5COSC023W - Tutorial 8 Exercises - Sample Solutions

1 Extending the Tic-Tac-Toe Game

File `MainActivity.kt`:

```
package com.example.tictactoe_composableapp

import android.content.Context
import android.content.SharedPreferences
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.fillMaxWidth
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.ElevatedButton
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.text.style.TextAlign
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

var human_wins = 0
var computer_wins = 0


// the computer player strategy
var computerPlayer: Player = LogicalPlayer()
//var computerPlayer: Player = Player()

class MainActivity : ComponentActivity() {
    lateinit var prefs: SharedPreferences

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        // create the shared preferences object
        prefs = getSharedPreferences("com.example.tictactoe_composableapp", MODE_PRIVATE)
        // restore the data
        human_wins = prefs.getInt("human_wins", 0)
        computer_wins = prefs.getInt("computer_wins", 0)

        setContent {
            GUI()
        }
    }

    override fun onPause() {
        super.onPause()

        // give me the editor associated with the sharedpreferences
        // object created in the onCreate() method
        var editor = prefs.edit()

        // start saving the data - in this case I just save the score
        editor.putInt("human_wins", human_wins)
        editor.putInt("computer_wins", computer_wins)

        // persist the data
        editor.apply()
    }
}

@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)) {

        // display the score
        Text(text = "Score: Human=$human_wins, Computer=$computer_wins",
            modifier = Modifier.padding(bottom = 50.dp, end = 20.dp).fillMaxWidth(),
            textAlign = TextAlign.End, fontSize = 18.sp
        )

        val context = LocalContext.current
        // display the board
        // 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') {
                                       if (winner == 'X' || winner == 'O') {
                                           if (winner == 'X')
                                               ++human_wins
                                           else if (winner == 'O')
                                               ++computer_wins
                                       }

                                       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 = computerPlayer.play(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') {
                                            if (winner == 'X')
                                                ++human_wins
                                            else if (winner == 'O')
                                                ++computer_wins

                                            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)
                    }
                }
            }
        }

        // New Game button
        ElevatedButton(modifier = Modifier.padding(top=30.dp),
            shape = RectangleShape,
            colors = ButtonDefaults.buttonColors(containerColor = Color.LightGray),
            onClick = {
                for (i in 0 until grid.size)
                    grid[i] = ' '

                gameIsPlayable = true
            })
        {
            Text(text ="New Game", color = Color.Black)
```

```
        }
    }
}

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()
}
```

File Board.kt:

```
package com.example.tictactoe_composableapp

/* 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
}
```

File Player.kt:

```
package com.example.tictactoe_composableapp

open class Player {
    // Random play startegy. Make a random move for the computer and return
    // the index of the move. If the board is full return -1
    open fun play(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()
    }
}
```

File LogicalPlayer.kt:

```
package com.example.tictactoe_composableapp

import android.widget.Toast
import androidx.compose.ui.platform.LocalContext

class LogicalPlayer(): Player() {
    /* a more intelligent player following the strategy:
    1. choose winning positions, i.e. if the computer player has already
       placed 2 'O's is a row, column or diagonal then it places the next
       'O' in the slot which completes 3 'O's to win the game.
    2. The intelligent computer player is able to defend
        itself, i.e. if the human player has already placed 2 'X's in a row,
        column or diagonal, the computer player will choose the free slot
        which will prevent the human to win in their next move.
    3.  If there  is neither a winning or defending move, the intelligent player will
        choose a valid move which creates 2 'O's is a row, column or
        diagonal.
    4. If none of the above is applicable the computer player will
       choose a random valid slot.
    */

    override fun play(board: MutableList<Char>): Int {
        var chosen_cell = get_winning_or_defend_cell(board)
        if (chosen_cell == null)  // no winning or defense slot - return a random cell
            return super.play(board)
        else
            return chosen_cell
    }


    /* assuming that the player plays always with symbol 'O':
       this will return a cell that it is either a winning move
```

```
    if there are already 2 Os in a row, column, diagonal
    or a defending move if there are already 2 Xs in a
    row, column, diagonal. Otherwise return null */
fun get_winning_or_defend_cell(board: MutableList<Char>): Int? {
    // 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

    var emptySlot: Int? = null
    var winningSlot: Int? = null
    var defendSlot: Int? = null

    // 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
            else  // empty cell
                emptySlot = index
        }

        // if a winning cell or defending cell was found note it down
        if (countO == 2 && emptySlot != null)  // winning move
            winningSlot = emptySlot
        else if (countX == 2 && emptySlot != null)
            defendSlot = emptySlot

        // reset counter values for the next row check
        countX = 0
        countO = 0
        emptySlot = null
    }

    // now check the columns
    // 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
            else  // empty cell
                emptySlot = index
        }
```

```
    // if a winning cell or defending cell was found note it down
    if (countO == 2 && emptySlot != null)  // winning move
        winningSlot = emptySlot
    else if (countX == 2 && emptySlot != null)
        defendSlot = emptySlot

    // reset counter values for the next column check
    countX = 0
    countO = 0
    emptySlot = null
}

// check top-left to bottom-right diagonal
for (i in 0..2) {
    var index = row_col2Index(i, i)
    if (board[index] == 'X')
        ++countX
    else if (board[index] == 'O')
        ++countO
    else  // empty cell
        emptySlot = index
}

// if a winning cell or defending cell was found note it down
if (countO == 2 && emptySlot != null)  // winning move
    winningSlot = emptySlot
else if (countX == 2 && emptySlot != null)
    defendSlot = emptySlot

// reset counter values for the next diagonal check
countX = 0
countO = 0
emptySlot = null

/*  check the top-right to bottom-left diagonal */
var x = 2
var y = 0
for (i in 0..2) {
    var index = row_col2Index(x - i, y + i)
    if (board[index] == 'X')
        ++countX
    else if (board[index] == 'O')
        ++countO
    else
        emptySlot = index
}

// if a winning cell or defending cell was found note it down
```

```
        if (countO == 2 && emptySlot != null)  // winning move
            winningSlot = emptySlot
        else if (countX == 2 && emptySlot != null)
            defendSlot = emptySlot

        if (winningSlot != null)
            return winningSlot
        else if (defendSlot != null)
            return defendSlot
        else
            return null
    }
}
```

File `MainActivity.kt`:

```
package com.example.tictactoe_composableapp

class RandomPlayer: Player() {
    override fun play(board: MutableList<Char>): Int {
        return super.play(board)
    }
}
```