Instituto Superior de Engenharia
Politécnico de Coimbra

Instituto Politécnico de Coimbra

Instituto Superior de Engenharia de Coimbra

Arquiteturas Móveis

Trabalho Prático 1 – Android/Kotlin

Docente: Álvaro Santos

Marco António de Jesus Coelho – 2018012765 – LEI

Pedro Jorge Fernandes Morais – 2018020733 – LEI

segunda-feira, 9 de janeiro de 2023

Índice

Índice de Figuras

1 Variações de níveis

No decorrer da evolução do jogo os parâmetros dos níveis vão evoluindo, sendo que até ao nível 3 só usamos números de 1 a 9, depois até ao nível 7 utilizamos números de 1 a 99 e depois usamos de 1 a 999.

Nas equações os operadores também se vão alterando, sendo que no primeiro nível é só equações de soma, no segundo soma e subtração, no terceiro soma, subtração e multiplicação e no quarto e em diante soma, subtração, multiplicação e divisão.

Cada nível tem um total de cinco tabuleiros de jogo.

Os tempos que o jogador tem para acabar cada nível também vão diminuindo, sendo que começa em 60 segundos e diminui 10 segundos a cada nível que passa, até um mínimo de 10 segundos por nível.

O jogador por cada resposta certa que der recebe 5 segundos extra, mas se errar uma resposta perde também 5 segundos.

Todas estas configurações estão no ficheiro de constantes (Constants.kt).

```kotlin
package pt.isec.amov.a2018020733.trabalhopratico1.models

const val COLLECTION_PATH = "TopScoresSingleplayer"
const val COLLECTION_FIELD_POINTS = "points"
const val COLLECTION_FIELD_TIME_PLAYED = "timePlayed"

const val DIRECTION_HORIZONTAL = 1
const val DIRECTION_VERTICAL = 2

const val SIZE_GAME_BOARD = 5
const val NUMBER_EQUATIONS_LEVEL = 5

const val TRANSITION_TIME_SECONDS = 5

const val STARTING_TIME_SECONDS = 60
const val DECREASE_TIME_PER_LEVEL = 10
const val MINIMUM_TIME_LEVEL = 10
const val ADDED_TIME_CORRECT_ANSWER = 5
const val REMOVED_TIME_WRONG_ANSWER = 5

val MAX_NUMBER_USED_IN_EQUATIONS_START = arrayOf(9,99,999)

val OPERATORS_USED_PER_LEVEL = arrayOf(arrayOf("+"), arrayOf("+","-"),
    arrayOf("+","-","x"), arrayOf("+","-","x","/"))
```

Figura 1 - Constants

2 GameBoard

Cada nível é constituído por vários tabuleiros de jogo, sendo que cada um tem duas equações que dão pontos ao utilizador. Os números e operadores de cada tabuleiro são aleatórios tendo em conta o nível atual. Cada tabuleiro guarda ainda o valor e posição da sua maior e segunda maior equação.
Está ainda definido na classe GameBoard duas funções estáticas, através do *companion object*, que permitem calcular o valor de uma dada equação. Para isto recebe a equação num Array de Strings, em que nas posições pares tem os valores numéricos e nas ímpares os operadores. O cálculo da equação é feito da esquerda para a direita, sendo que tem em conta as prioridades dos operadores, ou seja, a multiplicação e divisão é realizada antes da soma e subtração.

A função *operatorFromString()* permite realizar uma operação matemática enviando como argumento um operador.

```
private fun operatorFromString(stringOperator: String): (Double, Double) -> Double {
    return when (stringOperator) {
        "+" -> { a, b -> a + b }
        "-" -> { a, b -> a - b }
        "/" -> { a, b -> a / b }
        "x" -> { a, b -> a * b }
        else -> throw Exception("That's not a supported operator")
    }
}
```

Figura 2 - operatorFromString()

A função *calculateValueOperation()* recebe um *array* de *Strings* e realiza o cálculo da equação que é enviado no mesmo. Devolve o resultado da equação.

```
fun calculateValueOperation(operation: Array<String>): Double {

    var result: Double
    val mathPriority = ArrayList<String>()
    var flagSkip = false

    for (i in operation.indices) {
        if (operation[i] == "x" || operation[i] == "/") {
            mathPriority[mathPriority.size - 1] = operatorFromString(operation[i])
                .invoke(
                    mathPriority[mathPriority.size - 1].toDouble(),
                    operation[i + 1].toDouble()
                ).toString()
            flagSkip = true
        } else {
            if (!flagSkip)
                mathPriority.add(operation[i])
            flagSkip = false
        }
    }

    result = mathPriority[0].toDouble()
    for (i in mathPriority.indices)
        if (mathPriority[i] == "+" || mathPriority[i] == "-")
            result = operatorFromString(mathPriority[i])
                .invoke(
                    result,
                    mathPriority[i + 1].toDouble()
                )

    return result
}
```

Figura 3 - calculateValueOperation()

3 Game

Para guardarmos os vários tabuleiros de jogo que vamos tendo, utilizamos a classe *Game*, sendo que esta estende da classe *ViewModel*, de forma a que quando a atividade seja recarregada, por exemplo a alterar a orientação do ecrã do dispositivo, não sejam perdidos os dados do jogo.
A classe *Game* guarda para além de um *array* com todos os tabuleiros de jogo para todos os níveis até agora decorridos, a pontuação do jogador, o tempo de jogo do jogador, quantos segundos faltam até o jogador perder se não completar o nível atual e qual o nível e equação em que o jogador se encontra atualmente.

Na inicialização da classe são criados os tabuleiros de jogo para o primeiro nível.

```
class Game : ViewModel() {

    private var gameLevels: ArrayList<List<GameBoard>> = ArrayList()
    private var currentLevel: Int = 1
    private var currentEquation: Int = 0
    private var points: Int = 0
    private var timePlayedSeconds: Int = 0
    private var timeLeftLevel: Int = STARTING_TIME_SECONDS

    init {

        val equations: ArrayList<GameBoard> = ArrayList()

        for (i in 1 ≤ .. ≤ NUMBER_EQUATIONS_LEVEL)
            equations.add(GameBoard(currentLevel))

        gameLevels.add(equations)
    }
```

Figura 4 - Game

4 Singleplayer

4.1 Seleção da equação

Para o utilizador selecionar uma equação é utilizado o gesto Fling, sendo que para isto a atividade *SingleplayerActivity* estende da classe *GestureDetector.OnGestureListener*. Quando um gesto *onFling* é detetado, são verificadas as coordenadas, sendo que o utilizador apenas precisa de realizar o gesto na linha ou coluna pretendida, sem ter a necessidade de começar num ponto específico e terminar num ponto específico.

```
val direction =
    when (abs( x: e1.x - e2.x) > abs( x: e1.y - e2.y)) {
        true -> DIRECTION_HORIZONTAL
        false -> DIRECTION_VERTICAL
    }

val viewBoundsPrimeiraLinhaInicio = Rect()
findViewById<TextView>(R.id.cell00).getGlobalVisibleRect(viewBoundsPrimeiraLinhaInicio)

if (
    (viewBoundsPrimeiraLinhaInicio.contains(
        viewBoundsPrimeiraLinhaInicio.centerX(), e1.y.toInt()
    ) && direction == DIRECTION_HORIZONTAL)
    ||
    (viewBoundsPrimeiraLinhaInicio.contains(
        e1.x.toInt(), viewBoundsPrimeiraLinhaInicio.centerY()
    ) && direction == DIRECTION_VERTICAL)
)
    verificaConta(direction,  indice: 0)
```

Figura 5 - Verificação seleção, Primeira linha e coluna

4.2 Tempo restante

Para efetuar a contagem decrescente do tempo restante, e respetiva contagem crescente do tempo a que o utilizador está a jogar é utilizado a classe *Timer*.

Para isto utilizamos um *fixedRateTimer* que de um em um segundo decrementa o tempo restante e aumenta o tempo de jogo. Verifica ainda o término do jogo quando o tempo restante chega a zero.

```
private fun startTimeLeft() {
    timer = fixedRateTimer( name: "timeLeftCounter", daemon: false, initialDelay: 0L, period: 1 * 1000) { this: TimerTask
        this@SingleplayerActivity.runOnUiThread {
            tempoBack--
            game.decrementTimeLeft()
            game.incrementTimePlayed()

            binding.tvTimeLeft.text = game.getTimeLeftLevel().toString()
            binding.tvTimePlayed.text = game.getTimePlayed().toString()
        }

        //Parar a cena
        if (game.getTimeLeftLevel() <= 0) {
            acabarJogo()
            cancel()
        }
    }
}
```

Figura 6 - Timer tempo restante de jogo

4.3 Sair do jogo

Para o utilizador poder sair do jogo terá de carregar no botão *back* duas vezes seguidas num espaço de tempo de cinco segundos. O tempo e o segundo clique servem de verificação para confirmar que o utilizador pretende mesmo deixar o jogo atual.

```
override fun onBackPressed() {
    if (tempoBack <= 0) {
        tempoBack = 5
        Toast.makeText(
            applicationContext,
            "Press back again to exit",
            Toast.LENGTH_SHORT
        ).show()
    } else
        acabarJogo()
}
```

Figura 7 - Confirmação sair jogo atual

5 Login

Ao realizarmos o login na aplicação podemos fazê-lo de duas formas distintas:

- Através de um email e password;
- Através da conta Google.

Para realizarmos o login com o email e password, temos de antes utilizar o registo da aplicação e criarmos uma conta de utilizador. Já com a conta Google é um processo muito mais simples.

5.1 Email/Password

Para realizarmos o login com email e password utilizamos a autenticação da Firebase, tendo de enviar estes dados para a Firebase confirmar a sua autenticidade.

```
fun signInWithEmail() {

    if (strEmail.isBlank() || strPass.isBlank())
        return

    auth.signInWithEmailAndPassword(strEmail, strPass)
        .addOnSuccessListener(this) {  it: AuthResult!
            showUser(auth.currentUser)
            val intent = Intent( packageContext: this, MainActivity::class.java)
            startActivity(intent)
        }
        .addOnFailureListener(this) {  it: Exception
            showUser( user: null)
        }
}
```

Figura 8 - Autenticação Email/Password

5.2 Google

Para realizar a autenticação com a conta Google, do ponto de vista do utilizador basta selecionar a conta Google associada ao seu dispositivo, ou associar uma nova. Já no programa temos de utilizar o GoogleSignInClient para verificar a autenticação do mesmo, existindo ainda uma integração posterior com a Firebase para efetuar a autenticação na aplicação.

```
val signInWithGoogle = registerForActivityResult(
    ActivityResultContracts.StartActivityForResult()
) { result ->
    val task = GoogleSignIn.getSignedInAccountFromIntent(result.data)
    try {
        val account = task.getResult(ApiException::class.java)!!
        firebaseAuthWithGoogle(account.idToken!!)
    } catch (_: ApiException) {
    }
}

private fun firebaseAuthWithGoogle(idToken: String) {
    val credential = GoogleAuthProvider.getCredential(idToken,  accessToken: null)
    auth.signInWithCredential(credential)
        .addOnSuccessListener(this) {  it: AuthResult!
            showUser(auth.currentUser)
            val intent = Intent( packageContext: this, MainActivity::class.java)
            startActivity(intent)
        }
        .addOnFailureListener(this) {  it: Exception
            showUser(auth.currentUser)
        }
}
```

Figura 9 - Autenticação Google

6 Firebase

Para o desenvolvimento de algumas funcionalidades da nossa aplicação foi necessário a utilização da Firebase, nomeadamente para efetuar a autenticação de utilizadores, guardar as imagens de perfil dos utilizadores e armazenar as pontuações dos utilizadores.

6.1 Autenticação

Como referido no capítulo do Login utilizamos a Firebase para efetuar a autenticação de utilizadores na nossa aplicação.

6.2 Storage

De forma a guardarmos as imagens de perfil de cada utilizador utilizamos a Storage, em que temos uma pasta "users" que contém as imagens de perfil, sendo que estas são identificadas pelo seu nome que é igual ao email do utilizador a quem elas pertencem.

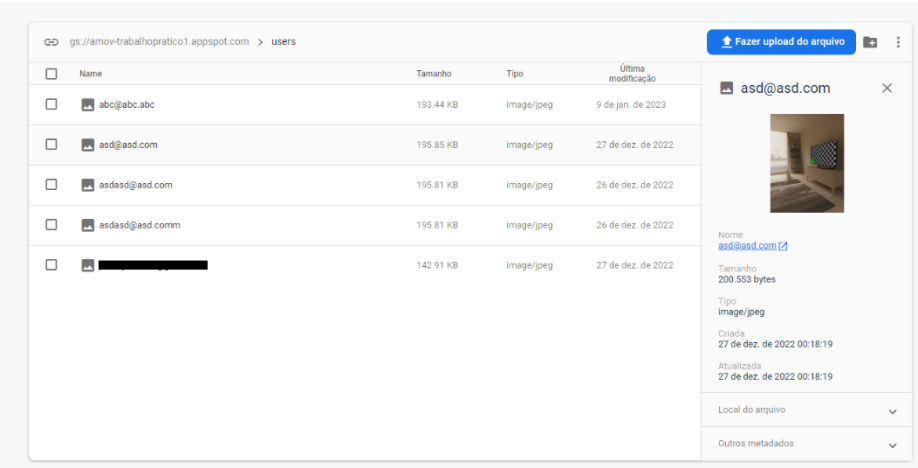

Figura 10 - Firebase Storage

```
fun updateImageFirebase() {
    FirebaseAuth.getInstance().signInAnonymously()

    val imageUri = FileProvider.getUriForFile(
        context: this,
        authority: "pt.isec.amov.a2018020733.trabalhopratico1.android.fileprovider",
        imagePath?.let { File(it) }!!
    )

    val ref = FirebaseStorage.getInstance().getReference( location: "/users/$strEmail")

    ref.putFile(imageUri!!).addOnSuccessListener {  it: UploadTask.TaskSnapshot!
        Log.i(
            tag: "PerfilActivity",
            msg: "Colocada com sucesso na FireBaseStorage a imagem: ${it.metadata?.path}"
        )

        ref.downloadUrl.addOnSuccessListener { url ->
            Log.i( tag: "PerfilActivity",  msg: "Localizaçao da imagem: $url")

            val db = Firebase.firestore
            val utilizador = db.collection( collectionPath: "Utilizadores").document(strEmail)

            utilizador.get(Source.SERVER)
                .addOnSuccessListener {  it: DocumentSnapshot!
                    utilizador.update( field: "urlImagem", url.toString())
                }
        }
    }
}
```

Figura 11 - Atualização da imagem de perfil na Firebase

6.3 Firestore Database

Para guardarmos as pontuações dos utilizadores utilizamos a *database* da *firestore*, sendo que esta permite ter coleções em que cada coleção tem vários documentos e cada documento tem vários campos. Utilizamos uma coleção chamada “TopScoresSingleplayer” onde temos vários documentos, sendo eles identificados pelo email do utilizador que representam. Cada um destes documentos tem dois campos, a maior pontuação do utilizador em questão e o tempo que este demorou a atingi-la.

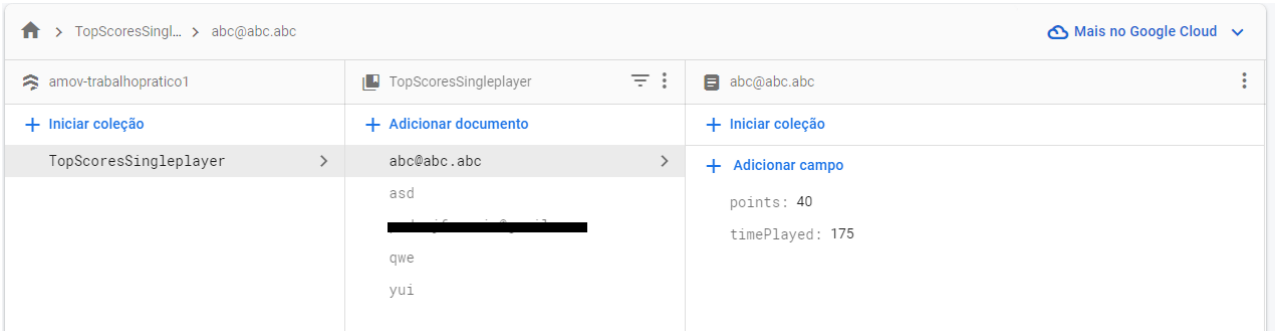

Figura 12 - Firestore Database

Caso este utilizador ainda não tenha pontuação guardada, é criado um novo documento para ele.

```
val db = Firebase.firestore
val v = auth.currentUser?.email?.let { db.collection(COLLECTION_PATH).document(it) }
db.runTransaction { transaction ->
    val doc = v?.let { transaction.get(it) }
    if (doc?.exists() == true) {
        val storedPoints = (doc.getLong(COLLECTION_FIELD_POINTS) ?: 0)

        if (game.getPoints() > storedPoints) {
            transaction.update(v, COLLECTION_FIELD_POINTS, game.getPoints())
            transaction.update(v, COLLECTION_FIELD_TIME_PLAYED, game.getTimePlayed())
        }
        null  ^runTransaction
    } else {
        val scores = hashMapOf(
            COLLECTION_FIELD_POINTS to game.getPoints(),
            COLLECTION_FIELD_TIME_PLAYED to game.getTimePlayed()
        )
        auth.currentUser?.email?.let {  it: String
            db.collection(COLLECTION_PATH).document(it).set(scores)
                .addOnSuccessListener {  it: Void!
                    Log.i( tag: "TAG",  msg: "addDataToFirestore: Success")
                }
                .addOnFailureListener { e ->
                    Log.i( tag: "TAG",  msg: "addDataToFirestore: ${e.message}")
                }
        }  ^runTransaction
    }
}.addOnSuccessListener {  it: Task<Void!>!
    Log.i( tag: "TAG",  msg: "updateDataInFirestoreTrans: Success")
}.addOnFailureListener { e ->
    Log.i( tag: "TAG",  msg: "updateDataInFirestoreTrans: ${e.message}")
}
```

Figura 13 - Criação ou atualização da pontuação na Firestore Database

7 Idiomas

No decorrer do desenvolvimento da aplicação foi sempre tido em atenção a utilização de dois idiomas, nomeadamente o Português e o Inglês. Para isto foram criados dois ficheiros *xml* de *Strings*, que contêm a tradução para cada um destes idiomas das várias frases utilizadas na interface do programa.

Key	Resource Folder	Untranslatable	Default Value	Portuguese (pt)
app_name	app/src/main/res	☐	Math Champion	Campeão das Matemáticas
sua_palavra_passe	app/src/main/res	☐	Your password	Sua palavra passe
password	app/src/main/res	☐	Password	Palavra Chave
logotipo_jogo	app/src/main/res	☐	Game Logo	Logotipo Jogo
insira_o_seu_email	app/src/main/res	☐	Insert your email	Insira o seu email
entrar_com_o_google	app/src/main/res	☐	Login with Google	Entrar com o Google
entrar	app/src/main/res	☐	Login	Entrar
email	app/src/main/res	☐	Email	Email
criar_conta	app/src/main/res	☐	Create Account	Criar Conta
confirmar_password	app/src/main/res	☐	Confirm Password	Confirmar Palavra Chave
take_photo	app/src/main/res	☐	Change Photo	Alterar Foto
errorDiferentPasswords	app/src/main/res	☐	Password dont match!	As palavras chaves são diferentes
errorCreatingUser	app/src/main/res	☐	Error creating user	Erro a registar o utilizador
userRegisteredSuccessfully	app/src/main/res	☐	User registered successfully	Utilizador registado com sucesso
passwordLength	app/src/main/res	☐	Password length should be 6 or mo	A password tem de ter 6 caracteres
errorLogin	app/src/main/res	☐	Email or password incorrect	Email ou palavra chave incorretos
edit_account	app/src/main/res	☐	Edit Account	Editar dados
logout	app/src/main/res	☐	Logout	Sair
logout_successful	app/src/main/res	☐	Goodbye	Adeus
error_logging_out	app/src/main/res	☐	Error logging out	Erro a fazer logout
edit_user_title	app/src/main/res	☐	Edit User	Atualizar Utilizador
error_edit_user	app/src/main/res	☐	Error updating user	Erro a atualizar utilizador
user_update_successful	app/src/main/res	☐	User updated successfully	Utilizador atualizado
edit_user_profile	app/src/main/res	☐	Edit profile	Editar perfil
language	app/src/main/res	☐	Language	Idioma
welcome	app/src/main/res	☐	Welcome	Olá
novo_jogo_singleplayer	app/src/main/res	☐	New Game - Singleplayer	Novo Jogo – 1 Jogador
singleplayerTitle	app/src/main/res	☐	Singleplayer	1 Jogador
points	app/src/main/res	☐	Points:	Pontuação:
time_left	app/src/main/res	☐	Time left:	Tempo restante:
previous	app/src/main/res	☐	Previous	Anterior
next	app/src/main/res	☐	Next	Próximo
level	app/src/main/res	☐	Level:	Nível:
equation	app/src/main/res	☐	Equation:	Equação:
finishLevel	app/src/main/res	☐	Finish Level	Terminar nível
time_played	app/src/main/res	☐	Time Played:	Tempo Jogado:
nextLevel	app/src/main/res	☐	Next Level	Próximo Nível
paused	app/src/main/res	☐	Paused	Pausado
top_5_players	app/src/main/res	☐	Top 5 players	Top 5 jogadores
top5_1_player	app/src/main/res	☐	Top 5 - 1 Player	Top 5 – 1 Jogador
user	app/src/main/res	☐	User:	Utilizador:
authors	app/src/main/res	☐	Marco Coelho - 2018012765[...]	Marco Coelho – 2018012765[...]
subject	app/src/main/res	☐	Mobile Architectures[...]	Arquiteturas Móveis[...]
logotipo_isec	app/src/main/res	☐	Logotipo ISEC	Logotipo ISEC
credits	app/src/main/res	☐	Credits	Créditos
pressBackExit	app/src/main/res	☐	Press back again to exit	Volte a carregar para sair

Figura 14 - Traduções

8 Orientações ecrã

De forma a melhorar a apresentação visual da nossa aplicação em certas atividades foi pensada a possibilidade do utilizador poder ter o dispositivo em diferentes orientações. Como tal, foi criada uma interface especifica para o jogo para quando o utilizador tem o dispositivo em modo *Portrait* ou em modo *Landscape*.

Em certas atividades em que isto não é permitido foi bloqueada a orientação a uma especifica, neste caso à orientação Portrait. Estas atividades serão a de Login, Registo, Edição da conta de utilizador, Top 5 utilizadores com mais pontos e os Créditos da aplicação.

```
<activity
    android:name=".LoginActivity"
    android:screenOrientation="portrait"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>

    <meta-data
        android:name="android.app.lib_name"
        android:value="" />
</activity>
```

Figura 15 - Orientação ecrã, Bloqueio orientação do Login

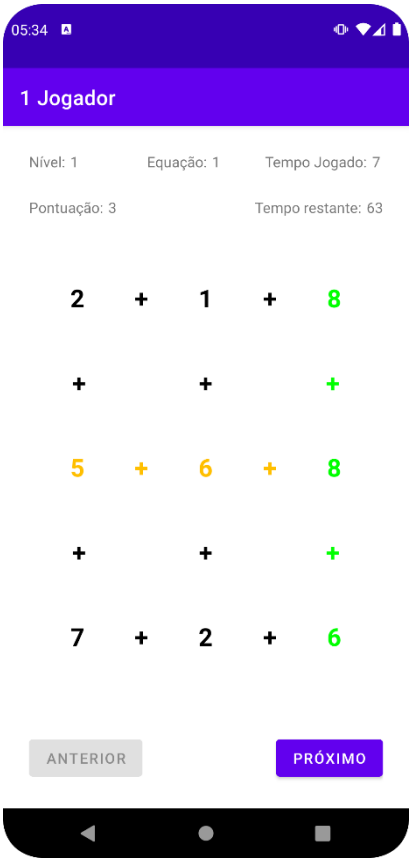

Figura 16 - Orientação de ecrã, Modo Portrait

Figura 17 - Orientação de ecrã, Modo Landscape