TEACHING UNIT 5

Introduction to Artificial Collective Intelligence

DEVELOPING AN ARTIFICIAL INTELLIGENCE CURRICULUM ADAPTED TO EUROPEAN HIGH SCHOOLS

2019-1-ES01-KA201-065742

erasmus.aiplus@udc.es

UNIVERSIDADE DA CORUÑA

Co-funded by the Erasmus+ Programme of the European Union

INDEX

1. INTRODUCTION

Collective intelligence refers to the response that emerges from the collaboration, collective efforts, and competition of many individuals. It is a concept normally associated to groups of humans and studied in social sciences, or to groups of animals and studied in ecology or biology. In the scope of this curriculum, we are interested in Artificial Collective Intelligence (ACI), that is, the intelligence that can be achieved when multiple AI systems are interconnected. A key aspect of ACI is that human users can be considered as elements of the system too, but the main components are artificial.

Within the field of AI, artificial collective intelligence has been studied from three main perspectives or sub-fields:

- *Distributed Artificial Intelligence (DAI):* it is an approach to solving complex learning, planning, and decision-making problems in parallel. Nowadays, DAI is focused on distributed problem solving, that is, how a complex problem can be decomposed into simpler ones, and the solutions can be synthesized. DAI exploits large scale computation and spatial distribution of computing resources.
- *Multi-agent systems (MAS):* this field arose from DAI, and it is focused on developing computational systems composed of multiple intelligent agents, which coordinate their knowledge and activities and reason about the processes of coordination. Agents are virtual entities that can act, perceive its environment and communicate with other agents.
- *Swarm intelligence*: it is the collective behavior of decentralized, self-organized agents. The main difference with MAS is that swarm intelligence is based on simple agents, with no individual intelligent features. The intelligence emerges from the interactions between them. A clear example of swarm intelligence are the algorithms that try to mimic the behavior of insects, for instance, ants, which cannot be considered as intelligent individually, but they take complex decisions when acting as a swarm.

Summarizing, ACI relies on the existence of independent AI systems or intelligent agents, that are distributed in the same computer, or throughout a network. These agents can operate independently and perform actions that solve a specific task or sub-problem. In this scope, the agents can be simply software agents, as in MAS, which operate in virtual environments, but normally they will be **real-world devices** (robots, cars, appliances, and even human users). We can see ACI as a generalization of MAS to the embedded intelligence.

A key feature in ACI is the connection between the agents to exchange information and solve more complex problems. To this end, the main concepts to be learned by students about ACI are related to **communications and coordination**, that is, how to manage the information provided by the agents and their actions to achieve the desired intelligent response in the group.

Fig. 1 was presented in the introductory TU (TU0), and it shows a representation of the AI ecosystem that will be managed in this curriculum, containing the considered AI topics. This

figure corresponds to an ACI system, made up of AI systems or agents (semi-spheres) that are interconnected by the red and blue arrows. As observed in the figure, the AI topics covered in the previous TUs are present in each of the agents of the ACI system.

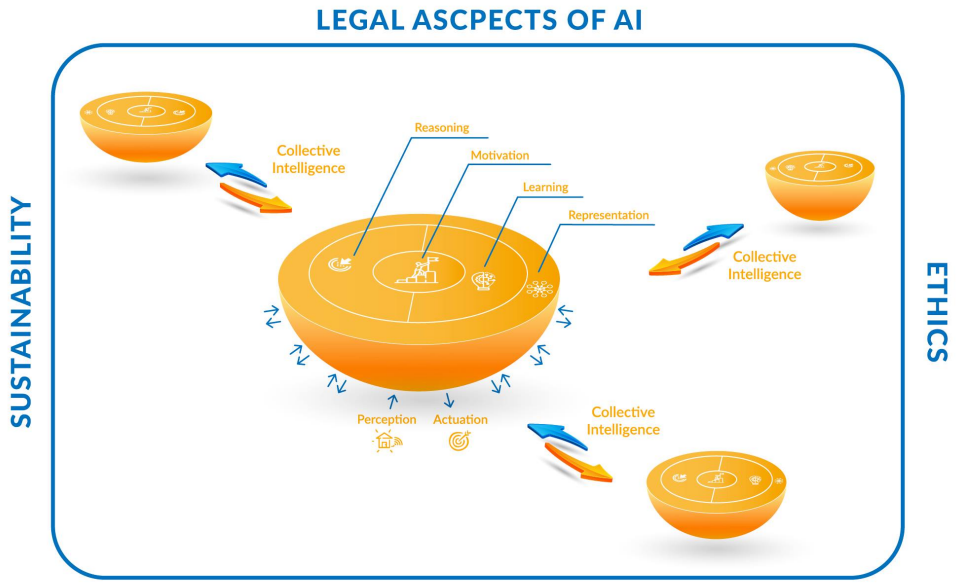


Fig. 1 AI ecosystem

To simplify the curriculum, we will focus on the specific application field of **Internet of Things (IoT)** to introduce ACI to students. IoT stands for a network of physical electronic devices which share information in a coordinated fashion through the internet. In the coming years, IoT will be updated as **AIoT**, that is, Artificial Intelligence of Things, meaning that the response of these networks of devices will show more intelligent features. According to the previous definitions, **AIoT is a ACI system where the agents operate in the physical world, and they communicate using the internet**. Hence, the main challenge in AIoT is still the same: developing a proper coordination between these devices so the collective response improves the individual one.

Fig. 2 AIoT environment

Fig. 2 summarizes the concept AIoT that will be studied with students in the curriculum. We can observe a classroom with computers, robots, smartphones, TVs, intelligent speakers and cameras connected to the same network. Each of the devices can be controlled by an independent intelligent agent, but from their coordinated response, artificial collective intelligence arises. For instance, this ACI classroom could help the teacher obtaining information from the internet about a question shared by many students, which can be detected using speech recognition, and illustrated using the TV. Or it could notify students to calm down if the sound level is too high. In Fig. 2, we can also observe people interacting between them and with the devices. As commented above, this is a key aspect of an ACI system. **Instead of considering human as simple users, they are included in the system as another agent that provides information and acts in this collective environment**.

Focusing now on this introductory TU, the objective is that students develop a simple AIoT system to realize the implications of using information provided by other AI systems. They will work on how to obtain real time data from a cloud database, how to represent it internally, how to reason with it, and how to coordinate the response of the ACI system. Specifically, they will improve the smartphone app developed in the previous TU, which manages a Scavenger Hunt searching game in the school based on machine learning. Now, they will have to include information about the gamer's achievements to create a more realistic app considering that this is a collective game.

2. CONTEXT

In order for the students to adequately meet the learning objectives of this TU, they must have the following prior knowledge:

- **Programming**: students must have experience in block-based programming, creating simple algorithms based on conditionals. In addition, in this TU they will use functions and lists of numbers (arrays), so it is recommended to review their basics.
- **Basic knowledge of MIT App Inventor:** students should have basic knowledge about the App Inventor application (https://appinventor.mit.edu).
- **Mathematics:** as this TU is a continuation of the previous one, students should have revised the fundamentals of algebra and functions, so they understand how the classification model works. For this specific TU, the only new topic is **time measurement**, so teachers should consider if it necessary to reinforce time related concepts to students.
- **Perception, actuation, representation and reasoning basics:** it is mandatory that students have finished TU2 and TU3, so they have the fundamentals explained in these teaching units.
- **Learning in AI:** it is mandatory that students have finished TU4 and they know the basics of learning in AI, and the operation of the "Capture it!" app created in that TU.

3. LEARNING OBJECTIVES

Once students have finished this TU, they will have acquired the following knowledge:

SPECIFIC	• Fundamentals of artificial collective intelligence • Storing and retrieving data in/from the cloud • Centralized vs distributed communications
TRANSVERSAL	• App Inventor software usage • Computational thinking: variables, conditionals, lists and functions • App interface design for computer games • Time counters in smartphone apps

4. CONTENTS

The following specific contents of artificial collective intelligent will be studied in this TU:

Fig. 3 Specific contents

5. TIMELINE

This TU is the fifth of six focused on introduction to AI. Remember that, after the completion of the previous TUs, students have already seen the fundamentals of perception, actuation, representation, reasoning and learning on AI, which are required to properly understand artificial collective intelligence. The next TU (TU6) will be focused on ethics, sustainability and legal aspects of AI.

The estimated time for this TU is **4 classroom hours**.

6. NECESSARY RESOURCES

The following hardware elements are required to carry out this TU:

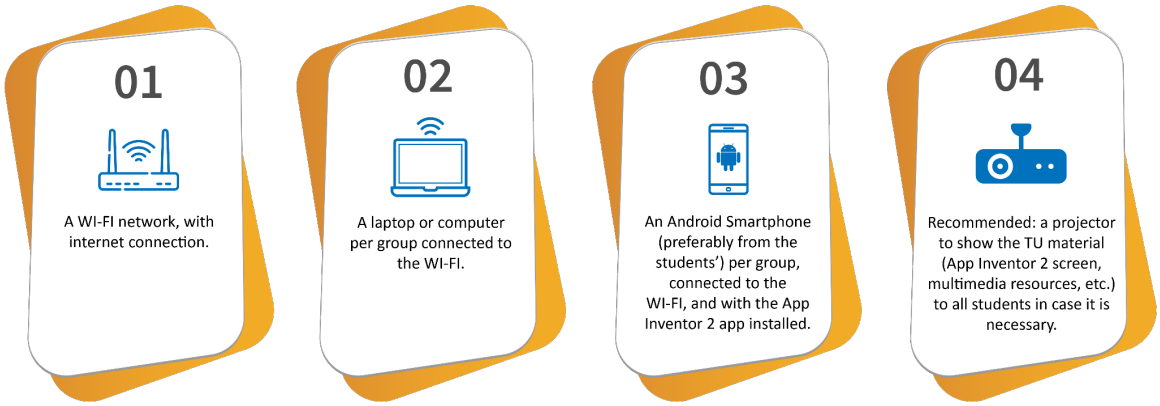


Fig. 4 Hardware elements required to carry out this TU

Regarding software, the following elements are required to carry out this TU:

1. Every group must have an active App Inventor account.
2. *TU4 app:* in this TU, students will create an improved version of the app developed in TU4. Therefore, it is recommended that each group uses their App Inventor project file corresponding to the Capture it! App. However, this TU includes a specific template for those groups who did not complete the previous TU, or for those who prefer to follow the specific recommendations provided in this TU. This new template is called "appInventor_template_TU5.aia".

NOTE: the USE OF THE TEMPLATE IS NOT MANDATORY, but the solution to the challenge included in this TU will be based on it. Those teachers who prefer to provide more freedom to their students in the app development or to reinforce the programming aspects, should start the project from scratch. In any case, the specifications established in section 9.1 must be respected in order to achieve the same result in all groups.

7. BIBLIOGRAPHY

- Artificial Collective Intelligence: https://bit.ly/31z2GuH
- Handbook of Collective intelligence: https://bit.ly/3u46SPaç
- Artificial Intelligence of Things (AIoT): https://bit.ly/3m6XnMw
- Internet of Things: https://bit.ly/2QIPrW3
- IoT and machine learning: https://bit.ly/2PHkvoH
- Swarm intelligence: https://bit.ly/3cDOFm0
- Ambient intelligence: https://bit.ly/2PMBiGN
- Cloud computing: https://bit.ly/3cDONSw

8. GROUPS

The project has been designed to be carried out in groups of 2 students. Each member of the team will have one role: one will be a *programmer* and the other a *manager*.

Although in this TU the main objective is that both students collaborate in the app programming, the *programmer* will be focused on the development of the program in App Inventor exclusively. On the other hand, the *manager* must handle all the aspects required to carry out the app properly, like taking notes, asking questions to the teacher, managing the time, and submitting deliverables to the teacher. It is very important that both members are always in agreement and aware of what each other is doing. That is, although the programmer monitors the programming, the manager must understand and agree with what the programmer is doing and vice versa.

The roles should be changed at least once in each session, so that every student performs both.

9. TEACHING UNIT MATERIAL

This unit has extra documents, which are available at:

- https://bit.ly/39uUf87

These documents are:

- Current document, with Teaching Unit 5 (TU5_Collective_Intelligent.pdf)
- Video of the app functioning (TU5_APP.mov)
- App Inventor solution for teachers (appInventor_solution_TU5.aia)
- App Inventor template for the student (appInventor_template_TU5.aia)
- Unit feedback document for teachers (teacher_feedback_TU5.docx)
- Diagram of the App Inventor project (diagram_TU5.pdf)
- Students' work (TU5_Students_work.pdf)

10. CHALLENGE / PROJECT

10.1 FINAL OBJECTIVE

WHAT TO DO?

Develop a smartphone app using App Inventor that allows to play Scavenger Hunt searching game in the school based on machine learning and using information from all the players.

NOTE: It is highly recommended to start from your app inventor file of TU4. If you did not develop it, you should download the template file *"appInventor_template_TU5.aia"*

HOW IT SHOULD WORK?

Watch the video "TU5_APP.mp4" included in the TU resources (and the video "TU4_APP.mp4" if you don't remember how the local game was created) to understand how this app should work, and the improvements that have been added to the game.

As shown in the video, the app allows now to play against others, creating a collective game. It maintains the intelligent features of the app developed in the previous TU, and it includes now global information of other players' advance, taken from the cloud. It is very important to understand that all the players must install the same app version, so students have to share it between them.

NOTE: Connected players will be those using the same app version (different versions of the game will not be connected). Consequently, the objects that players must find are the same for all.

APP SPECIFICATIONS

These specifications must be present in all the apps created by students, even those that do not follow the project template. The app has to:

1. Use a cloud database accessible from App Inventor (we recommend using *CloudDB Chat App*)
2. Before starting the game, in order to identify the players, each user must enter his or her name and it must be saved in the cloud database.
3. Each time a player finds an object, it must be notified to the other players which object was found and by whom.
4. Once one player finds all the objects, it must be notified to all the users that they have lost and finish all games.
5. As time goes by, users must receive different messages warning them. For example, every minute.

10.2 ACTIVITIES

NOTE: This section is aimed at teachers whose students will complete the TU in a teacher-guided manner. It organizes the project into activities, indicates what the student should do in each of them, and contains the solution for the teacher. Even in the case that students do not need instructions, we recommend teachers to read this section in order to have a clear idea of the proposed solution and the aspects where most emphasis should be placed.

Dividing a global challenge in tasks that lead to the challenge completion is a key competence in STEM methodology that students must acquire. In these initial TUs, we will provide such division to show how it can be carried out in different problems.

Fig. 5 displays a diagram that must be explained to students, and which summarizes the steps that must be followed to solve the challenge. First, some background knowledge about ACI must be acquired, in this case, attending the teacher's explanation. Then, two activities are considered, one focused on sending, receiving and managing the messages about the found objects, and the other focused on warning users about the elapsed time.

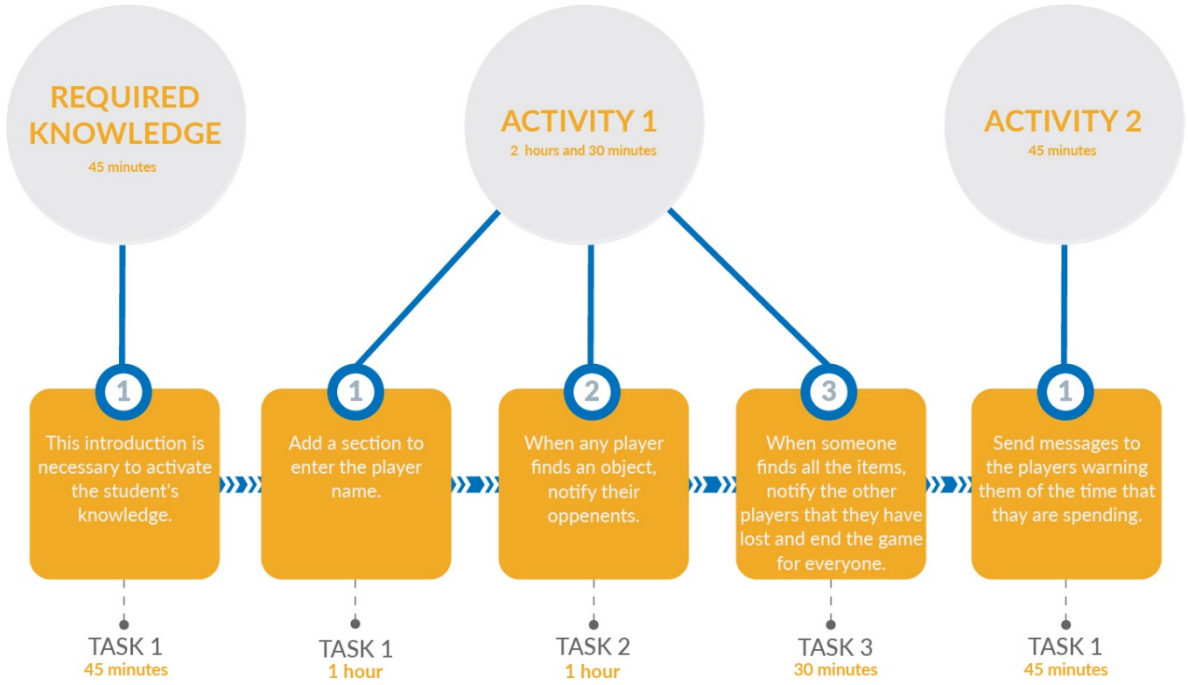
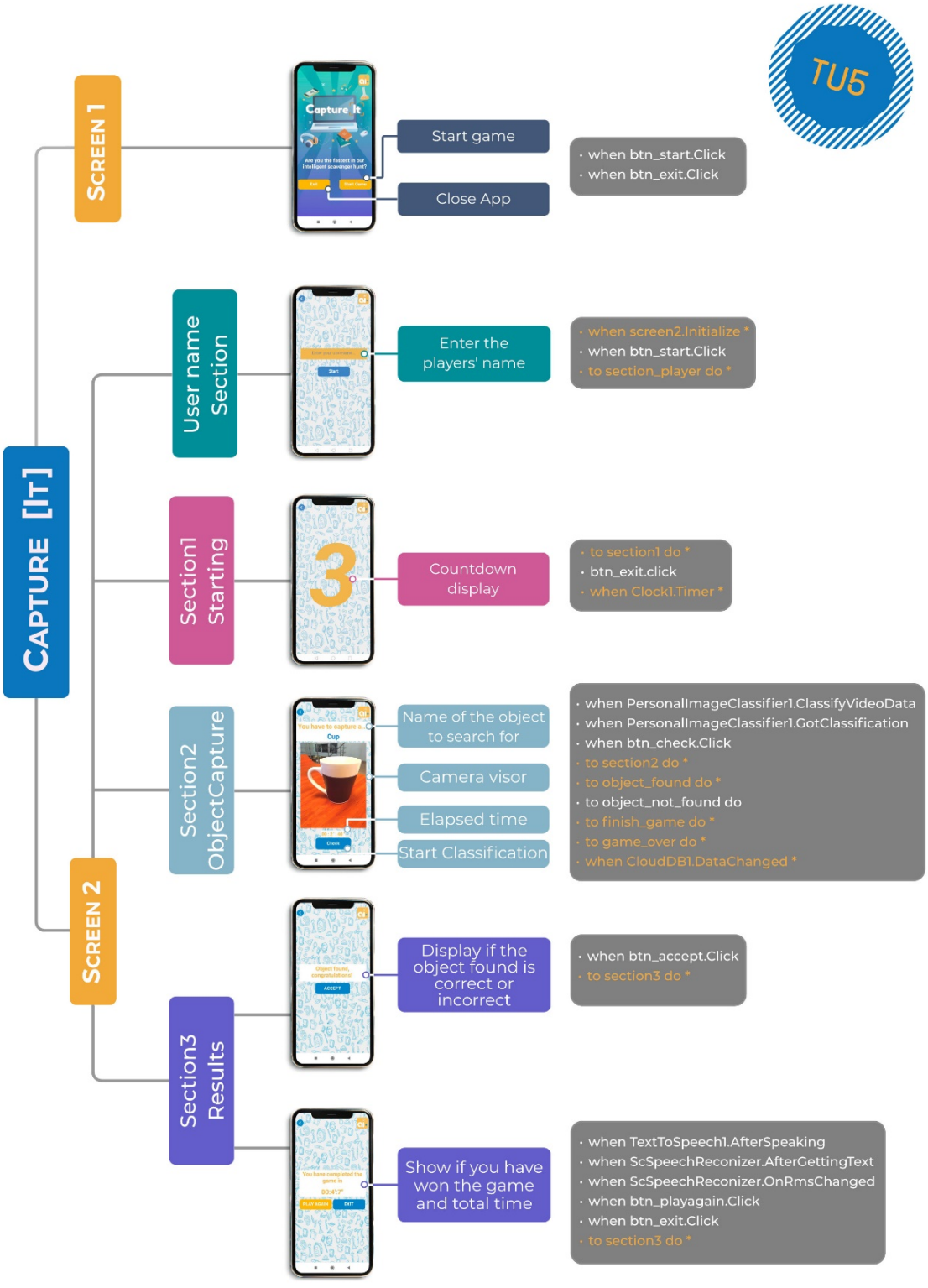


Fig. 5 Flowchart of the application development process

The following figure shows a diagram of the App Inventor project contained in the template file. **Both students and teachers should review this diagram carefully** in order to understand the overall organization of the different screens and sections that make up the app. In this diagram, the already implemented functions are shown in grey color, while those they must develop are in orange, to provide a clear overview of what they have to implement to face the challenge.

In the case that students do not start from the template, but from their game created in TU4, some functions may have a different name, for instance "*to object_found do*" and "*to object_not_found do*".



** The orange functions must be developed by students*

Fig. 6 Diagram of the App Inventor project

Required knowledge

This introduction is necessary to activate the student's knowledge

45 Minutes

Required knowledge to be introduced to students:

For a start, it is proposed to **perform a short discussion (20 min maximum)** in the classroom, in which all the groups give their opinion about what artificial collective intelligence (ACI) is and what it reminds or sounds like. During this period, students can search on the web about the following topics related to the field of Artificial Intelligence of Things (AIoT), which will help them to figure out what is this topic about:

- Internet of things
- Smart environments
- Home automation
- Smart cities
- Ambient intelligence

For instance, the following videos can be used:

- Smart cities: https://bit.ly/2POhSSg
- Internet of Things: https://bit.ly/2QIPrW3
- Artificial Intelligence of Things: https://bit.ly/3m6TtDc

After this initial contact, teachers should focus the explanation in one of the basic elements of ACI, communications over networks. To do it, many web references can be used:

- https://bit.ly/3wgbWSr
- https://bit.ly/2PMBJkp
- https://bit.ly/3rGkgHI
- https://do.co/39xuELE

The objective is that students understand the main elements required to communicate AI systems between them: network infrastructure, protocol and data.

Moving now to ACI systems, an important topic derived from the previous one is that of **centralized vs decentralized** communication, which must be introduced to students. The idea is that the information that is exchanged between AI systems in real time can pass over a central point that distributes it over the collective systems, or it can be directly passed between agents. The first approach is called centralized and the second one, decentralized:

- https://bit.ly/3m69vx
- https://bit.ly/2QXyYxD

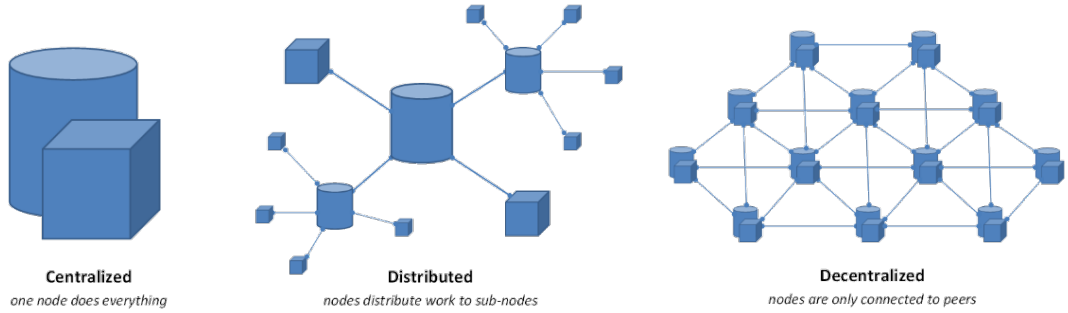


Fig. 7 Representation of centralized, distributed and decentralized communication networks

In this educational level, what is relevant for students is to understand that centralized ACI systems are simpler to implement, because all the information is available for all the members at the same time, and the communications are simpler too. But, on the other hand, if this central element fails, all the system can collapse because no information is shared. Decentralized ACI systems are more robust, because their coordination does not depend on one of them, but at the same time, maintaining coherence in the information is more complicated. Imagine a multi-robot system with decentralized communications. In a given instant of time, one robot could detect something important for the global group, but as it could be far from other robots, the information exchange could not be immediate, which implies a degradation on the system response.

Finally, and directly related with this TU, teachers should explain to students the fundamentals of cloud storage and cloud services, so they can clearly understand how data is stored in the cloud and how this information can be used remotely. This concept opens the opportunity to create intelligent apps by using information from distant places to take a decision, which is the basic concept we must introduce in this TU:

- What is the cloud?: https://bit.ly/3m5MFFW
- Cloud computing basics: https://bit.ly/2PNtxQU
- Cloud computing: https://bit.ly/3cDONSw

Summarizing, in this TU, students will develop a centralized ACI system based on cloud storage. Specifically, their app will share game information of all the users in order to create a network version of the Capture it! App.

Activity 1

To make the game notify the online users of the objects found by the other players

2 Hours and 30 Minutes

Activity 1 – Task 1

1

Add a section to the game for users to enter their player's name

1 Hour

Student´s work:

In this new version of the game, multiple players will be able to be connected at the same time and they will be notified of objects found by their opponents.

So, **the first change in the app must be to include a module that allows to enter the name of the different players.**

Taking into account that it is necessary to start from the game created in the TU4...

Where should the module be?

Remember that the app has two screens: the initial one and a second one that was divided in three sections. We propose to include this module as a new section of the second window. Thus, when the user clicks "start game" in the first window, the app opens this module on the second window and the user can introduce his/her name. Once done, a new button (starting) already created in the second screen in TU4, appears and the game can start.

Note: Look at the sections in screen2 to know what the format of this new module should look like.

What should the module include?

The module must include a text box in which the player writes his/her name and a button to start the game. In addition:

1. The game must start by clicking this new button and not another one.
2. It´s necessary to check that the user has written his/her name and save it in a variable.

1 Hour

Test:

It is necessary to check if the student has carried out this task correctly. To do it, **when students finish the task, it is necessary to check items 1, 2, 3 and 4 of the checklist in the Annex II: Teachers program test.**

Teachers' solution:

The first step to overcome this task is to remember what the format of the sections on Screen2 was like. As can be seen in the following figure (Fig. 8), the screen was divided into three sections, which were limited by a vertical arrangement in such a way that, in order to activate one or another, it is only necessary to make the desired vertical arrangement visible and the rest invisible.

Fig. 8 Left: First section of the Screen2. Middle: Second section of the Screen2. Right: Third section of the Screen2

Taking this into account, a possible solution to create this new section would be to add a vertical layout and introduce the necessary components into it: a text box and a button. An example of solution is shown in Fig. 9. In this case, it has been decided to put the text box with an orange colour, and the button with blue. On the other hand, three arrangements were added between the components in order to improve their appearance. It is not mandatory to do it as shown in the figure, but it is important that students pay attention to the app aspect to make it friendly to users.

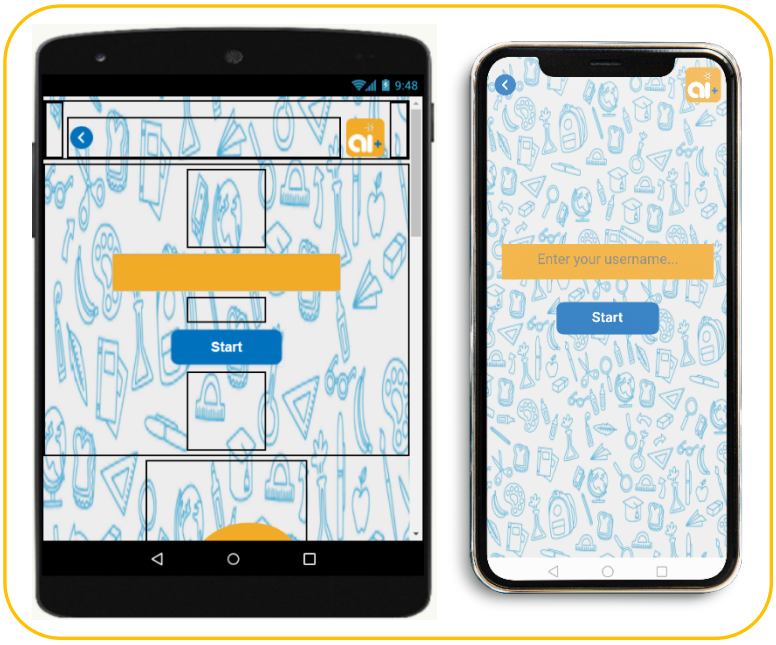


Fig. 9 Left: New section of Screen2. Right: Screenshot of the new section

In addition, it is advisable to modify the name of the vertical arrangement that encompasses the entire section to make it easier to find when programming. In this case, as shown in the figure, it has been called *vertical_player*.

Fig. 10 Vertical arrangement name

The first step to program the user name input is to make this new section the one that is visible when screen2 is initialised. To do this, it is necessary to set the section created as visible, and the rest as not visible. If you look at how this was handled in the previous TU, you can see that there were three functions called section1, section2 and section3. In these, the vertical layout they refer to was made visible. Accordingly, to continue with this format, one solution would be to create a new function as shown in the figure below.

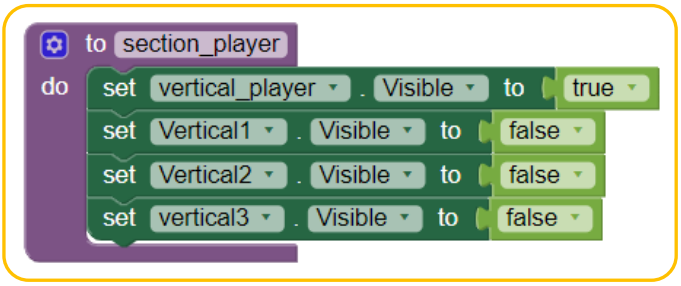


Fig. 11 Section_player function

Once the function has been created, it must be called up in the appropriate place. We want this section to be visible when Screen2 is started, so it is necessary to include it in the event when *Screen2.Initialize*. In this event there was a call to the function section1, which has to be removed to the *section_player* function. The final appearance of this event should be as shown in the Fig. 12.

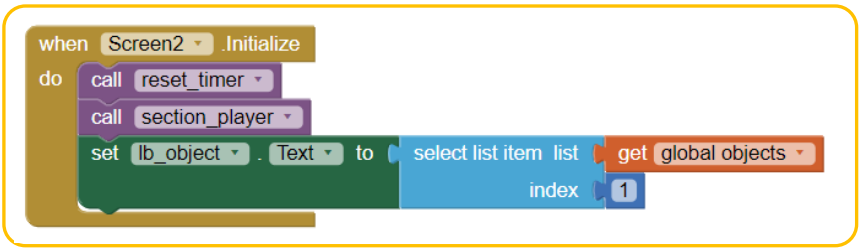


Fig. 12 Screen2.Initialize event

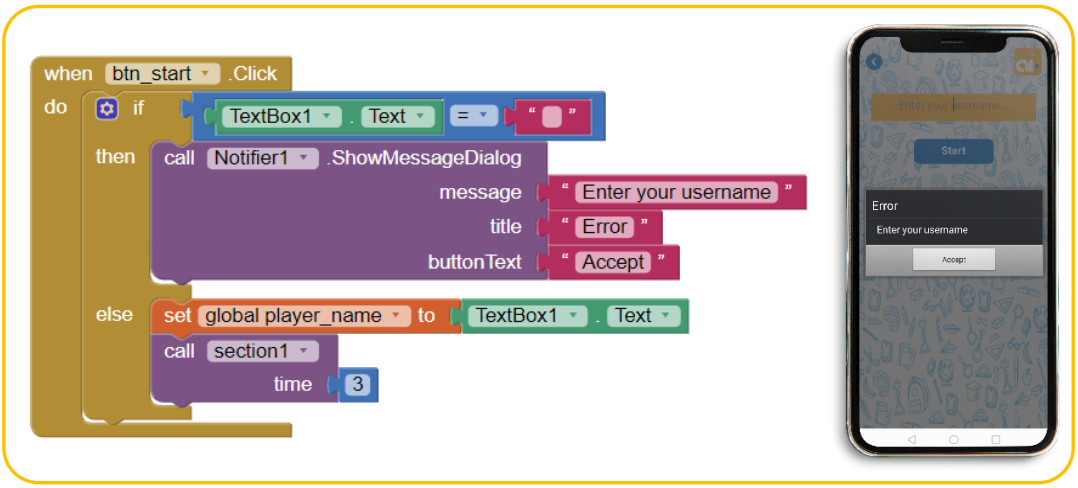


Fig. 13 Left: Event when the start button is clicked. Right: Screenshot of the error message

The last step is to program the start button. When it is clicked, it is necessary to check that the user has typed his/her name. If this is the case, it is time to **save the name in a variable** and start the game by calling the game's start section (section1). If the name was not introduced, we must notify the user of this error (as in the

Fig. **13** Right). A possible solution of the programming is shown in the

Fig. **13** Left.

Note: Students should test the proper functioning of the user's name input, which must start the game or show the error message

Activity 1 – Task 2

2

When any player finds an object, notify their opponents

1 hour

Required knowledge to be introduced to students:

This activity implies to use a cloud database, in this case, the one provided by App Inventor: *CloudDB Chat App*. This uses a component to store data in the cloud and allows multiple users of the app to communicate over multiple devices. Obviously, using information stored in the cloud does not transform an app in "intelligent", but being able of using remote information as this, is the starting point for ACI. In fact, this new version of the "Capture it!" game is in fact an ACI app, because it is based on an intelligent app based on machine learning.

At this point, we recommend teachers to read the original documentation of CloudDB and explain the basics to students before proceeding with the task:

- https://bit.ly/3wgcZ4P

Student´s work:

All the players of the game must run the same app version on their smartphone. So, the objects they have to find are the same (those trained for the model used in the app).
The objective of this task is that the application notifies the users when another player finds a particular game object. To do this, the CloudDB component will be used, and the following steps must be performed:

1. Add the CloudDB component in the appropriate screen (its properties should remain in default values).
2. When a user finds an object, his/her name must be saved in the cloud, together with a label that indicates the specific object found (name, object number...). To do this, the "call CLoudDB.storeValue" block can be used.
3. Every time a new item is saved in the cloud, it must be notified to all users, except to those who found it. To do it, the "CLoudDB.dataChanged" block can be used. Every time a player finds an object, this event is executed in all cases (even in the case of the user who just found the object). That is why it is necessary to control not only the object but also the player's name.

Note: **The best way to test an app using CloudDB and multiple users, is to use multiple devices. The better option is to build the apk, and install your app on more than one mobile device. From the Build menu, choose "App (provide QR code for .apk)". Once the apk is built, scan the generated QR code, and install the apk on multiple devices. Then start with the game and when someone found an object, messages should appear in the rest of devices.**

1 hour

Test:

Once the students have finished and tried out everything in this task, they must show the teacher that it works, and **he/she will cover the points 5 and 6 of the checklist in the Annex II: Teachers program test.**

Teachers' solution:

To notify players when an opponent finds one of the game objects, we need to store this information in a location that can be accessed from all devices simultaneously (**centralized communication**). As commented above, we propose to use the cloudDB component of App Inventor, which manages this information easily. In order to use it, it is necessary to add the non-visible component CloudDB, which is located in the storage palette **¡Error! No se encuentra el origen de la referencia.**), to screen2.

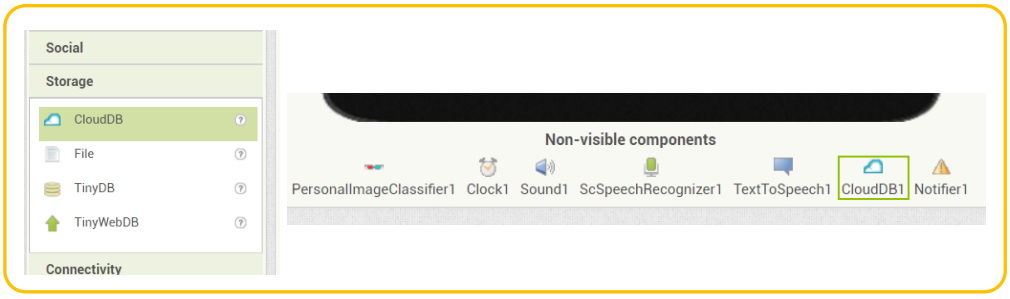


Fig. 14 Left: Storage palette. Right: Non-visible components

The next step is to store in the cloud the objects that users find. To do it, the call *CloudDB.StoreValue* block will be used. As you can see in **¡Error! No se encuentra el origen de la referencia.**, the block has two entries: *tag* and *valueToStore*. The *tag* will be the name of the player previously saved in a variable, and the *valueToStore* will be the found object name or code. In the solution proposed in **¡Error! No se encuentra el origen de la referencia.** and **¡Error! No se encuentra el origen de la referencia.**, it has been decided to store the name of the object.

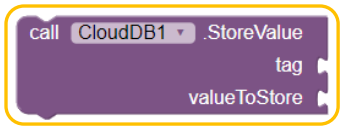


Fig. 15 CloudDB1.StoreValue block

This block must be executed every time the player finds an object, so the best place to put it is inside the *object_found* function and in the *finish_game* function (both created in the previous TU).

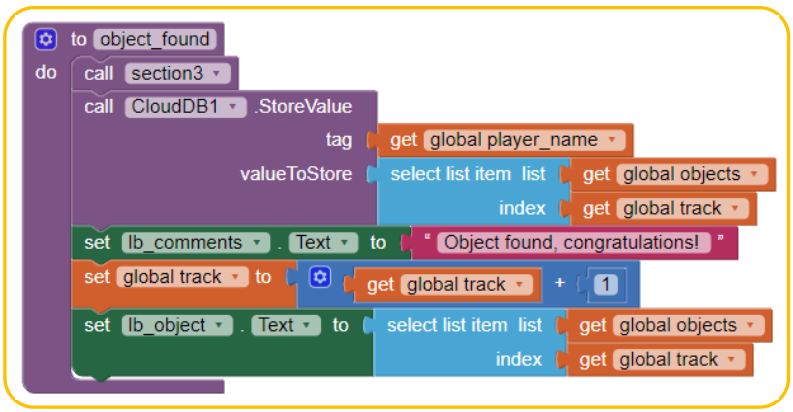


Fig. 16 Object_found function with CloudDB1.StoreValue block

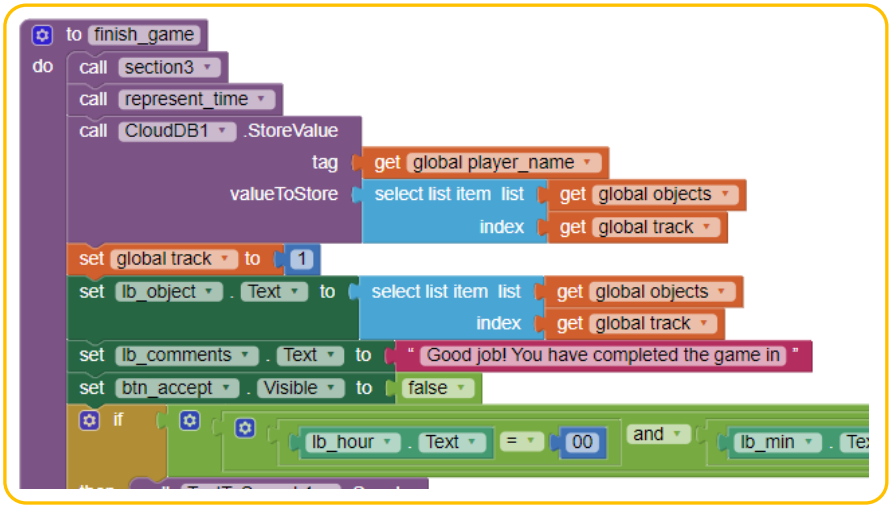


Fig. 17 Finish_game function with CloudDB1.StoreValue block

At this point, every time a user finds an object it will be saved in the cloud. Consequently, the rest of the players should be notified. To do it, the CloudDB component has a block called *when CloudDB.Datachanged*, that returns the tag and value of every new entry.

But before notification, it is necessary to check that the new object has been found by an opponent, which can be easily performed just by checking if its tag is different from the player's own. In the solution shown in Fig. 15, an alert has been included which reads: "*player name* just found the *object found*".

Fig. 18 Left: CloudDB.DataChanged event. Right: Screenshot with the message

Activity 1 – Task 3

3

When someone finds all the items, notify the other players that they have lost and finish the game.

30 Minutes

Student's work:

When one of the players finds all the items, the others will be notified that they have lost and will move on to the final screen where they will be asked if they want to play again (section3 of screen2). To achieve this, you must take into account that:

1. You must add "something" to the event when *CloudDB.DataChanged* to control if the object found is the last object.
2. If the object found is the last one, the rest of the players must finish the game. In the previous version of the app (TU4), when the game was finished, the only thing to do was to call the *finish_game* function. In this case, you have to:
   - Go to section3.
   - Turn off the *accept* button.
   - Write on the label that the player has lost.
   - Ask the player, using the *TextToSpeech* component, if he/she wants to play again.

40 minutes

Test:

Once the students have finished and tried out everything in this task, they must show the teacher that it works and **he/she will cover the points 7, 8 and 9 of the checklist in the Annex II: Teachers program test.**

Teachers' solution:

The first step in this task is to control when a player finds the last item. As mentioned in the previous task, every time a player finds an object, his/her username and the object label are stored in the cloud. As it happens, the "*when CloudDB.DataChanged* "event is executed, even in the case of the user who just found the object. That is why it is necessary to control, not only the object, but also the player's username.

In addition, it is also necessary to check if the new object is the last one for any player, by modifying the *CloudDB.DataChanged* event. Fig. 19 shows a possible solution. In it, what has been done is to add a conditional in which the name of the object found is checked. As the order of the objects is always the same, if the name is not the last one (in this solution it is the *mouse*), we are in the case solved in the previous task. If it is the last one, the game must be ended, and the players must be notified that they have lost. In this case, the app should:

- Go to section3.
- Turn off the accept button.
- Write down on the label that the player has lost.
- Ask the player, using the *TextToSpeech* component, if he/she wants to play again.

In the solution provided in Fig. 19, we have created a function called *game_over* to encompass these four actions. Moreover, we decided to include the representation of the elapsed time (using the function already created in the TU4 *represent_time*), but it´s not mandatory. Then, this function is called within the conditional.

At this point, it is important to clarify to students that **the CloudDB component of App Inventor performs all the operations required for a reliable centralized communication between the players**. This component manages the communication protocol, and allows to use the same representation of objects used in the local version of the app. Moreover, it controls that the information exchange occurs in a reliable way, so all the players receive the information at the same time through the WIFI. As a consequence, it could seem to students that communications are very simple and easy to perform, but they are not, and all the technology behind this type of applications must be highlighted.

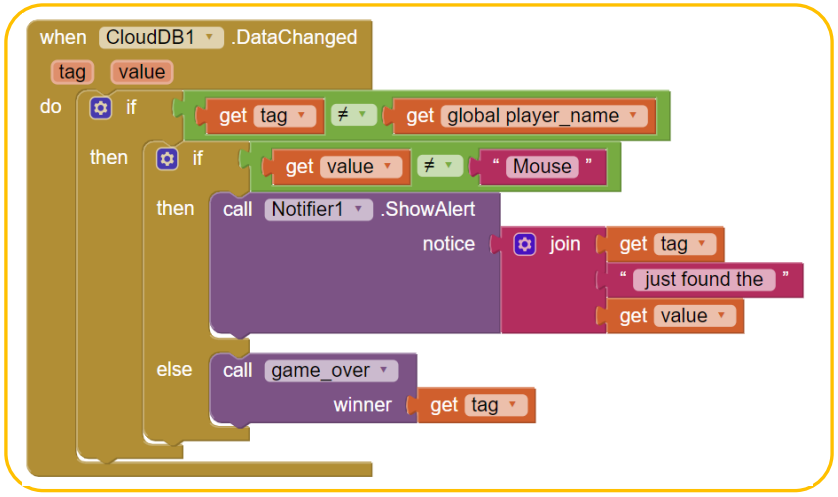


Fig. 19 CloudDB.DataChanged event with object control

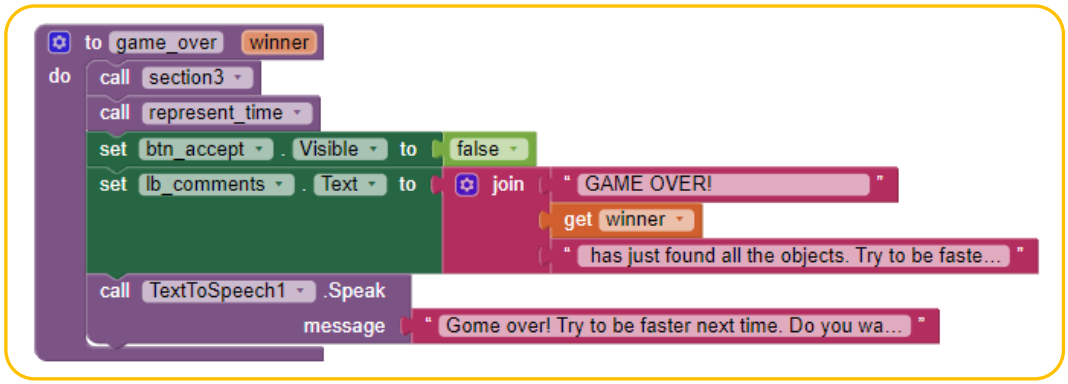


Fig. 20 Game_over function

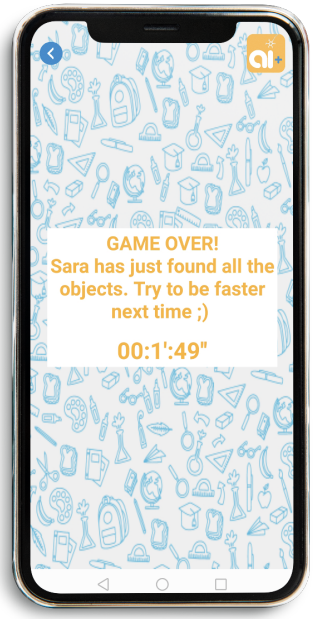


Fig. 21 Screenshot of the game over message

Activity 2

Send messages to the players warning them of the time that they are spending

45 Minutes

Activity 2 – Task 1

1

Send messages to the players warning them of the time that they are spending

45 minutes

Student´s work:

To complete the game, it has been decided to send all the players a message showing every time a minute passes. The idea is that as more time goes by, more alarming the notification should be, which implies to interact directly with the user. In this version of the game, only 3 objects have been added. As a consequence, we are going to limit these messages only to the first 3 minutes.

This should be programmed at the *when Clock1.Timer* event displayed in the below, so the first step will be to understand the programming of that block. This event is executed every second. The first thing to do is to check if the countdown or chronometer variable is set to true. The first condition is used to countdown at the start of the game, and the latter is used to time how long it takes for players to finish the game.

In the countdown part, it subtracts 1 from the number that appears in the label of section 1 called *lb_number* (the one you see on the screen just before starting the game) until that number is 1. At that moment, it means that the countdown is finished and:

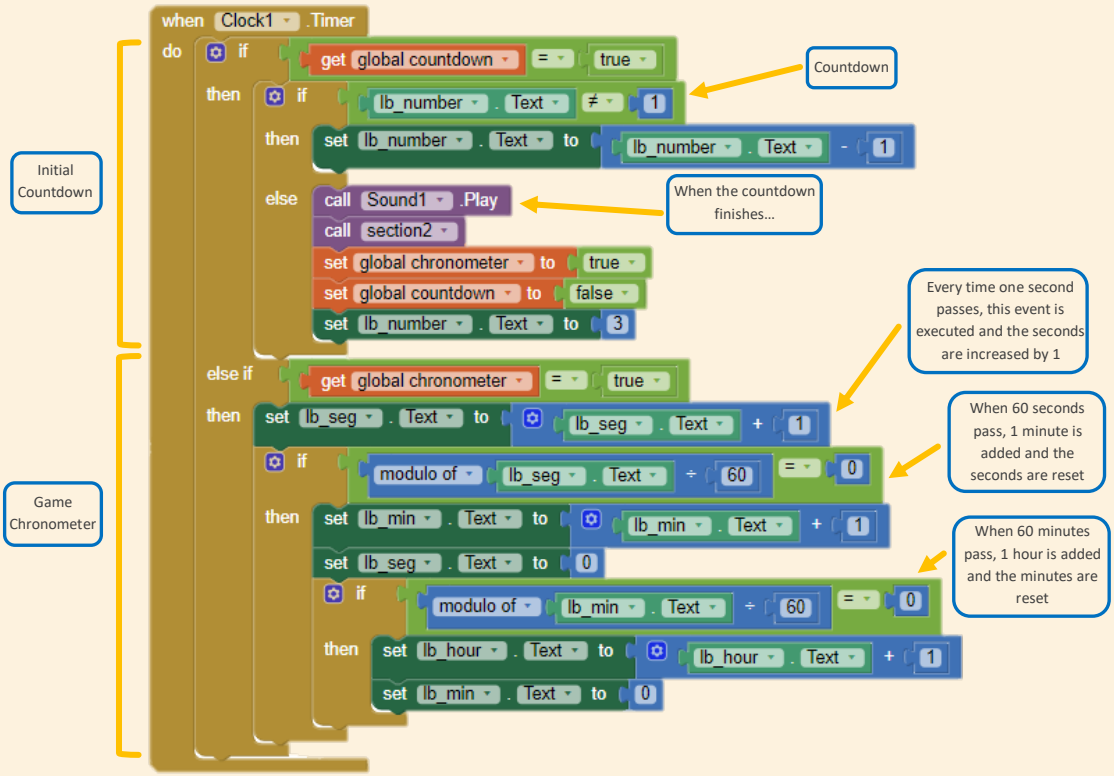
- Play a sound.
- Sets the countdown variable to false (because it is finished).
- Sets the chronometer variable to true (because the game starts).
- Activates section 2.
- Sets the label *lb_number* to number 3, to have it ready for the next countdown.

In the chronometer part, what it does is to update the *lb_sec*, *lb_min* and *lb_hour* labels. They will contain the seconds, minutes and hours that the player is spending.

Each execution of the event, i.e, each time a second passes, it adds 1 to the *lb_seg* label and checks if that label is 60 (dividing by 60 and checking if the remainder is 0). If it is not, it continues adding to the seconds, but if it is, it means that one minute has passed, so 1 is added to the *lb_min* label and the seconds are set to 0.

As with the seconds, when adding the minutes label, the number of minutes is checked to see if the number of minutes is 60. If it is, add 1 to the *lb_hour* and set the minutes to zero. If not, continue adding the seconds.

Its appearance is:



Knowing this, **you must think about how to modify this event to send three different messages in minutes 1, 2 and 3 of the game.**

Remember that you may need to create variables and you should use the Notifier component to show the alert.

45 minutes

Test:

Once the students have finished and tried out everything in this task, they must show the teacher that it works and **he/she will cover the points 10, 11, 12, 13 and 14 of the checklist in the Annex II: Teachers program test.**

Teachers' solution:

Initially, the timer looks like the one shown in Fig. 22. This event is executed every second. The first thing to do is to check if the countdown or chronometer variable is set to true. The former is used to countdown at the start of the game, and the latter is used to time how long it takes for players to finish the game. Therefore, the modification to be made will be within the second part, i.e., in the chronometer part.

Specifically, it will need to be included in the following conditional, which checks if *lb_seg* is 60, which would mean that one minute has passed.

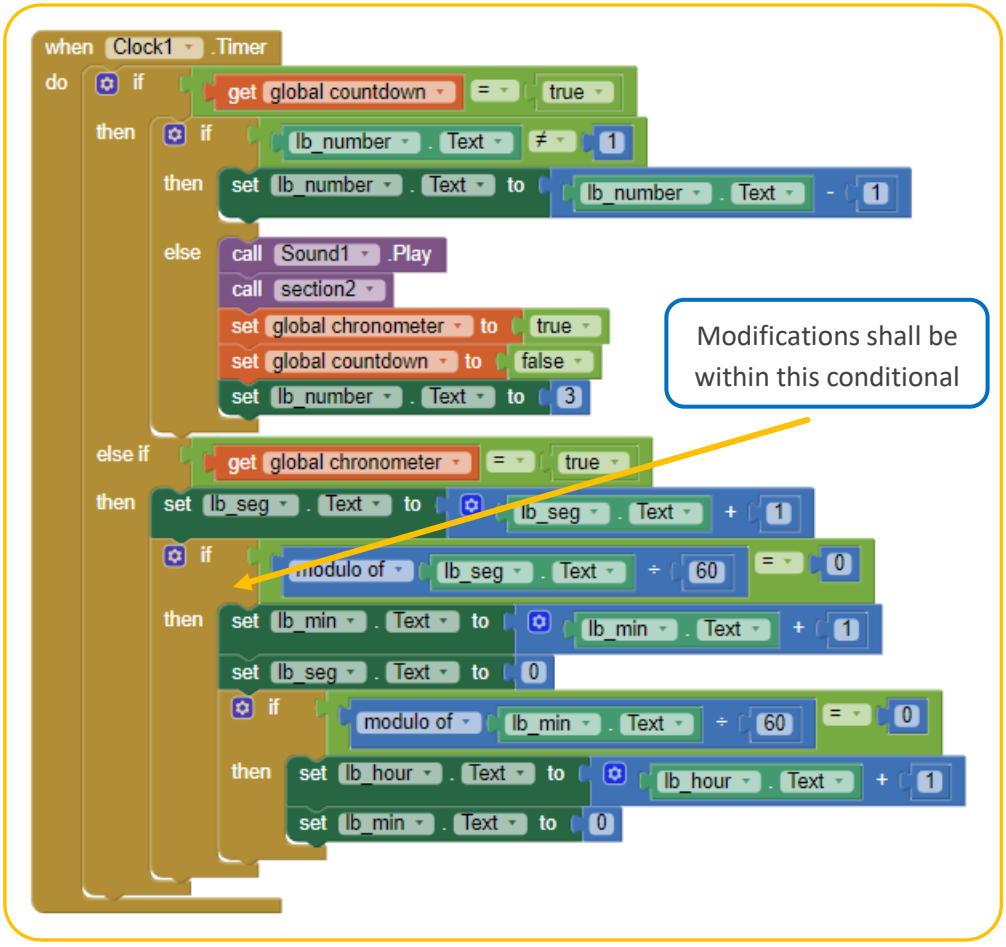


Fig. 22 Clock1.Timer event (without changes)

In this case, and due to the fact that the messages must be different for each minute, it has been decided to create a list type variable (Fig. 23), in which three messages are stored, making it very easy to increase the number of messages to be sent.

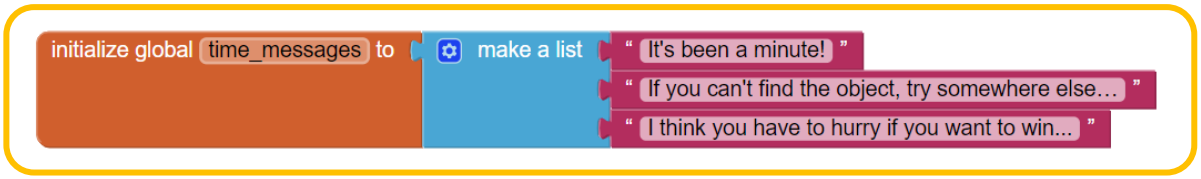


Fig. 23 Time_messeges global

Finally, to display the messages, the *Notifier* will be used, and the appropriate message from the list will be shown using the *select list item* block. This block contains two entries: the list and the index. The list will be the previously created one called *time_messages* and the index will be the minute it is in. The current minute is represented in the label called *lb_min*. Therefore, the index will be the text of that label.

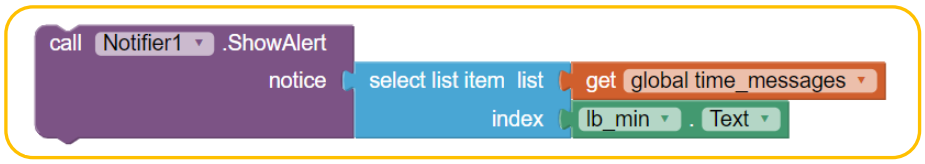


Fig. 24 Minute alert

This block must be within the mentioned conditional. It must be controlled that it is not executed if more minutes than messages have passed. This is because, if that happens, the program would be trying to access to a position of the variable which does not exist, and it would return an error. Consequently, it is necessary to include a conditional in which the condition is that the minutes are lower than or equal to 3, as shown in the following figure.

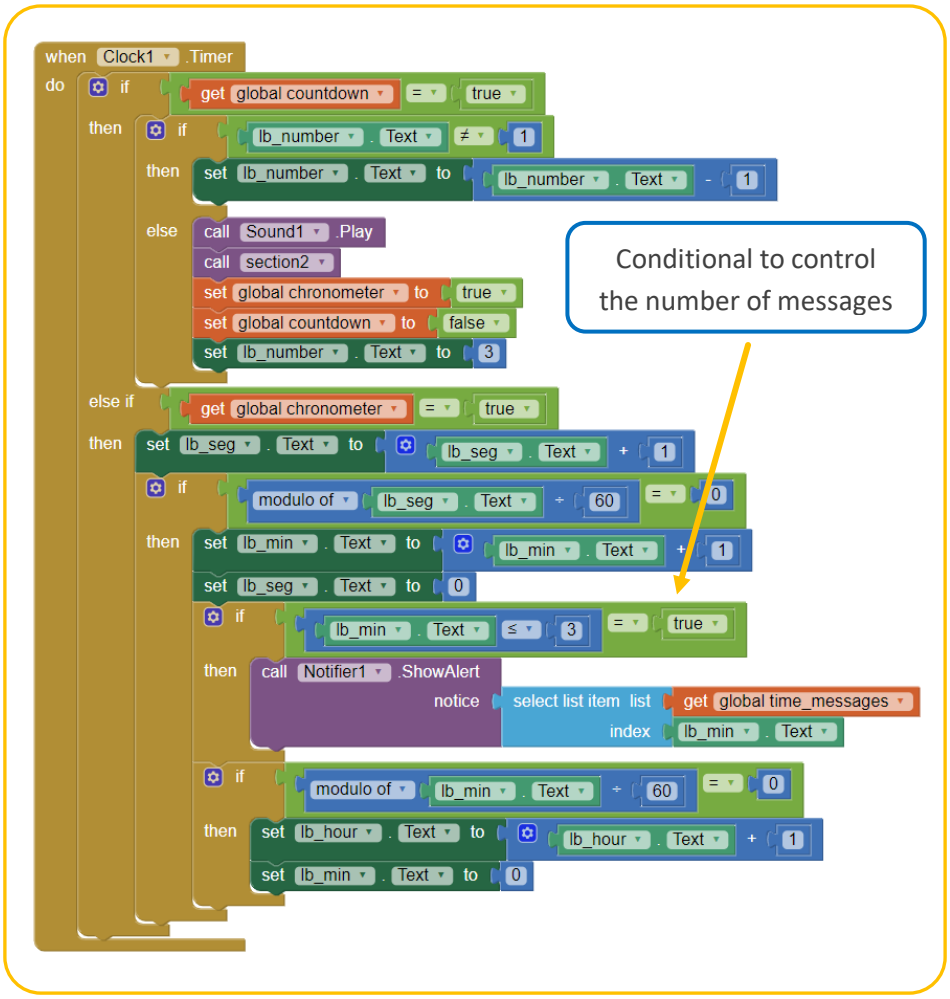


Fig. 25 Clock1.Timer event final aspect

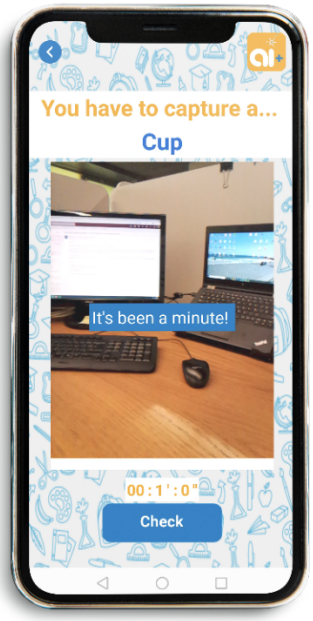


Fig. 26 Screenshot of the spent time message

10.3 FINAL REMARKS

Once the app has been completed, it is time to point out the main AI concepts the students have practiced in the TU.

Concerning perception, the app uses the following sensors:

- The camera, to capture the objects.
- The microphone, to pick up the user's speech.
- The touch screen, where the user can click on buttons.
- The keyboard, to input the username.

Regarding actuation, the app includes:

- The LCD screen, on which the game information is displayed.
- The speaker, communicating the end of the game.
- The WIFI antenna, communicating the objects and name through the WIFI.

In terms of representation, apart from the sensorial inputs, this app uses:

- A representation of the captured objects as text labels
- A representation of the usernames as text labels

In terms of reasoning, the app is very simple, and only a comparison between times is used to send notifications to the players and to establish the game winner.

Regarding learning, students have used the image classification model created in TU4, based on an ANN.

Finally, the main topic treated in the TU has been that of artificial collective intelligence, and students have seen:

- How to create a **centralized** ACI system based on a cloud database
- The relevance of having a software component that creates the communication protocol and manages the information exchange
- The potential of using collective information in order to develop more powerful apps

11. EVALUATION

As established in the introductory TU, three evaluation methodologies are proposed to this TU. The score of the will be calculated on 100 and the weights of each of the evaluation methodologies will be:

Assessable activity	Score
1- Teacher program test	65
2- Student's video	20
3- Individual rubric	15
TOTAL:	100

1. *Final test of the program:* in this TU, we propose that the test of the program is carried out in this case only by the teacher. As the students were doing each of the tasks, it was proposed to the teacher to check the proper functioning of the app following the checklist in the **Annex II: Teachers program test.** Each of the points present in the checklist (filled in between the tests of each task) must be added up, and the final evaluation of this part will be obtained (maximum 65).

   In addition to the direct program test, all **students must submit the programming code of their solution**, so the teacher could test it, if required. As this is not a programming curriculum, the evaluation emphasis will not be on the code quality, but in the previous points.

2. *Final test of theoretical concepts*: we propose that students create a short video performing a sort of app review, where they should explain how it works thinking that it could be watched by an external user. The main objective is that students explain, not only the game functioning, but the AI features it contains in terms of perception, interaction with the user, learning and collective communications. In this last topic, students should provide details about how their game uses the information of other players, and why this is relevant.

   This video has a maximum score of 20 and should be evaluated by the teacher taking into account not only its appearance but also what it should contain and the quality of the contain.

3. *Ongoing work during the TU*. This methodology is very important, and it will be evaluated using an individual rubric that the teacher must fill for every activity (**Annex I: Teacher rubrics**). The work of each of the students will be evaluated under this rubric. There are a total of 5 fields in this rubric and each of them will be worth a maximum of 6 points (the score that each student will take depending on the level he

or she achieves is established in the table). The student who achieves the expert level of each of them will reach the maximum score for this part: 15 points.

12. COMPLEMENTARY ACTIVITIES

The following improvements to the program are proposed, mainly for groups that finish the main goal before others, or for teachers that consider that an extra evaluation can be proposed to students:

1. GETTING THE GAME STARTED FOR ALL PLAYERS AT ONCE

Nowadays all group games start at the same time. This complicates the programming a bit but it is possible to do it with the knowledge the students have at this point. The idea would be that one of the users would create a session and the rest would join in by means of a code, names, etc. It will be a great improvement for the game!

2. ADD A LIST OF RESULTS

Once the game is over it would be very interesting to have a final ranking of results. A ranking in which the data of the fastest players' times would be reflected.

3. ADD A LIST OF PLAYERS

It would be very useful if section2 of screen2 showed the names of the players online. To do this, it would be necessary to modify the appearance of the game but this is simple in terms of programming.

13. ANNEX

ANNEX I: TEACHER RUBRICS

Level (score) / Aspects to be evaluated	Expert (3 points)	Competent (2 points)	Partially competent (1 points)	Not yet competent (0 points)
Adequate selection of information (Videos, App Inventor user manual…)				
Time management (the student is aware of deadlines and progress…)				
Design and construction of the solution (goal understanding and reliability of the program)				
Creativity (autonomy and improvement of the basic solution)				
Teamwork (organization)				

ANNEX II: TEACHERS PROGRAM TEST

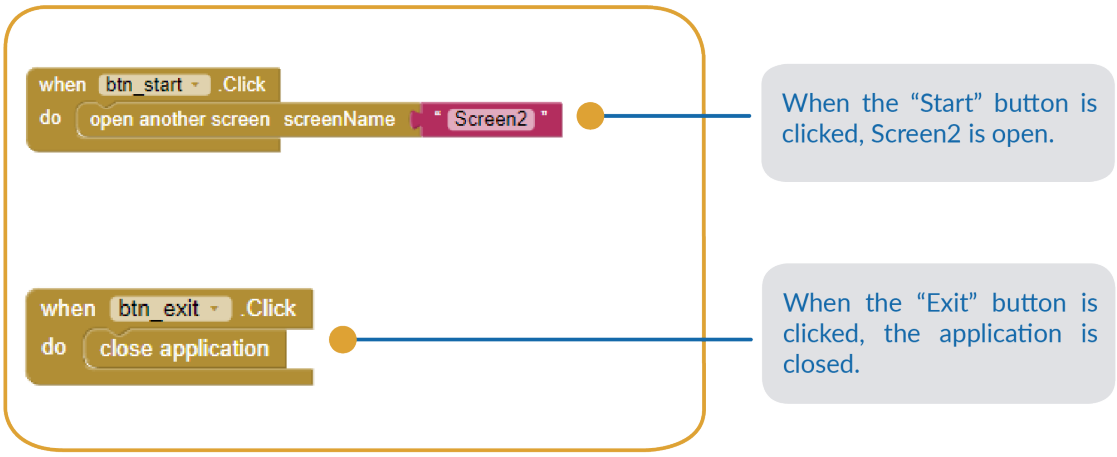
The following table contains a checklist for the teacher to evaluate the operational part of the application. Tick yes or no as appropriate with an x.

	Level (score) / Aspects to be evaluated	Score	Yes (total score)	No (0 points)
1	The format of the new module is correct, within a vertical displacement (check it in the design part of app inventor)	4		
2	The design has a good appearance (font, colours, sizes...)	4		
3	When you click start without entering the name, the game does not start and a warning message is displayed	4		
4	When you click start after having entered the name the game starts correctly	5		
5	The application notifies users when one of their opponents finds an object	5		
6	The messages are easy to interpret by the players and contain the appropriate information (name of the player and object found)	4		
7	When the object found is the last one, the game is over for all players	5		
8	The accept button does not appear at the end of the game below the text	4		
9	Check in App Inventor if a function has been created for when the player loses that includes the necessary calls (section3, turn off accept button, write on the label and ask the user if he wants to play again using the TextToSpeech component)	4		
10	When the minutes are up, all players are notified	4		
11	The messages are different depending on the minute that has passed	4		
12	The programming checks that the number of minutes is less than the number of messages available and therefore never gives an error when accessing the list	5		
13	The human-machine interaction aspects are correct: clear and direct sentences	5		
14	The general app functioning is correct, with no stops or pauses	8		

ANNEX III: PRE-PROGRAMMED BLOCKS

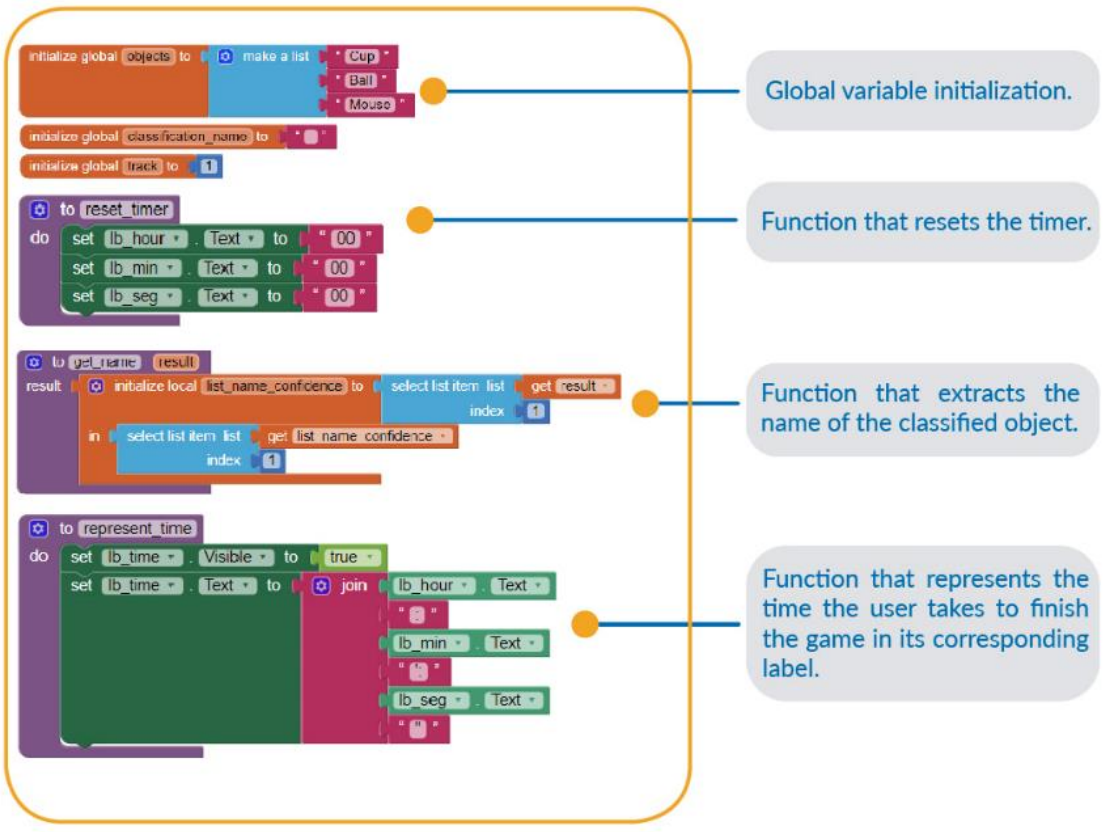
- **SCREEN1:**

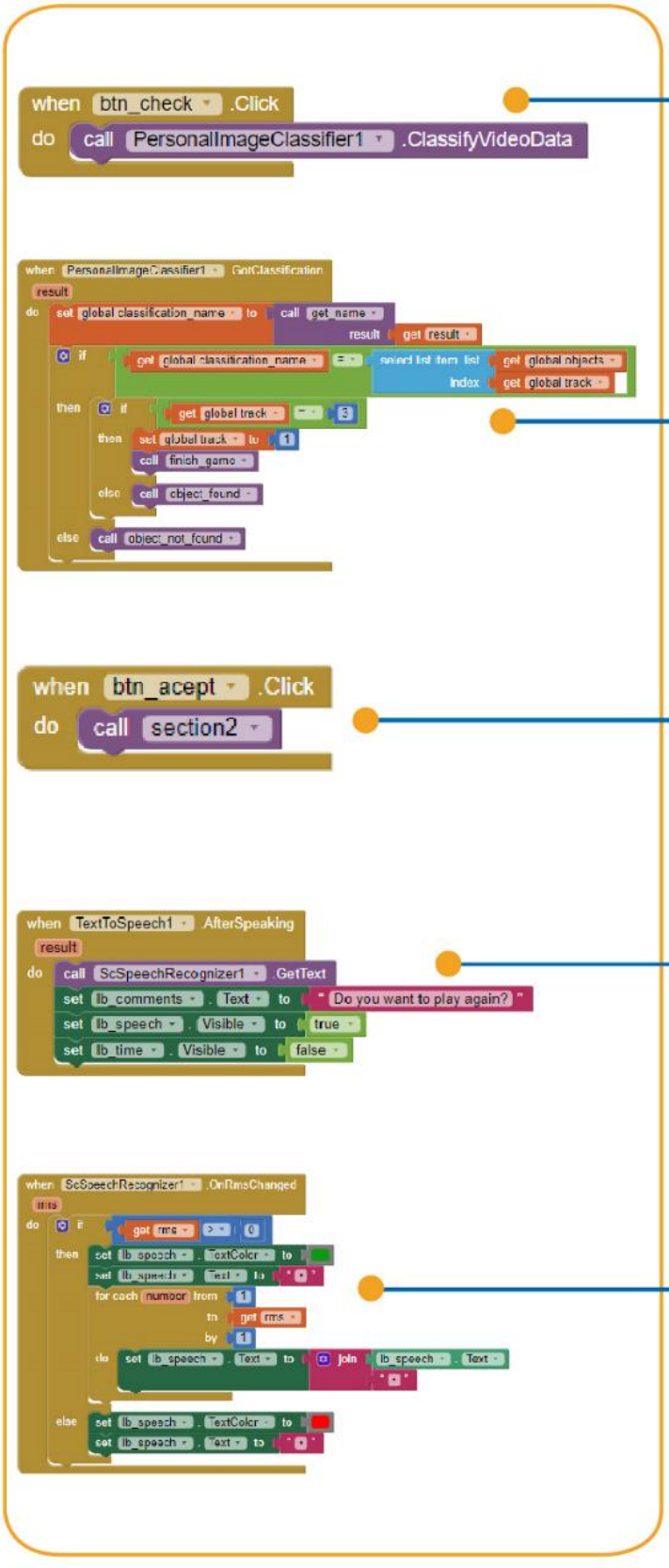
In this first window, all the blocks are completely programmed.



- **SCREEN2:**

In this window, the blocks that should not be modified will be:





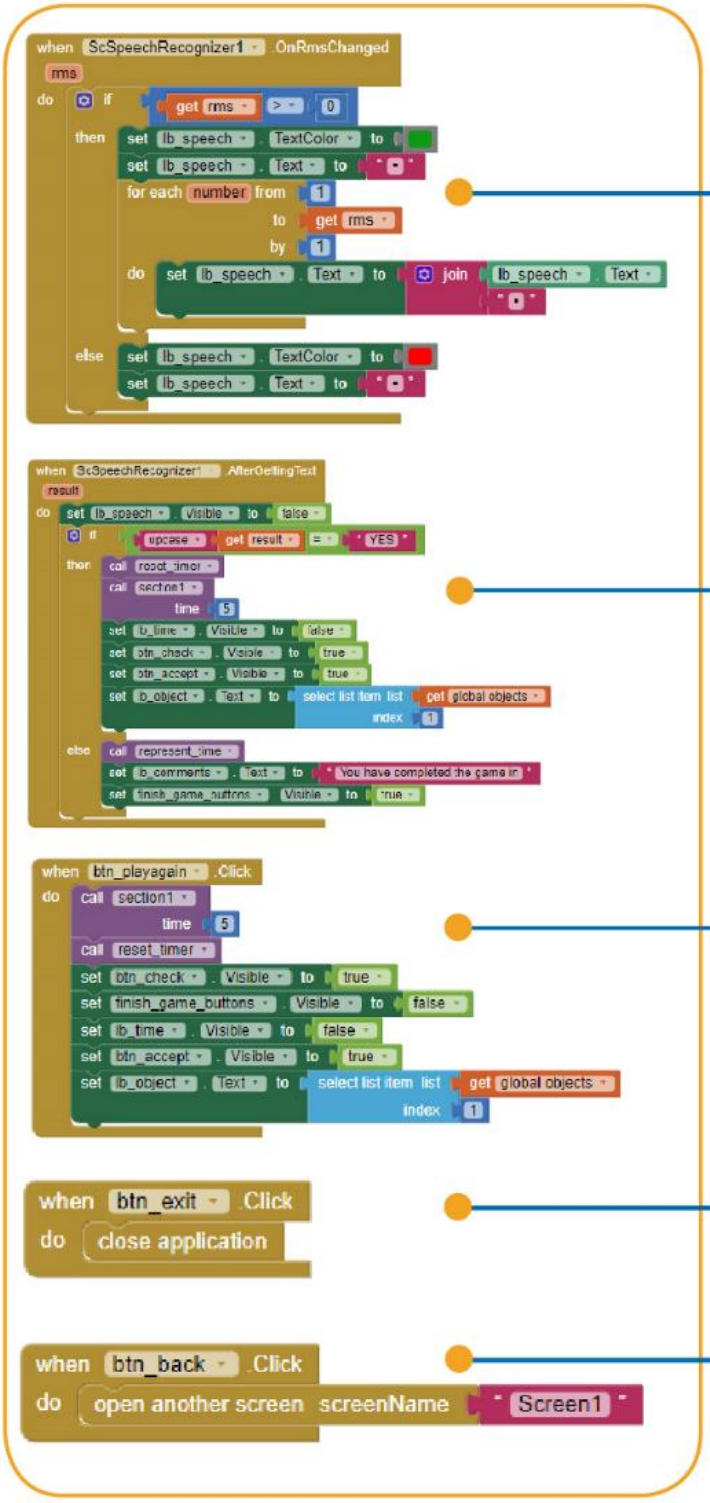
When the user clicks the check button, the classifier is called.

When the classification has been performed, it is checked whether the object found is the correct one or not (and whether it is the last one or not) and the corresponding function is called..

When the "Acept" button is clicked, section 2 is displayed on the smartphone screen.

This event, when the voice message has already been said, calls the voice recognition, shows the message on the screen and activates/deactivates the necessary labels.

Programming the speech recognition part to display a green or red bar depending on if any sounds are detected or not.



Programming the speech recognition part to display a green or red bar depending on if any sounds are detected or not.

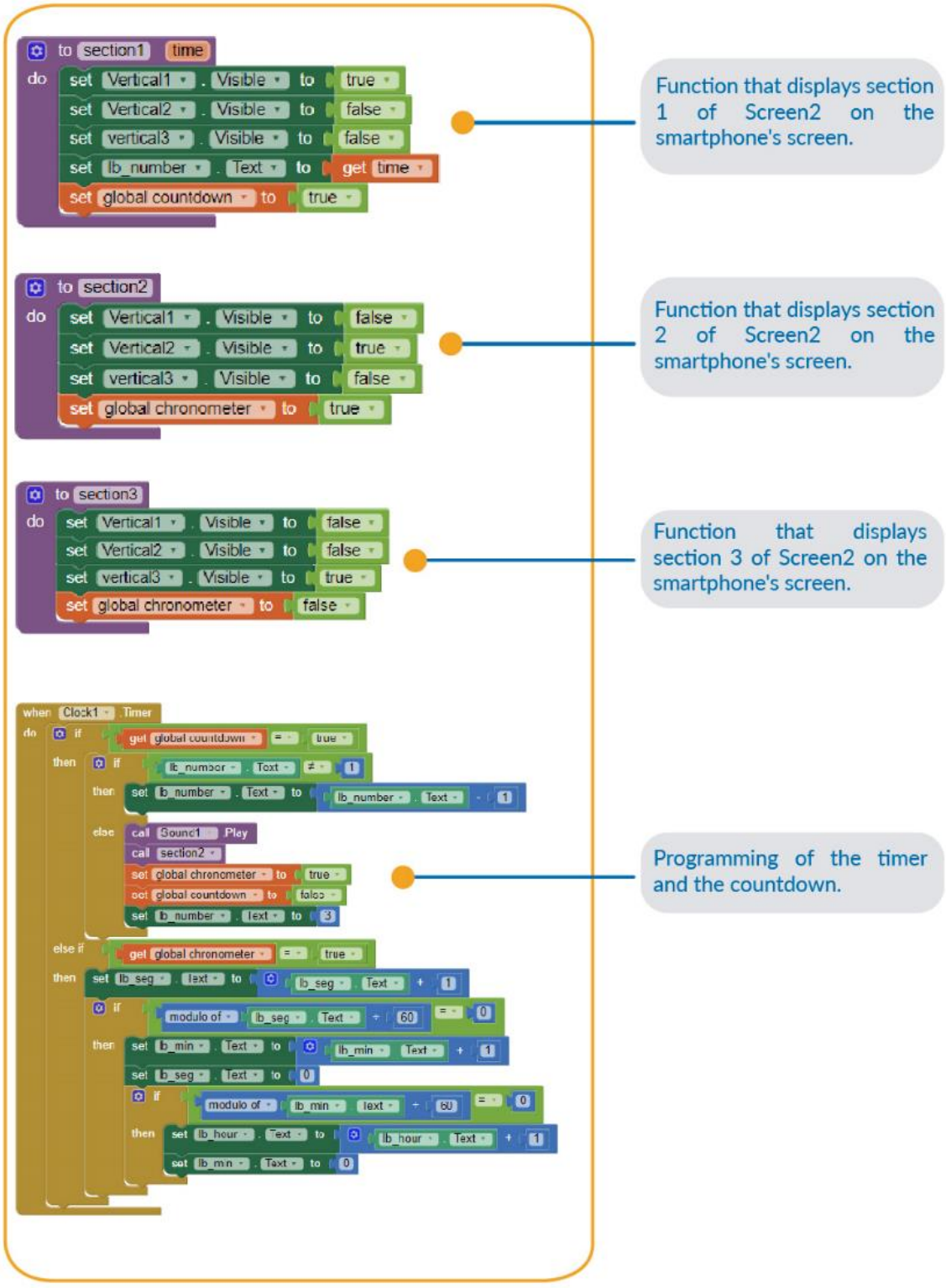
Programming of the voice recognition part to start the game again or not, once the user's answer is received.

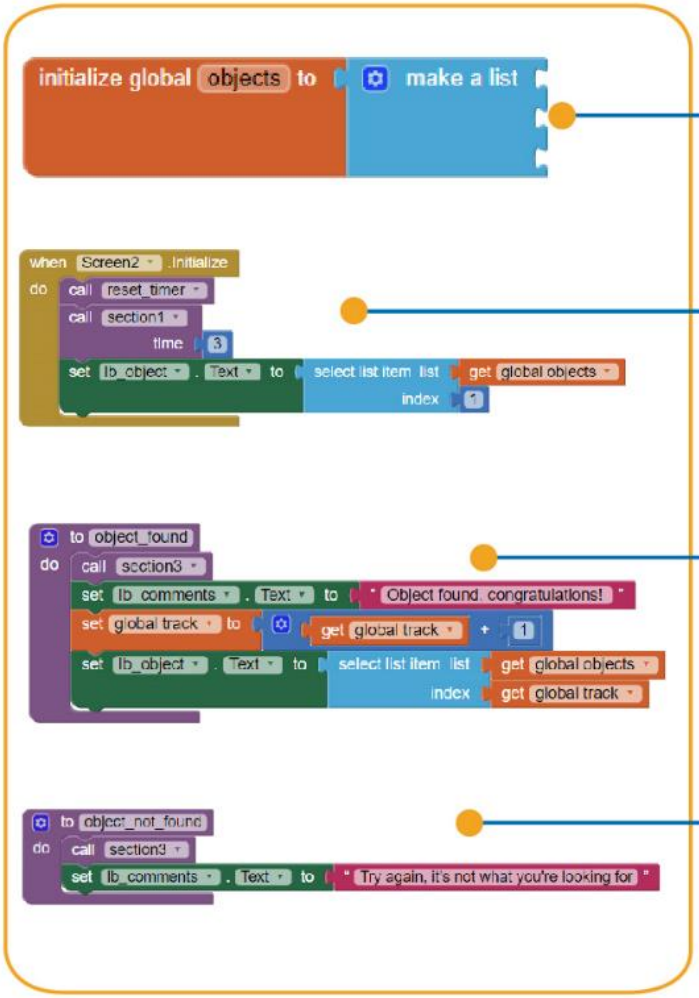
When the "Play again" button is pressed, the countdown is displayed, the timer is reset, and the necessary buttons are shown.

When the "Exit" button is clicked, the application is closed.

When the "Back" button is clicked, the Screen1 is open.

The blocks that should be modified will be:





Initialization of the objects variable.

When the Screen2 is initialized, the timer is reset, and the section 1 the object to be found are represented in the smartphone screen.

Function that displays section 3 of Screen2 on the smartphone's screen.

Function that tells the user that the object has not been found.