5COSC023W - Tutorial 6 Exercises - Sample Solutions

1 The Memory Game

Your tutor will show and explain to you step-by-step how to develop an Android application for the problem. Make sure that you follow the same steps by using the Android Studio and at the same time that your tutor is showing them to you. ***Make sure that you understand all of the steps, do not just repeat them mechanically!***

```
package com.example.memorygamecomposable

import android.os.Bundle
import android.os.CountDownTimer
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
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.padding
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.text.style.TextAlign
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import kotlin.random.Random

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
```

```
        setContent {
            // render the screen
            GUI()
        }
    }
}


/* Which buttons have been presented to the user as green in the current game round.
   Buttons are identified with pairs of ints corresonding to (row, col) pais,
   e.g. starting with (1,1), (1,2), (1,3) for the first row, etc. */
var presented_buttons: MutableList<List<Int>> = mutableListOf()

// how many green cells will be presented to the user in the start of each game
val random_cells = 6

var number_of_clicks  = 0

var score_total = 0  // total answers given
var score_correct = 0  // total correct answers given by the user

// Returns a list with the rows and columns of the grid
fun initialise(): List<Int> {
    // 1. Decide on the number of rows and columns of the grid
    var rows = 3 + Random.nextInt(3)
    var cols =  3 + Random.nextInt(3)

    // clear presented button from the previous game
    presented_buttons.clear()

    number_of_clicks = 0

    // 2. Choose some (6) random cells to become green
    var count = 0
    while (count < random_cells) {
        var r = 1 + Random.nextInt(rows)
        val c = 1 + Random.nextInt(cols)
        val new_pair = listOf(r, c)

        // check the uniqueness of highlighted cells
        if (new_pair !in presented_buttons) {
            presented_buttons.add(new_pair)
            ++count
        }
    }

    return listOf(rows, cols)
}
```

```
// Switch off green cells by making them to their original colour again
@Composable
fun disableGreenCells(onCompletion: () -> Unit) {
    // 4. Present some cells to the user with green colour - for 3 secs
    val timer = object: CountDownTimer(3000, 1000) {
        override fun onTick(millisUntilFinished: Long) { }

        override fun onFinish() {
            onCompletion()
        }
    }
    timer.start()
}

@Preview
@Composable
fun GUI() {
    // false or true depending on whether the timer displaying some green cells
    // has run previously - it needs to be run only once per game
    var run_timer_before by remember{ mutableStateOf(false) }

    var rows by remember{ mutableStateOf(5) }
    var cols by remember{ mutableStateOf(5) }

    var selected_green: MutableList<Boolean> =
                    remember{MutableList(rows*cols) {i -> false}.toMutableStateList() }
    var selected_red: MutableList<Boolean> =
                    remember{MutableList(rows*cols) {i -> false}.toMutableStateList() }

    // true when a new game starts
    var new_game by remember{ mutableStateOf(true) }

    if (new_game) {
        // determine the grid size and the green cells to remember
        var dimensions = initialise()
        rows = dimensions[0]
        cols = dimensions[1]

        // reset the values of the cell colours in the 2 lists
        for (i in 0..selected_green.size-1) {
            selected_green[i] = false
            selected_red[i] = false
        }

        // the timer needs to run again to highlight green cells for some secs
        run_timer_before = false

        new_game = false
```

```
}

// make green the selected cells and keep them green until after
// the timer runs to completion
if (!run_timer_before) {
    for (cell in presented_buttons) {
        val r = cell[0]
        val c = cell[1]

        selected_green[(r - 1) * cols + c - 1] = true
    }

}

Column() {
    // Render the score
    Text("Score: $score_correct/$score_total", modifier = Modifier.fillMaxWidth(),
          fontSize = 32.sp, textAlign = TextAlign.End)

    // 3. Draw the grid using rows of buttons
    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(top = 100.dp),
        horizontalAlignment = Alignment.CenterHorizontally,
    ) {
        for (r in 1..rows) {
            Row() {
                for (c in 1..cols) {
                    // index of the button in a linearly 1-dimensional calculated way
                    val button_index = (r - 1) * cols + c - 1

                    var background_colour =
                        if (selected_red[button_index] == true)
                            Color.Red
                        else if (selected_green[button_index] == false)
                            Color.Gray
                        else
                            Color.Green

                    Button(
                        colors = ButtonDefaults.buttonColors(
                                    containerColor = background_colour
                        ),
                        onClick = {
                            if (number_of_clicks < presented_buttons.size) {
                                if (listOf(r, c) in presented_buttons) {
                                    selected_green[button_index] = true
                                    updateScore(1)
```

```
                                    }
                                    else {
                                        selected_red[button_index] = true
                                        updateScore(0)
                                    }
                                    ++number_of_clicks
                                } else {
                                    new_game = true
                                }
                            }) {
                            if (background_colour == Color.Red)
                                Text("X")
                        }
                    }
                }
            }
        }

        // run the count down timer only once
        if (!run_timer_before) {
            // pick up all the cells that were determined to be displayed as green
            disableGreenCells({
                // pick up all the cells that were determined to be displayed as green
                // and make them as gray
                for (coords in presented_buttons) {
                    val r = coords[0]
                    val c = coords[1]
                    selected_green[(r-1)*cols + c - 1] = false
                }
            })

            //disable the timer for the current game
            run_timer_before = true
        }
    }


fun updateScore(correct: Int) {
        ++score_total
        score_correct = score_correct + correct
}

/* Converts a pair of row (r) and column (c) arguments coordinates (1 to maxRow or maxCol)
   to a single index 0 to max_elements, based on row traversing - The function is not used
   here but it could make things easier for calculations */
fun RowCol2Index(r: Int, c: Int, cols: Int): Int {
    val index = (r-1)*cols + c - 1
```

```
    return index
}
```