Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського» Факультет інформатики та обчислювальної техніки Кафедра інформатики та програмної інженерії

Звіт
до лабораторної роботи № 5 з дисципліни
«Розробка мобільних застосувань під Android»

Виконала ІК-24 Юхимець Л.

Перевірив Орленко С.П.

Київ 2025

Лабораторна робота № 5

Мета роботи

змоделювати головну проблему наступного року навчання (у вигляді вибору теми дипломного проекту) шляхом самостійного формування завдання на 6 лабораторну роботу.

Завдання

Написати програму під платформу Андроїд, яка буде заснована на темі, яка не розглядалась на попередніх лабораторних роботах (попередні навички теж можна використовувати, але «основою» має бути саме нова обрана тема).

Моя реалізація

Темою для проєкту шостої лабораторної роботи я обрала гру “Хрестики нолики”

Опис функціоналу

Користувач має змогу зіграти безліч партій проти “AI” суперника

Код

```
package com.example.lab6tictactoe


import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.activity.enableEdgeToEdge
import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import
androidx.compose.foundation.gestures.detectTapGesture
s
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
```

```
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.aspectRatio
import
androidx.compose.foundation.layout.fillMaxHeight
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.width
import androidx.compose.material3.Button
import
androidx.compose.material3.CircularProgressIndicator
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.mutableStateListOf
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import
androidx.compose.runtime.rememberCoroutineScope
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.geometry.Offset
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.ColorFilter
import androidx.compose.ui.input.pointer.pointerInput
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
```

```
import androidx.compose.ui.unit.sp
import kotlinx.coroutines.delay
import kotlinx.coroutines.launch

enum class Win {
    PLAYER,
    AI,
    DRAW
}

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

@Composable
fun TTTScreen() {
    val playerTurn = remember { mutableStateOf(true) }
    val moves = remember {
        mutableStateListOf<Boolean?>(
            null,
            null,
            null,
```

```
            null,
            null,
            null,
            null,
            null,
            null
        )
    }
    val win = remember { mutableStateOf<Win?>(null) }

    val onTap: (Offset) -> Unit = {
        if (playerTurn.value && win.value == null) {
            val x = (it.x / 333).toInt()
            val y = (it.y / 333).toInt()
            val posInMoves = y * 3 + x
            if (moves[posInMoves] == null) {
                moves[posInMoves] = true
                playerTurn.value = false
                win.value = checkEndGame(moves)
            }
        }
    }

    Column(modifier = Modifier.fillMaxSize(),
horizontalAlignment = Alignment.CenterHorizontally) {
        Text(
            text = "Tic Tac Toe",
            fontSize = 30.sp,
            fontWeight = FontWeight.Bold,
```

```
            modifier = Modifier.padding(16.dp)
        )

        Header(playerTurn.value)

        Board(moves, onTap)

        if (!playerTurn.value && win.value == null) {
            CircularProgressIndicator(color =
Color.Red, modifier = Modifier.padding(16.dp))

            val coroutineScope =
rememberCoroutineScope()
            LaunchedEffect(key1 = Unit) {
                coroutineScope.launch {
                    delay(1000L)
                    val aiMove = findBestMove(moves)
                    moves[aiMove] = false
                    playerTurn.value = true
                    win.value = checkEndGame(moves)
                }
            }
        }

        if (win.value != null) {
            when (win.value) {
                Win.PLAYER -> Text(text = "Player has
won \uD83C\uDF89", fontSize = 25.sp)
                Win.AI -> Text(text = "AI has won
\uD83D\uDE24", fontSize = 25.sp)
```

```
                Win.DRAW -> Text(text = "It`s a draw
\uD83D\uDE33", fontSize = 25.sp)
                else -> {}
            }

            Button(onClick = {
                playerTurn.value = true
                win.value = null
                for (i in 0..8) moves[i] = null
            }) {
                Text(text = "Restart the game")
            }
        }
    }
}

fun checkEndGame(m: List<Boolean?>): Win? {
    val winPatterns = listOf(
        listOf(0, 1, 2), listOf(3, 4, 5), listOf(6, 7,
8),
        listOf(0, 3, 6), listOf(1, 4, 7), listOf(2, 5,
8),
        listOf(0, 4, 8), listOf(2, 4, 6)
    )

    winPatterns.forEach { pattern ->
        if (pattern.all { m[it] == true }) return
Win.PLAYER
        if (pattern.all { m[it] == false }) return
Win.AI
    }
```

```
    if (m.none { it == null }) return Win.DRAW
    return null
}

fun findBestMove(moves: MutableList<Boolean?>): Int {
    var bestScore = Int.MIN_VALUE
    var bestMove = -1

    for (i in moves.indices) {
        if (moves[i] == null) {
            moves[i] = false
            val score = minimax(moves, 0, false)
            moves[i] = null
            if (score > bestScore) {
                bestScore = score
                bestMove = i
            }
        }
    }

    return bestMove
}

fun minimax(board: MutableList<Boolean?>, depth: Int,
isMaximizing: Boolean): Int {
    val result = checkEndGame(board)
    if (result != null) {
        return when (result) {
```

```
            Win.PLAYER -> -10 + depth
            Win.AI -> 10 - depth
            Win.DRAW -> 0
        }
    }

    if (isMaximizing) {
        var bestScore = Int.MIN_VALUE
        for (i in board.indices) {
            if (board[i] == null) {
                board[i] = false
                val score = minimax(board, depth + 1,
false)
                board[i] = null
                bestScore = maxOf(score, bestScore)
            }
        }
        return bestScore
    } else {
        var bestScore = Int.MAX_VALUE
        for (i in board.indices) {
            if (board[i] == null) {
                board[i] = true
                val score = minimax(board, depth + 1,
true)
                board[i] = null
                bestScore = minOf(score, bestScore)
            }
        }
        return bestScore
```

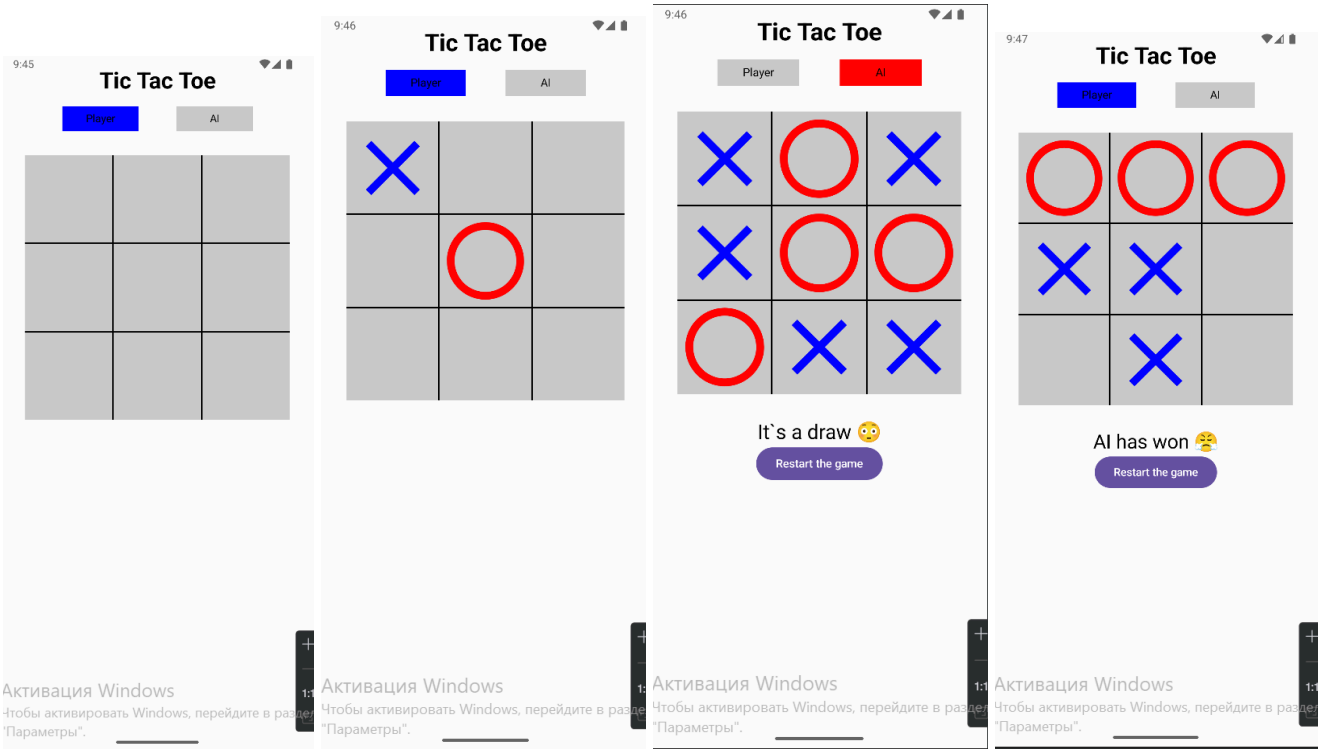
```
    }
}

@Composable
fun Header(playerTurn: Boolean) {
    Row(
        verticalAlignment =
Alignment.CenterVertically,
        horizontalArrangement =
Arrangement.SpaceAround
    ) {
        @Composable
        fun PlayerBox(text: String, color: Color) {
            Box(
                modifier = Modifier
                    .width(100.dp)
                    .background(color)
            ) {
                Text(
                    text = text, modifier = Modifier
                        .padding(8.dp)
                        .align(Alignment.Center)
                )
            }
        }

        PlayerBox("Player", if (playerTurn) Color.Blue
else Color.LightGray)
        Spacer(modifier = Modifier.width(50.dp))
        PlayerBox("AI", if (playerTurn)
```

```
Color.LightGray else Color.Red)
    }
}


@Composable
fun Board(moves: List<Boolean?>, onTap: (Offset) ->
Unit) {
    Box(
        modifier = Modifier
            .aspectRatio(1f)
            .padding(32.dp)
            .background(Color.LightGray)
            .pointerInput(Unit) {
                detectTapGestures(onTap = onTap)
            }
    ) {
        Column(verticalArrangement =
Arrangement.SpaceEvenly, modifier =
Modifier.fillMaxSize(1f)) {
            repeat(2) {
                Row(
                    modifier = Modifier
                        .height(2.dp)
                        .fillMaxWidth(1f)
                        .background(Color.Black)
                ) {}
            }
        }
        Row(horizontalArrangement =
Arrangement.SpaceEvenly, modifier =
Modifier.fillMaxSize(1f)) {
```

```
            repeat(2) {
                Column(
                    modifier = Modifier
                        .width(2.dp)
                        .fillMaxHeight(1f)
                        .background(Color.Black)
                ) {}
            }
        }
        Column(modifier = Modifier.fillMaxSize(1f)) {
            for (i in 0..2) {
                Row(modifier = Modifier.weight(1f)) {
                    for (j in 0..2) {
                        Column(modifier =
Modifier.weight(1f)) {
                            GetComposableFromMove(move
= moves[i * 3 + j])
                        }
                    }
                }
            }
        }
    }
}

@Composable
fun GetComposableFromMove(move: Boolean?) {
    when (move) {
        true -> Image(
            painter = painterResource(id =
```

```
R.drawable.x),
                contentDescription = null,
                modifier = Modifier.fillMaxSize(1f),
                colorFilter = ColorFilter.tint(Color.Blue)
            )

            false -> Image(
                painter = painterResource(id =
R.drawable.o),
                contentDescription = null,
                modifier = Modifier.fillMaxSize(1f),
                colorFilter = ColorFilter.tint(Color.Red)
            )

            null -> Image(
                painter = painterResource(id =
R.drawable.null1),
                contentDescription = null,
                modifier = Modifier.fillMaxSize(1f)
            )
        }
}
```

Вигляд програми



Перемогти так і не вийшло(