameGrid(
                ships = playerShips,
                attackedTiles = emptyMap(),
                onTileClick = { row, col ->
                    val position = row * 10 + col
                    if (selectedTiles.size < 3) selectedTiles.add(position)
                }
            )

            Spacer(modifier = Modifier.height(16.dp))


            // Action Buttons (Attack, Defend, Reset)
            Row {
                Button(
                    onClick = { /* Attack happens via grid click */ },
                    colors          =          ButtonDefaults.buttonColors(containerColor          =
Color(0xFFFF4444.toInt()))
                ) {
                    Text(text = "ATTACK", color= White) // Attack button
                }

                Spacer(modifier = Modifier.width(16.dp)) // Space between buttons

                Button(
                    onClick = onDefend,
                    colors          =          ButtonDefaults.buttonColors(containerColor          =
Color(0xFF888888.toInt()))
                ) {
                    Text(text = "DEFEND", color = White) // Defend button
                }

                Spacer(modifier = Modifier.width(16.dp))

                Button(
                    onClick = onReset,
                    colors          =          ButtonDefaults.buttonColors(containerColor          =
Color(0xFF444444.toInt()))
                ) {
                    Text(text = "RESET", color = White) // Reset button
                }
            }
        }
}
```

```
@Composable
fun GameGrid(
    ships: List<Ship>,                      // List of ships on this grid
    attackedTiles: Map<Int, String>,    // Map of attacked positions (shows hit/miss)
    onTileClick: (Int, Int) -> Unit      // Function triggered when a tile is clicked
) {
    Column(
        modifier = Modifier
            .fillMaxWidth()                    // Make grid span full width
            .padding(8.dp),                     // Provide padding for layout
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        // Create a 10x10 grid using LazyColumn & LazyRow
        LazyColumn {
            items(10) { row ->                 // Loop through rows
                Row {
                    (0 until 10).forEach { col -> // Loop through columns
                        val position = row * 10 + col    // Calculate unique position ID

                        Box(
                            modifier = Modifier
                                .size(40.dp)                  // Set tile size
                                .border(1.dp, Black) // Add a border
                                .background(
                                    when {
                                        attackedTiles[position] == "hit" -> Color.Red    //
Hit marker
                                        attackedTiles[position] == "miss" -> Color.Gray
// Miss marker
                                        ships.any { it.positions.contains(position) } ->
Color.Blue

                                        else -> White    // Default tile
                                    }
                                )
                                .clickable { onTileClick(row, col) },    // Click to attack this
tile
                            contentAlignment = Alignment.Center
                        ) {
                            Text(
                                text = if (attackedTiles.containsKey(position))
attackedTiles[position]!! else "",
                                Color =White,
                                fontSize = 14
                            )
                        }
                    }
                }
            }
        }
    }
}

private fun <BoxScope> BoxScope.Text(text: String, Color: Color, fontSize: Int) {

}
```

```
fun onAttack(row: Int, col: Int, attackedTiles: MutableMap<Int, String>, playerShips: List<Ship>,
switchTurn: () -> Unit) {
    val position = row * 10 + col // Calculate grid position
    val isHit = playerShips.any { it.positions.contains(position) } // Check if hit
    attackedTiles[position] = if (isHit) "hit" else "miss" // Store result
    switchTurn() // Switch turn after attack
}


@Preview(showBackground = true)
@Composable
fun GamePlayScreenPreview() {
        GameplayScreen(
            playerShips = listOf(Ship(listOf(11, 12, 13))), // Example ship positions
            opponentShips = listOf(Ship(listOf(44, 45))), // Example opponent ship
positions
            onAttack = { row, col -> println("Attacked ($row, $col)") }, // Simulate attack
action
            onDefend = { println("Defend action triggered") }, // Simulate defense action
            attackedTiles = mapOf(11 to "hit", 55 to "miss"), // Sample attack results
            onReset = { println("Game Reset") }, // Simulate game reset
            backgroundColor = Color.Red,
            isPlayerOneTurn = true // Example turn indicator (Player 1)
        )
    }
```