RELATÓRIO

Trabalho Prático - Kotlin

Arquiteturas Móveis

Licenciatura em Engenharia Informática

Desenvolvimento de um jogo para a plataforma Android capaz de ajudar a desenvolver a agilidade no cálculo matemático dos jogadores.

Trabalho realizado por:

Tomás Gomes Silva - 2020143845

Rafael Gerardo Couto - 2019142454

Tânia Beatriz Moreira Guedes - 2020139445

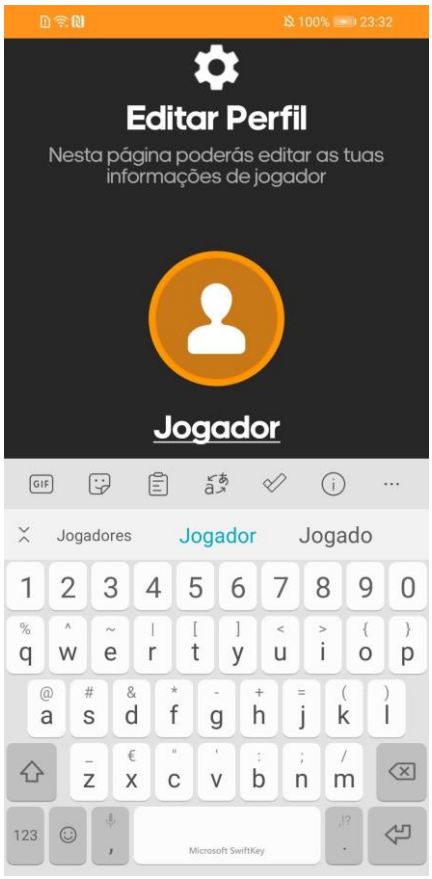
Índice

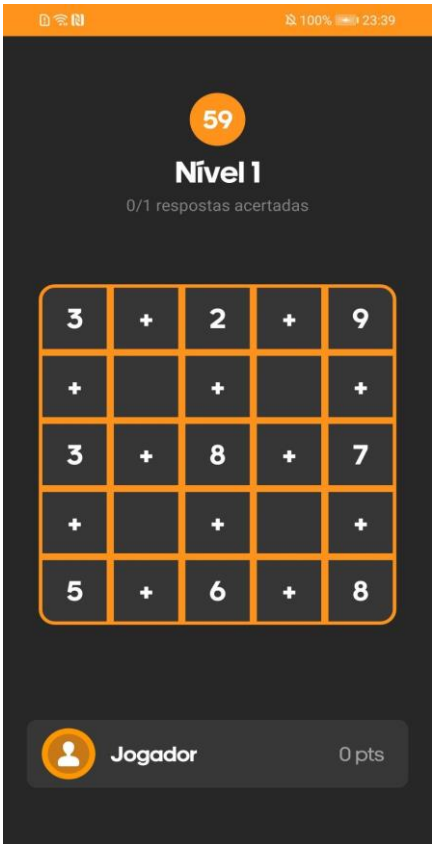
Introdução

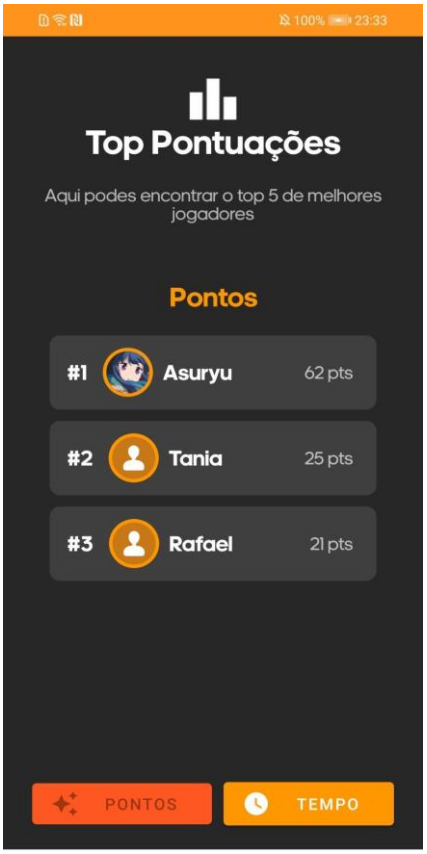
O trabalho prático de Arquiteturas Móveis consiste na criação de uma aplicação, em Kotlin, que suporta um jogo de agilidade de cálculo matemático por parte dos jogadores. Este é pontuado por cálculos acertados e é passado de nível em nível até ao término do tempo estipulado.
O jogo está dividido em dois modos, sendo eles o modo único jogador e modo multijogador. Em ambos os casos, as pontuações alcançadas são guardadas em Firebase e posteriormente representadas num Top 5.
É permitido ao utilizador alterar os seus dados de jogo como nome e avatar.

Funcionalidades

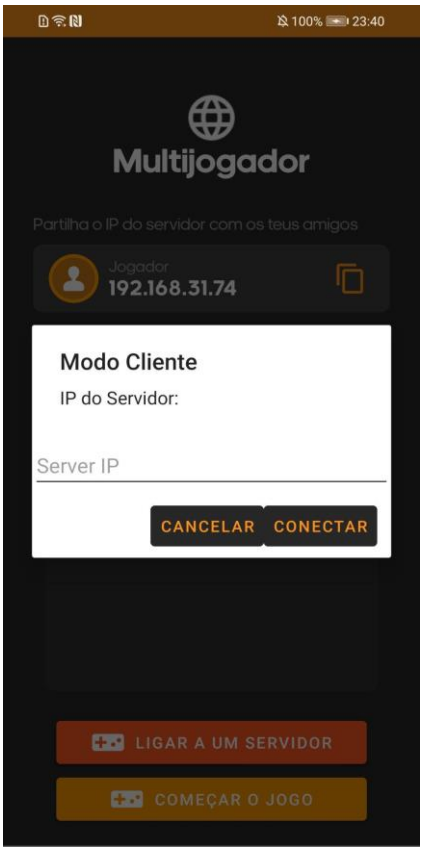
Interação com o utilizador








100% 23:40
Multijogador
Partilha o IP do servidor com os teus amigos
Jogador
192.168.31.74
LIGAR A UM SERVIDOR
COMEÇAR O JOGO


100% 23:40
Multijogador
Partilha o IP do servidor com os teus amigos
Jogador
192.168.31.74
Modo Cliente
IP do Servidor:
Server IP
CANCELAR
CONECTAR
LIGAR A UM SERVIDOR
COMEÇAR O JOGO


100% 23:40
Jogo terminado!
Nível alcançado: 1
Pontos ganhos: 0
TENTAR NOVAMENTE
MENU PRINCIPAL

Modo 1 Jogador

O modo de um jogador permite que o jogador jogue apenas contra o relógio sendo que para ganhar pontos necessita de acertar as expressões com maior valor. O jogo acaba quando o tempo acabar.

Para gerir o jogo foi criada uma classe Game que gere o jogo de um jogador. Esta classe contém várias propriedades e métodos para gerar o tabuleiro, verificar as expressões selecionadas, etc…

Algo que não foi implementado por opção foi a espera de 5 segundos entre níveis visto que concordámos que isso afetava a imersividade do jogo.

```kotlin
fun getTimerObject (time: Long) : CountDownTimer {
    return object : CountDownTimer(time, countDownInterval: 1000) {
        override fun onTick(millisUntilFinished: Long) {
            binding.timer.text = (millisUntilFinished / 1000).toString()
            timeLeft = millisUntilFinished
            totalGameTime += 1000
        }

        override fun onFinish() {
            binding.timer.text = "0"
            timeLeft = 0
            context.showEndGameScreen()
        }
    }
}
```

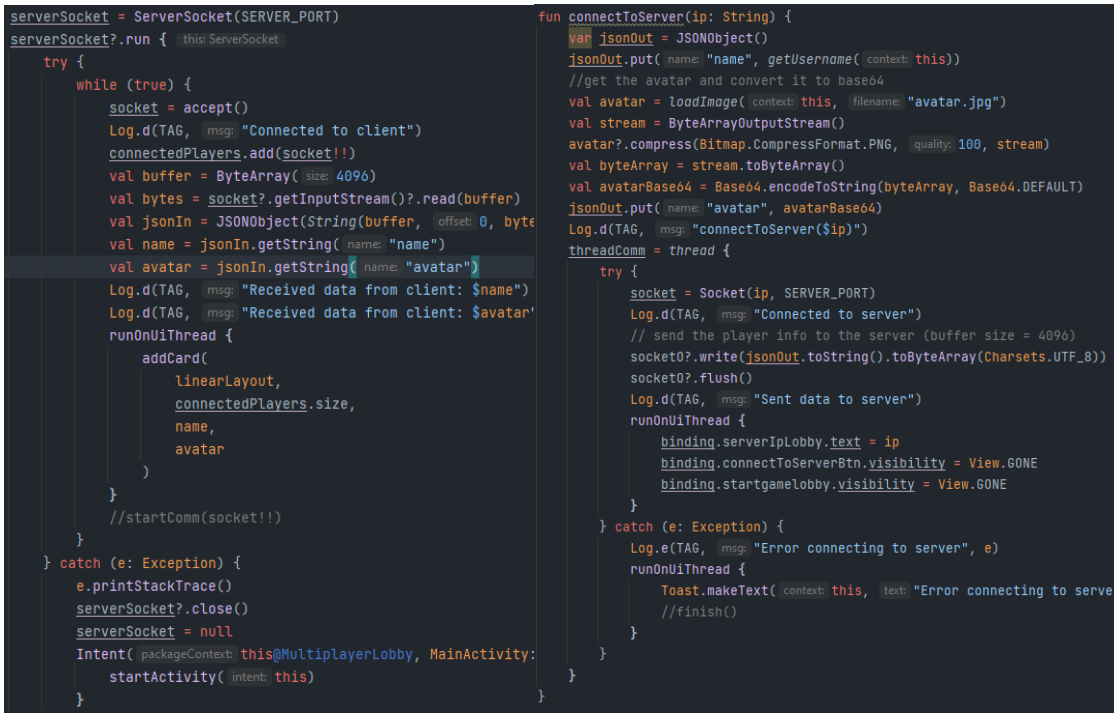
```kotlin
fun generateBoard(){
    board.clear()
    expressions.clear()

    for(i in 0 ≤ .. ≤ 4){
        var row : ArrayList<String> = ArrayList()
        for(j in 0 ≤ .. ≤ 4){
            if(i % 2 == 0){
                if(j % 2 == 0){
                    row.add(getRandomNumber().toString(
                } else {
                    row.add(getRandomOperator())
                }
            } else {
                if(j % 2 == 0){
                    row.add(getRandomOperator())
                } else {
                    row.add(" ")
                }
            }
        }
        board.add(row)
    }

    updateBoard()

    parseBoard()
    Log.i( tag: "Asuryu", expressions.toString())
}
```

```kotlin
fun evaluateExpression(expression : String) : Int{
    var result = 0
    var operator = '+'
    var number = ""
    for(i in 0 ≤ until < expression.length){
        if(expression[i] == '+' || expression[i] == '-' || e
            when(operator){
                '+' -> result += number.toInt()
                '-' -> result -= number.toInt()
                '*' -> result *= number.toInt()
                '/' -> result /= number.toInt()
            }
            operator = expression[i]
            number = ""
        } else {
            number += expression[i]
        }
    }
    when(operator){
        '+' -> result += number.toInt()
        '-' -> result -= number.toInt()
        '*' -> result *= number.toInt()
        '/' -> result /= number.toInt()
    }
    return result
}
```

Para além das funções em cima apresentadas, foram desenvolvidas funções para dar *parse* ao tabuleiro (extrair todas as expressões existentes) e atualizar o tabuleiro na UI.

Modo Multijogador

O modo de multijogador ficou praticamente por implementar. A única parte desta funcionalidade que ficou a funcionar foi o lobby onde os vários jogadores entram. O jogador-servidor aceita as ligações dos vários jogadores-cliente e adiciona uma "carta" com o nome do jogador à ScrollView.
Para o jogador se conectar ao servidor basta introduzir o IP no input do AlertDialog e é efetuada uma ligação TCP através de sockets.



```
serverSocket = ServerSocket(SERVER_PORT)
serverSocket?.run {
    try {
        while (true) {
            socket = accept()
            Log.d(TAG, "Connected to client")
            connectedPlayers.add(socket!!)
            val buffer = ByteArray(4096)
            val bytes = socket?.getInputStream()?.read(buffer)
            val jsonIn = JSONObject(String(buffer, 0, byte
            val name = jsonIn.getString("name")
            val avatar = jsonIn.getString("avatar")
            Log.d(TAG, "Received data from client: $name")
            Log.d(TAG, "Received data from client: $avatar
            runOnUiThread {
                addCard(
                    linearLayout,
                    connectedPlayers.size,
                    name,
                    avatar
                )
            }
            //startComm(socket!!)
        }
    } catch (e: Exception) {
        e.printStackTrace()
        serverSocket?.close()
        serverSocket = null
        Intent(this@MultiplayerLobby, MainActivity:
            startActivity(this)
        }
```

```
fun connectToServer(ip: String) {
    var jsonOut = JSONObject()
    jsonOut.put("name", getUsername(this))
    //get the avatar and convert it to base64
    val avatar = loadImage(this, "avatar.jpg")
    val stream = ByteArrayOutputStream()
    avatar?.compress(Bitmap.CompressFormat.PNG, 100, stream)
    val byteArray = stream.toByteArray()
    val avatarBase64 = Base64.encodeToString(byteArray, Base64.DEFAULT)
    jsonOut.put("avatar", avatarBase64)
    Log.d(TAG, "connectToServer($ip)")
    threadComm = thread {
        try {
            socket = Socket(ip, SERVER_PORT)
            Log.d(TAG, "Connected to server")
            // send the player info to the server (buffer size = 4096)
            socket0?.write(jsonOut.toString().toByteArray(Charsets.UTF_8))
            socket0?.flush()
            Log.d(TAG, "Sent data to server")
            runOnUiThread {
                binding.serverIpLobby.text = ip
                binding.connectToServerBtn.visibility = View.GONE
                binding.startgamelobby.visibility = View.GONE
            }
        } catch (e: Exception) {
            Log.e(TAG, "Error connecting to server", e)
            runOnUiThread {
                Toast.makeText(this, "Error connecting to serve
                //finish()
            }
        }
    }
}
```

Definição do perfil de jogador

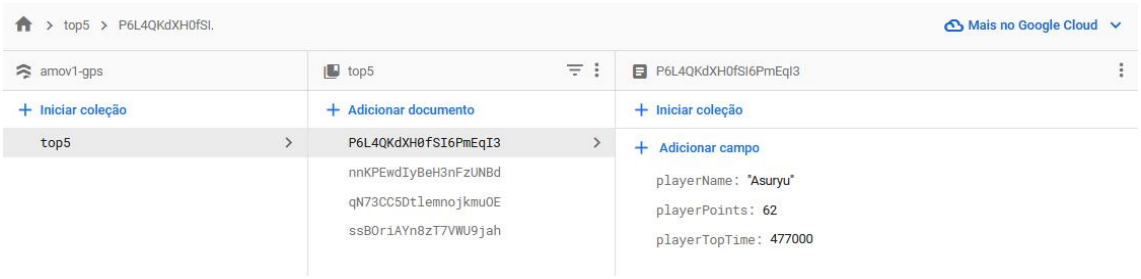
O jogador tem a possibilidade de definir um nome de jogador e uma imagem que vai servir para o identificar não só no Top 5 como também no modo multijogador.

Para guardar estes dados utilizámos a *external storage* e as *shared preferences* do dispositivo. Para aceder a esta funcionalidade o utilizador tem de dar permissão à aplicação para aceder a este recurso.

```kotlin
fun saveUsername(context: Context, username: String) {
    val prefs = context.getSharedPreferences( name: "pt.isec.amov.tp1", Context.MODE_PRIVATE)
    prefs.edit().putString("username", username).apply()
}

fun getUsername(context: Context) : String? {
    val prefs = context.getSharedPreferences( name: "pt.isec.amov.tp1", Context.MODE_PRIVATE)

    return prefs.getString( key: "username", "Jogador")
}
fun saveImage(context: Context, bitmap: Bitmap, filename: String = "avatar.jpg") {
    val cw = ContextWrapper(context)
    val directory = cw.getDir( name: "imageDir", Context.MODE_PRIVATE)
    val file = File(directory, filename)
    if (file.exists()) file.delete()
    FileOutputStream (file).use { out ->
        bitmap.compress(Bitmap.CompressFormat.JPEG,  quality: 100, out)
    }
}

fun loadImage(context: Context, filename: String = "avatar.jpg") : Bitmap? {
    val cw = ContextWrapper(context)
    val directory = cw.getDir( name: "imageDir", Context.MODE_PRIVATE)
    val file = File(directory, filename)
    if (!file.exists()) return null
    return BitmapFactory.decodeFile(file.absolutePath)
}

fun createFileFromUri(
    context: Context,
    uri: Uri,
    filename: String = getTempFileName(context)
) : String {
    FileOutputStream(filename).use{ outputStream ->
        context.contentResolver.openInputStream(uri)?.use { inputStream -> inputStream.copyTo(o
    }
    return filename
}
```

Gestão do Top 5

A gestão e apresentação do top 5 é feita com recurso à Firebase. A Firestore Database permite-nos guardar a informação sobre os jogadores num formato NoSQL fácil de aceder e interpretar. Todos os jogadores são guardados na tabela independentemente de terem entrado no Top 5 ou não. Só depois é que ao aceder às Leaderboards é que os resultados são ordenados e apenas obtidos os 5 primeiros.

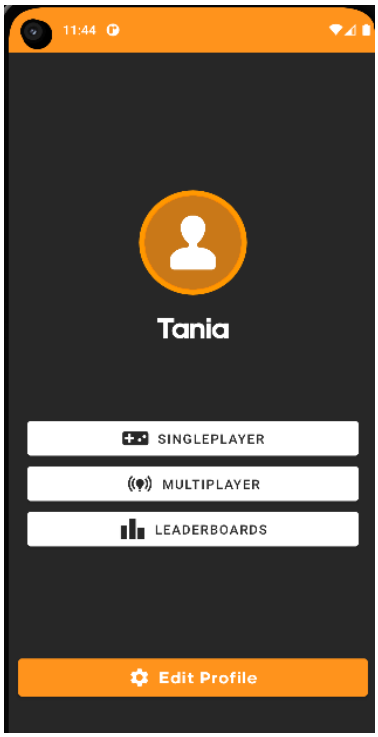



```kotlin
db.collection( collectionPath: "top5") CollectionReference
    .orderBy( field: "playerPoints", com.google.firebase.firestore.Query.Direction.DESCENDING) Query
    .limit(5)
    .get() Task<QuerySnapshot!>
    .addOnSuccessListener { documents ->
        var count = 1
        for (document in documents){
            Log.i( tag: "Asuryu",  msg: "${document.id} => ${document.data}")
            val card = layoutInflater.inflate(R.layout.top5_card, binding.cardWrapperTop5,  attachToRoot: false)
            val hashtag = card.findViewById<TextView>(R.id.top_hashtag)
            val name = card.findViewById<TextView>(R.id.player_name_top5)
            val points = card.findViewById<TextView>(R.id.player_points_top5)
            hashtag.text = "#${count}"
            count += 1
            name.text = document.data["playerName"].toString()
            points.text = "{document.data["playerPoints"] as long} pts"
            getPlayerAvatar(card, document.id)
            linearLayout.addView(card)
        }

    }
    .addOnFailureListener { exception ->
        Log.w( tag: "Asuryu",  msg: "Error getting documents: ", exception)
    }
```

Como extra, decidimos também tirar partido da Firebase Storage e adicionámos ao Top 5 as imagens dos jogadores. As imagens são guardadas num bucket criado para o efeito e as imagens são identificadas pelo nome do documento que se refere ao utilizador, desta forma podemos fazer a ligação entre a base de dados (top 5) e as imagens dos vários jogadores.

Ecrã de créditos e internacionalização

Com vista à internalização da nossa aplicação para android, optou-se por criar também uma versão em inglês, como se pode verificar na imagem a seguir apresentada.



Versão original: versão portuguesa

Versão inglesa

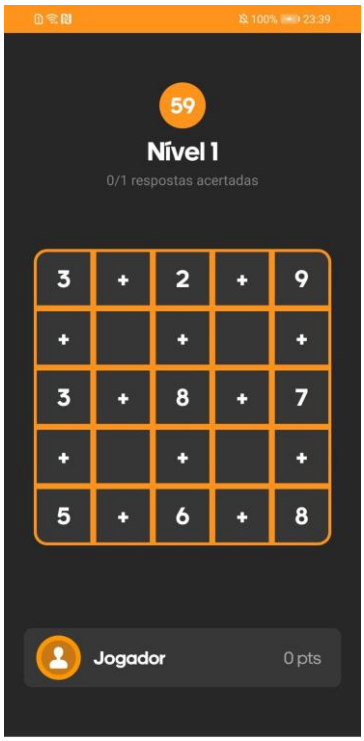
app_name	app/src/main/res	☑	9.5 a AM1	
singleplayer	app/src/main/res	☐	1 Jogador	Singleplayer
multiplayer	app/src/main/res	☐	Multijogador	Multiplayer
editar_perfil	app/src/main/res	☐	Editar Perfil	Edit Profile
alterar_fotografia	app/src/main/res	☐	Alterar fotografia	Change profile picture
jogador	app/src/main/res	☐	Jogador	Player
jogadorMp	app/src/main/res	☐	Jogador %1$d	Player %1$d
salvar	app/src/main/res	☐	Salvar	Save
edit_paragraph	app/src/main/res	☐	Nesta página poderás editar as tua	Here you can edit your profile and cl
points_placeholder	app/src/main/res	☐	%1$d pts	%1$dpts
points_card	app/src/main/res	☐	Pontos: %1$d	Points: %1$d
level_card	app/src/main/res	☐	Nível: %1$d	Level: %1$d
time_card	app/src/main/res	☐	Tempo: %1$d	Time: %1$d
nivel_placeholder	app/src/main/res	☐	Nível %1$d	Level %1$d
exit_dialog_title	app/src/main/res	☐	Sair	Exit
exit_dialog_message	app/src/main/res	☐	Queres mesmo sair do jogo?	Do you really want to exit the game?
sim	app/src/main/res	☐	Sim	Yes
nao	app/src/main/res	☐	Não	No
you_won	app/src/main/res	☐	Ganhou!	You won!
game_over	app/src/main/res	☐	Jogo terminado!	Game over!
try_again	app/src/main/res	☐	Tentar novamente	Try again

Editor de Traduções

Diferentes orientações

Visto que o Android destrói a atividade e cria uma nova sempre que existe uma rotação, foi necessário fazer uma adaptação ao código para que o ecrã de jogo pudesse suportar vários layouts.

Para contornar este problema criámos um construtor por cópia da classe Game para que recebesse um objeto da classe Game e copiasse toda a informação do objeto anterior continuando assim o jogo no mesmo estado em que se encontrava antes da rotação ter sido efetuada.



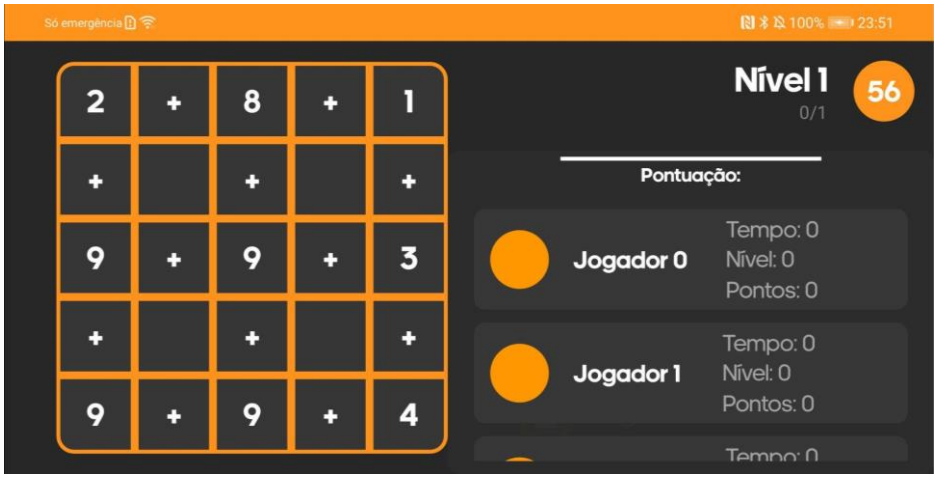
Robustez do código

Duas das tarefas mais importantes das linguagens são a gestão de memória e o tratamento de excepções. É aqui que muitas linguagens falham. O Java tem classes que tornam o tratamento de excepções totalmente previsível e transparente. O Java é uma linguagem que dá pouca liberdade ao utilizador no que respeita ao uso de tipos e declarações de dados, e determina os erros mais comuns durante a fase de compilação. Como se pode ver no exemplo desta imagem, em caso de ocorrer uma excepção "dentro do try", o programa imprime logo como output a excepção captada pelo catch. Desta forma, para além de uma melhor apresentação do código escrito, é também garantida uma melhor segurança de código.

```
try {
    socket = Socket(ip, SERVER_PORT)
    Log.d(TAG, msg: "Connected to server")
    // send the player info to the server (buffer size = 4096)
    socket0?.write(jsonOut.toString().toByteArray(Charsets.UTF_8))
    socket0?.flush()
    Log.d(TAG, msg: "Sent data to server")
    runOnUiThread {
        binding.serverIpLobby.text = ip
        binding.connectToServerBtn.visibility = View.GONE
        binding.startgamelobby.visibility = View.GONE
    }
} catch (e: Exception) {
    Log.e(TAG, msg: "Error connecting to server", e)
    runOnUiThread {
        Toast.makeText( context: this, text: "Error connecting to server", Toast.LENGTH_SHORT).show()
        //finish()
    }
}
```

Visualização inteligente de informação

Para visualizar a informação de vários jogadores de forma inteligente foi utilizado o layout BottomSheetLayout durante o ecrã de jogo para que o jogador, durante uma partida no modo multijogador, possa ver a pontuação dos outros jogadores neste jogo. Para isso basta arrastar a "sheet" que diz "Pontuação" a partir do fundo do ecrã para revelar uma lista de jogadores e os seus pontos.
Como não conseguimos implementar o modo multijogador, preenchemos essa "sheet" com informação aleatória que poderia facilmente ser preenchida caso o modo multijogador estivesse a funcionar.



Funcionalidades Não Implementadas

Todas as funcionalidades relativas à primeira meta foram implementadas exceto o modo de multijogador que não foi implementado na sua totalidade.
As restantes funcionalidades da aplicação encontram-se completamente implementadas, sendo que seria possível até colocar o jogo na Play Store para mais pessoas poderem jogar!