

# 5COSC023W - Tutorial 7 Exercises - Sample Solutions

## 1 Extending the Colour Game

Simply save the `colour_chosen` and restore it in the `onCreate`. Do not forget to initialise it again at the first line of the GUI function with the restored value.

## 2 A Tic-Tac-Toe Game

Add the following colour in the `colors.xml` resources:

```
<color name="yellow">#FFFFCC00</color>
```

The main code (based on the skeleton provided) is:

```
package com.example.tictactoe_composableapp

import android.content.Context
import android.os.Bundle
import android.widget.Toast
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.layout.wrapContentSize
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.runtime.toMutableStateList
import androidx.compose.ui.Alignment
```

```

import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.RectangleShape
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.res.colorResource
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp

// dimensions of the board
val rows = 3
val cols = 3

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            GUI()
        }
    }
}

@Preview
@Composable
fun GUI() {
    var grid = remember{MutableList(rows*cols) {i -> ' '}.toMutableStateList()}
    var gameIsPlayable by remember{ mutableStateOf(true) }

    Column(modifier = Modifier
        .fillMaxSize()
        .padding(top = 100.dp),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.spacedBy(0.dp)) {

        val context = LocalContext.current
        // create 3 rows of buttons
        for (r in 0..rows-1) {
            Row {
                // each row has 3 buttons (columns)
                for (c in 0..cols-1) {
                    var index = r*rows + c
                    Button(shape = RectangleShape,
                        colors = ButtonDefaults.buttonColors(containerColor =
                            colorResource(id = R.color.yellow)),
                        modifier = Modifier
                            .size(width = 80.dp, height = 80.dp)
                            .padding(1.dp),
                        enabled = gameIsPlayable,

```

```

onClick = {
    // human move
    if (grid[index] == ' ') {
        grid[index] = 'X'

        val winner = checkWinner(grid)
        if (winner == 'X' || winner == 'O') {
            displayGameOver(winner, context)
            gameIsPlayable = false
        }
        else {
            // now it is the turn of the computer player.
            // Initially this is a naive player choosing
            // a random cell among the available ones
            var chosen_cell = ComputerMove(grid)

            // if board is not full play the move
            if (chosen_cell != -1) {
                grid[chosen_cell] = 'O'

                val winner = checkWinner(grid)
                if (winner == 'X' || winner == 'O') {
                    displayGameOver(winner, context)
                    gameIsPlayable = false
                }
            }
            else {
                displayGameOver('-', context)
                gameIsPlayable = false
            }
        }
    }
    else
        Toast.makeText(context,
            "Move is invalid", Toast.LENGTH_SHORT).show()
    ) {
        Text(text = ""+grid[index], fontSize = 32.sp, color = Color.Black)
    }
}
}
}
}

// initially make a random move for the computer and return
// the index of the move. If the board is full return -1
fun ComputerMove(board: MutableList<Char>): Int {

```

```

var emptySlots = mutableListOf<Int>()

for (i in 0..board.size-1) {
    if (board[i] == ' ')
        emptySlots.add(i)
}

// return one of the empty slots randomly, otherwise if it is empty return a -1
if (emptySlots.isEmpty())
    return -1
else
    return emptySlots.random()
}

fun displayGameOver(winner: Char, context: Context) {
    var message = "GAME OVER! "
    if (winner == '-')
        message += "It is a draw"
    else
        message += "$winner wins"

    Toast.makeText(context, message, Toast.LENGTH_SHORT).show()

    /* Check if there was a winner and return 'X' or 'O' for the winner,
    otherwise it returns '-'. */
    fun checkWinner(board: MutableList<Char>): Char {
        // how many consecutive Xs we have during a check in a row, col or diagonal
        var countX = 0
        // how many consecutive Os we have during a check in a row, col or diagonal
        var countO = 0

        // check the rows first
        for (r in 0..2) {
            for (c in 0..2) {
                var index = row_col2Index(r, c)
                if (board[index] == 'X')
                    ++countX
                else if (board[index] == 'O')
                    ++countO
            }

            if (countX == 3)
                return 'X'
            else if (countO == 3)
                return 'O'
        }
    }
}

```

```

        countX = 0
        countO = 0
    }

    // check the columns for a winner
    for (c in 0..2) {
        for (r in 0..2) {
            var index = row_col2Index(r, c)
            if (board[index] == 'X')
                ++countX
            else if (board[index] == 'O')
                ++countO
        }

        if (countX == 3)
            return 'X'
        else if (countO == 3)
            return 'O'

        countX = 0
        countO = 0
    }

    // check top-left to bottom-right diagonal
    if (board[0] == board[4] && board[4] == board[8])
        // check that the diagonal cells are not empty
        if (board[0] == 'X' || board[0] == 'O')
            return board[0]

    // check top-right to bottom-left diagonal
    if (board[2] == board[4] && board[4] == board[6])
        // check that the diagonal cells are not empty
        if (board[2] == 'X' || board[2] == 'O')
            return board[2]

    // no winner
    return '-'
}

/* converts the row and column coordinates in the range [0,2]
to a single index to access the 1-dimensional grid data structure
which stores the tic-tac-toe board. Cell (0, 0) is mapped to 0,
cell (0, 1) is mapped to 1 ... up to the last cell (2, 2) i.e.
the last row and last column is mapped to 8 */
fun row_col2Index(row: Int, col: Int): Int {
    return row*3 + col
}

```