Guillem Frisach Pedrola
Francesc Ferré Tarrés

Aplicacions Mòbils i Encastades

Documentation Practices 2 i 3

Index

1 Structure of ANDROID practice

This practice involves performing the popular memory game called Simon Says for Android.

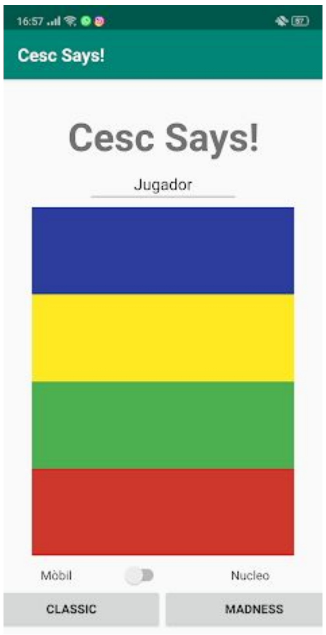
Once the game has been created, it must be connected via Bluetooth to the board and the LED on the program should illuminate.

It also includes a top 5 of the best players who have played on that mobile, regardless of whether the application remains active or not.

The links to the projects are as follows:

Project name	**Link to practice**
Practica2_AME (Android)	https://github.com/CescFT/Practica2_AME/
Practica3AME (Android + MBED)	https://github.com/CescFT/Practica3AME/

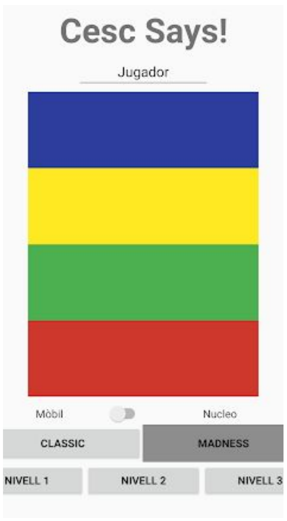
1.1 Design decisions ANDROID



Our game is generated by two different layouts, controlled by two different activities, the first layout gives you the option to type the name of the current player and allows you to choose between two game modes:

● **Classic:** In this mode, the game behaves as it always has. It goes from levels and in each of the levels the sequence of the previous level must be repeated, and so on.
● **"Madness" mode:** This mode is divided into three different levels. This mode, in general, is colored sequences, without phase dependence, that is, each sequence is generated new and randomly at the beginning of each phase. Depending on the level chosen (1, 2 or 3), more or less colors appear in the sequence.

We decided to implement two game modes so that our practice could add value. It's not a complex feature, nor a "revolutionary" feature, but it is clearly different from the classic Simon Says.



In case the player selects the game option " Madness", he allows you to choose the different levels, and the appearance is as follows (See image on the left margin of the paragraph).

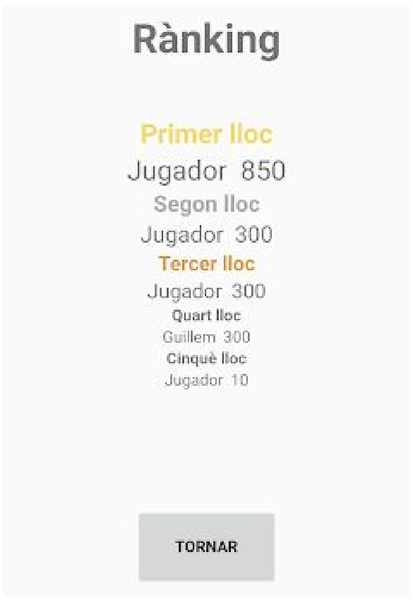
Each level is the representation of the number of increases per round, meaning that at level 1, each round adds a color to the sequence, at level 2, two are added, and at level 3, they are added three by three.

The decision to implement levels in "madness" and not in the classic is purely to maintain the classic mode as the name implies.

In Classic mode, if the player is mistaken, he is redirected directly to the second activity, which is the Best Player Ranking. If the user does not indicate his name on the main screen of the game, the default name is "Player".

It has been decided to implement the ranking only in the classic mode as the "madness" mode has different levels and different scoring modes. This would have required ranking with each of the different difficulties, and we felt it would be a mess for both application programming and player understanding, as you should be aware of which ranking you are in.

This activity is the one that shows the ranking, the appearance of this layout is as follows:



Rànking

Primer lloc
Jugador 850
Segon lloc
Jugador 300
Tercer lloc
Jugador 300
Quart lloc
Guillem 300
Cinquè lloc
Jugador 10

Shared Preferences are used to keep the information in the application persistent. A SharedPreferences object points to a file that contains key value pairs and provides simple methods for reading and writing them. Each SharedPreferences file can be private or shared.

A design decision made at this point was to minimize the space we use within the Shared Preferences. If we had not controlled it, the rankings would have increased uncontrollably with the results of the games played. In order to control the size of this list where scores are saved, what we did was save only the top 5 results. This way we will always have the number of results we have saved, and the size of the list. So we worry about optimizing the memory of the Android device of the end player.

The problem with ***Shared Preferences*** is that it is difficult to store a list of values here.

The JSON (JavaScript Object Notation) interoperable language has been used to keep the list stored. At the end of the game, what you do is store the new player you just played.

To keep the top 5 neat, what has been done is to generate a custom object that represents a player (it contains the name and the score obtained).

This class implements Serialization (for JSON to work) and the Comparable <T> interface. That way, the compareTo () method must be implemented and we can keep the list sorted using the sort () function of Collections:

```
Collections.sort(List<T>);
```

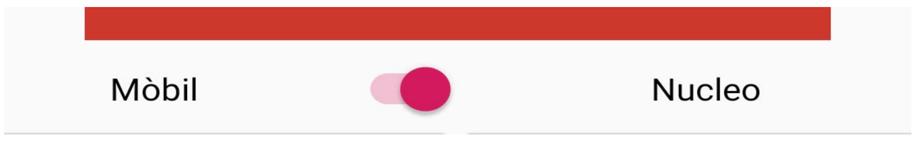
Although the list is organized, the meaning of it is descending (the smallest score is the first). In order to flip it over and sort it out as best we can, we just call Collections again with the reverse() method:

```
Collections.reverse(List<T>);
```

So the ranking is well organized.

Finally, the return button returns to the main screen and allows you to start a new game, keeping the player name.

In the event that before starting the game, the user activates the switch that is in the main layout, it allows us to reproduce the same board that is going on in the game (by turning on the LEDS). In order for it to work, the device we play with must be previously linked to the bluetooth on the board.



This way, with Bluetooth enabled, the board will repeat the color sequences and when a color is pressed, the LED will also light up.

1.2 Additional elements implemented

To make it a more lively game, and as a simple added value, it has been decided:

- Assign each button (representing a color) a different sound. You have chosen to assign a music note.
- The buttons for choosing one game mode or another also have a sound assigned to them. This sound is a simple "click".
- The phase change also has a distinctive sound, it was decided to include a sound in the phase change (a bell) so that the user is aware that he has exceeded the level.
- When the player plays in classic mode and loses, the mobile phone vibrates to indicate that the game is over.

2 Bluetooth communication with the board

In order to communicate with the plate, the following steps must be followed:

1. **Enable Bluetooth from system settings:** Before turning on the application, if you want to play with the board, you must enable Bluetooth from the settings, just before entering the application.
2. **Enter the game and activate the switch:** This will allow Bluetooth messages to reach the board.
3. **Play in any of the two game modes:** While playing, the LEDs will open and off according to the sequence of colors that emerge and also when the player presses a button.

In addition, the plate is also aware of the following events:

- ***Game Over:*** Informs the player who has lost, because he has made a mistake.
- ***Phase Shift :*** Informs the player when a phase has passed.
- ***I can continue playing:*** It means that the player was not confused by clicking on the color he was playing.

2.1 Implementation of Bluetooth communication

The source code used for Bluetooth communication, in both cases was searched on the Internet:

- Bluetooth for Android: https://stackoverflow.com/questions/22899475/android-sample-bluetooth-code-to-send-a-simple-string-via-bluetooth
- Bluetooth for the board: https://os.mbed.com/questions/83079/Send-data-between-PC-Monitor-and-Bluetoo/

Basically the code on Android is very simple, it simply retrieves the devices paired with the mobile device and through code it connects. If there are no devices connected, write to the Logs that Bluetooth is not found or there are no paired devices.

The Bluetooth messages we send are as follows:

- **"1"**: Red button pressed or in sequence and the LED must therefore be switched on.
- **"2"**: Green button pressed or in sequence and the LED must therefore be switched on.
- **"3"**: Yellow button pressed or in sequence and the LED must therefore be switched on.
- **"4"**: Blue button pressed or in sequence and the LED must therefore be switched on.
- **"5"** or **"0"**: Informs the player that he has not been confused by pushing the button and that he can continue playing.
- **"7"**: Informs the player that he has passed a level or phase.
- **"8"**: Tell the player that he was wrong and what he lost.

These codes are those that the Android application sends to the board via Bluetooth wireless communication.

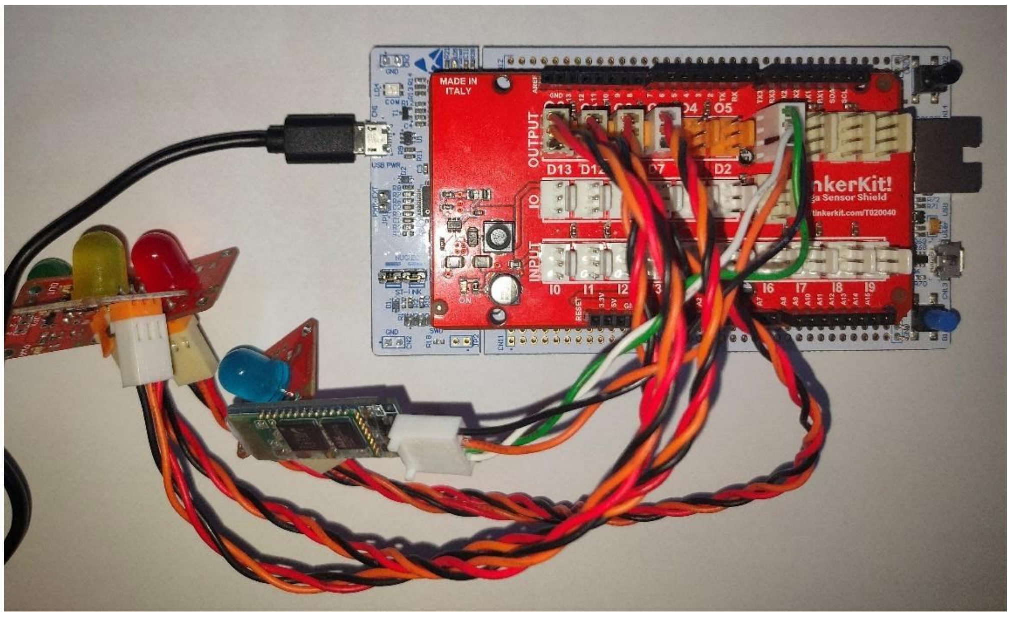
The board receives them via the Serial port prepared by Bluetooth (D1, D0). In this it receives in *chars*. Then, in order for communication to be effective, you must see the codes in which ASCII values are represented:

Character or String	ASCII number
“0”	48
“1”	49
“2”	50
“3”	51
“4”	52
“5”	53
“7”	55
“8”	56

Depending on what you receive, the plate has one behavior or another, as described above.

2.2 Connection manual

In order for our board software to work properly, you need to connect the LEDs and the Bluetooth circuit to the corresponding pins. It should be connected as in the following image:



More specifically, the connected pins are:

O0	D11	Green
O1	D10	Blue
O2	D9	Red
O3	D6	Yellow
Serial 1	(D1,D0)	Bluetooth circuit

2.3 Implementations not reached

An attempt was made to allow gameplay from the board, that is, the user had the choice of whether to play from the mobile device or by entering the values through the console.

It got implemented, and it worked by passing codes between android and the board. Each code had a meaning, and these are the same as stated above.

For each value that is entered by console, the board communicated to android with a char that, the same application verified following the same process that follows if it is played from the mobile.

If the user input was correct, android sent an "all ok" message to the board, which again asked the user for input.

The downside we encountered was a serious concurrency issue. Since both the functions (read on keyboard and listen on android) of the board were in the while inside the main, the function that listened and turned on the LEDS did not have time to show the sequence to the user, and the board it was locked waiting for user input.

We know that we should have applied multithreading, but we didn't have the necessary knowledge.