TEACHING UNIT 4

Introduction to Learning in AI

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

Learning is a key property of any intelligent system, which must show improved skills obtained from experience. This way, situations that were not previously seen or programmed can be addressed, and even anticipated. In the scope of AI, learning is performed through algorithms that run on a computer-based system, which create mathematical models from data. All these algorithms are included in the field of **Machine Learning**, where three main types of techniques are considered: **supervised learning**, **unsupervised learning** and **reinforcement learning**. Their main difference is the information the designer provides to them in order to guide learning. As a result of the execution of any machine learning algorithm, the computational model that arises can provide output data for any set of input data, in fact, for cases typically different from those used during the learning stage.

The mathematical models are learned from data, so data must be adequate to allow for learning: they must be enough in quantity, representative of the cases to learn, and good in quality. In order to achieve these requirements, in the machine learning field there are many computational techniques dealing with **data preparation**, which must be taken as a basic part of this topic too.

Finally, once the model has been learned, it must be tested to verify if it provides successful results, that is, if it is able to predict from previously unseen cases. This is the last sub-topic of machine learning we will point out here, and can be called **model validation or testing**.

In this first TU dedicated to learning, students will practice with a ***supervised learning*** algorithm to solve a problem of image classification. In addition, they will have a first experience on the relevance of a proper data preparation for learning, and on how to analyze the results provided by the resulting model. Specifically, the challenge they will solve consists on developing a smartphone app that implements a **Scavenger Hunt**, that is, a group game where the participants must search for a series of objects in the real world. The app will include a mathematical model previously learned by students, which will be able to recognize the correct objects from a photo taken with the smartphone's camera. This challenge will be completed in the following TU by adding it collective intelligence features.

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 and loops. 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:** fundamentals of algebra and functions. It is very important that students understand the concept of mathematical model, and the relation between

inputs and outputs. We recommend that teachers revise such concepts with them, for instance, using the following references:

- https://youtu.be/52tpYl2tTqk
- https://bit.ly/30v16uk

- **Perception and actuation in AI:** it is mandatory that students have finished TU2 and they know the basics of perception and actuation in AI.
- **Representation in AI:** it is mandatory that students have finished TU3 and they know the basics of representation in AI, mainly in terms of images representation.
- **Previous knowledge:** it is recommendable that students watch the following videos before starting the TU, to get an initial idea of the fundamentals of machine learning:
  - https://youtu.be/Wm1Id-vEX3U
  - https://youtu.be/z-EtmaFJieY

3. Learning Objectives

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

Specific	• Understanding of the concept of model learning. • Why data preparation is important in machine learning. • Basic working of supervised learning. • What is an Artificial Neural Network. • Importance of model validation.
Transversal	• App Inventor software usage. • Computational thinking: variables, conditionals, lists and functions. • App interface design for computer games.

4. Contents

The following specific contents of machine learning will be studied in this TU:

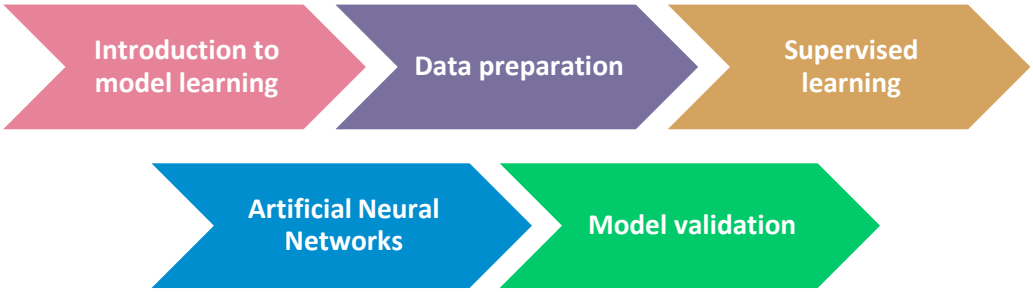


Fig. 1 Specific contents

5. TIMELINE

This TU is the fourth of six focused on introduction to AI. Remember that, after the completion of the previous TUs, students have seen the fundamentals of perception, actuation, representation and reasoning on AI, which are required to properly understand how machine learning works. The next TU will be focused on collective intelligence, and it will develop an improvement of the smartphone app created in this TU.

The estimated time for this TU is **6 teaching hours**.

6. NECESSARY RESOURCES

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

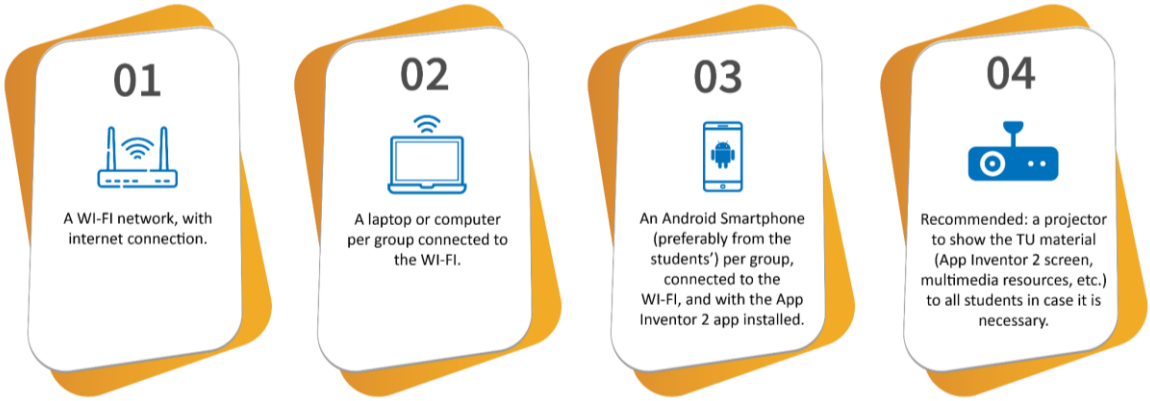


Fig. 2 Hardware elements required to carry out this TU

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

1. Every group must have an active App Inventor account.
2. *Template app:* An App Inventor project template is provided with the TU to focus the attention on AI aspects and not in programming ones. This template includes the graphic part of the app, and some pre-programmed blocks. It is named "appInventor_template_TU4.aia" and it must be stored in the computer of each group to be loaded from the App Inventor web application. For a better understanding of the template, *Annex I: Pre-programmed blocks,* explains all the pre-programmed blocks used in the application.

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.

7. BIBLIOGRAPHY

- App Inventor documentation: https://bit.ly/2Wt7Khr
- Personal Image classifier: https://bit.ly/3ialHtQ
- Teachable machine: https://bit.ly/3fuaClq
- Supervised machine learning: https://bit.ly/2PsyNpM

8. GROUPS

It is recommended to organize the students in groups of 2. 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 in 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. UNIT MATERIAL

This unit has extra documents, which are available at:

https://drive.google.com/drive/folders/1lohSLWg8yRsZQEiM2X2oW6Ne_hDYqI6M?usp=sharing

These documents are:

- Document with Teaching Unit 4 (TU2_Learning.pdf)
- Video of the app functioning (TU4_APP.mov)
- App Inventor solution for teachers (appInventor_solution_TU4.aia)
- App Inventor template for the student (appInventor_template_TU4.aia)
- Unit feedback document for teachers (teacher_feedback_TU4.docx)
- Student's program test in .docx format. (student_program_test_TU4.docx)

10. CHALLENGE / PROJECT

10.1 FINAL OBJECTIVE

WHAT TO DO?

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

HOW IT SHOULD WORK?

Watch the video "TU4_APP.mp4" included in the TU resources to understand how the app should work. Each group could implement their variations to this example to improve the game, but the overall functioning must be maintained.

The aim of the game is that the user finds the object that is shown in the screen in the shortest time, and when it is found, he/she must take a picture using the smartphone's camera. The app recognizes the object using a previously learned model and notifies the user if it is the correct one or not. If it is not, the user must continue searching for it, while if it is the right one, a new object is proposed. When all the objects are found, the app will inform of the total time used to overcome the challenge, and asks if he wants to play again.

It is not a fully smartphone game, because the game itself takes place in the real world, where the objects must be placed and discovered. But the app must manage the overall functioning of the game, controlling the correct realization and finalization. We will call this app *Capture it,* and it will be completed in the next TU by adding it collective features.

From a machine learning teaching perspective, the model used in the app to recognize the objects with the camera is created by the students. Hence, the game can be adjusted to the specific objects they have in their school.

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. Be developed in the App Inventor platform.
2. Contain a mathematical model that recognizes at least 3 objects from pictures taken with smartphone's camera.
3. Create the model using supervised learning.
4. Show a timer on screen while the user is playing.
5. Show the object to be found on screen (at the beginning and every time an object is found).
6. Have a viewfinder to capture pictures.
7. Have a button or similar to check whether the captured object is correct or not.
8. Show a message indicating if the captured object is correct or not.
9. Control that all objects are found and show a message with the time spent.
10. Reproduce a voice message when the game is over and ask the player if he/she wants to play again.
11. Perform voice recognition to capture player's answer to play again.
12. Restart or finish the game according to the user's answer.

10.2 ACTIVITIES

NOTE: This section (Activities) is aimed at teachers whose students complete the TU following the provided template. 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 follow the template, 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 STEAM 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.

Therefore, as shown in the time organization section, the project is divided into two activities, one for each topic, and the first one implies two different tasks. Fig. 3 displays a diagram that must be explained to students, and which summarizes the steps that must be followed to develop the program.

In order to make it easier for students to understand how the application will be developed, they should be shown the following figure, which summarizes the steps to be followed.

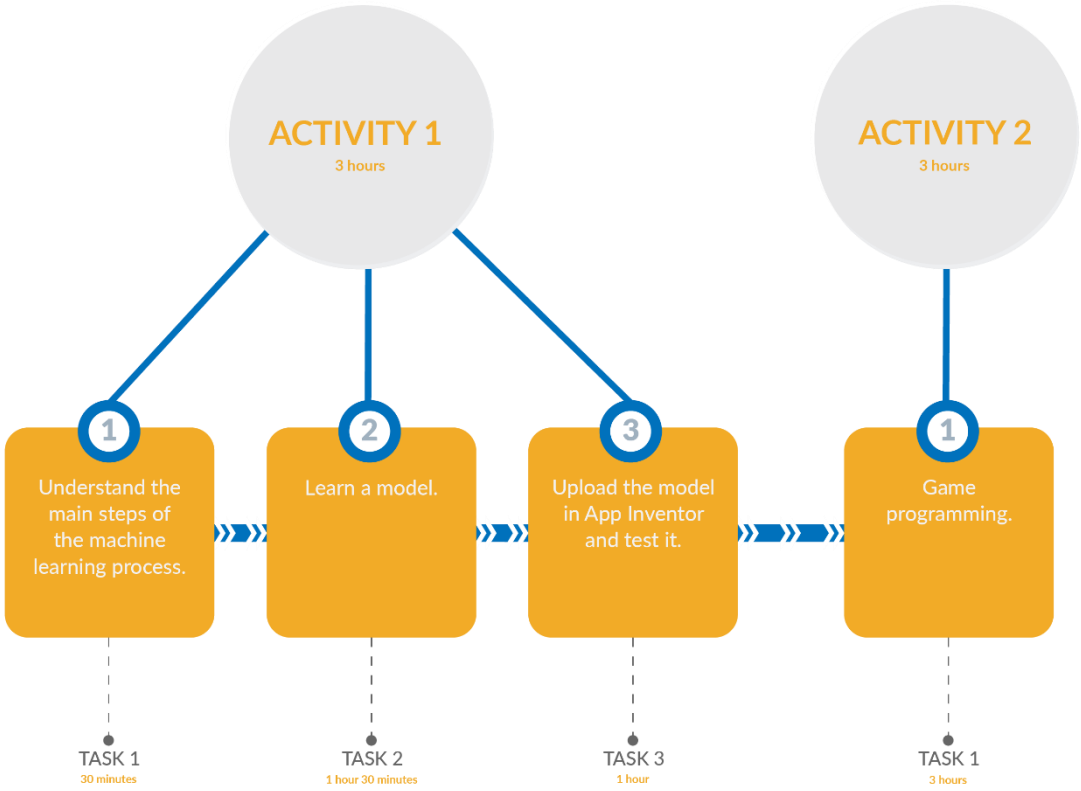


Fig. 3 Flowchart of the application development process

Activity 1:

Learn a computational model that performs image recognition

3 Hours

Activity 1 – Task 1

1

Understand the main steps of the machine learning process.

30 minutes

Required knowledge to be introduced to students:

Before starting, students should have watched the videos recommended in section 2. From them, and highlighting that the goal is to create a model that recognizes specific objects from a picture, it is proposed to **perform a short discussion** in the classroom, in which all the groups give their opinion about how they think this learning should be carried out.

After this initial contact, teachers should provide students with a brief introduction to machine learning, specifically, to **supervised learning**. We recommend the following references, although many others can be used:

- Jonathan S. Walker. 2018. *Machine Learning for Beginners: Your Ultimate Guide To Machine Learning For Absolute Beginners.* CreateSpace Independent Publishing Platform, North Charleston, SC, USA.
- Supervised machine learning: https://bit.ly/2PsyNpM
- Introducing machine learning to kids: https://bit.ly/31oYDRn
- Training data and bias: https://youtu.be/x2mRoFNm22g
- Neural Networks: https://youtu.be/JrXazCEACVo

Teachers must clarify to students that adjusting a function to data in algebra is a very similar concept to learning a model we are introducing here. The main difference is that in machine learning we use powerful computational algorithms to perform such adjustment, so more complex functions can be obtained.

From a practical point of view, students should have clear that, to obtain a machine learning model, the following 4 steps must be performed:

Data preparation

Every machine learning algorithm requires a set of representative data from the problem we aim to generalize. In the case of supervised algorithms, the data contains inputs (problem variables) and the corresponding target outputs (desired values), as explained in the previous references.

Model training

There are different computational models in supervised learning that can be learned, and which generalize the data. In all the cases, these models must be "trained", that is, it is required to execute an algorithm in the computer to adjust the model parameters until they predict the target data correctly, or until a maximum number of learning steps is achieved. There are also many training algorithms, which normally depend on the selected computational model, but it is out of the scope of this TU to review them. In fact, we will use just one of the most representative models, an Artificial Neural Network, which will be trained using a "gradient descent algorithm", but students do not have to implement them, they just need to understand how the learning process works. Model training is a computational process that can take a long time, depending on the model complexity and on the size of the dataset.

Model testing

Once trained, it is necessary to test the model with a different set of images from those of the training set, because it is typical in machine learning that the model does not generalize properly. This normally happens when the dataset size is not enough for the problem complexity, or when the number of training steps is low. The programmer has to validate the model reliability, and if it is not adequate for prediction, it is necessary to return to step 1 and add more data, or to step 2 and increase the number of learning steps.

Model usage

Once the model has been tested, it is ready to be used. It implies that the final values of the parameters must be stored in the computer, so the computational model can be executed every time a new input data is obtained. It is very important to understand that the model execution is very fast, because it does not require to train the parameters again, only to use the final values in a single shot run.

In this TU, the model we are going to train uses, as input data, the images of the objects to recognize, and as targets, the labels containing the object name. In the machine learning terminology, we are dealing with an **Image Classification** problem. We recommend teachers to show students the following presentation in order to understand the basics of this scope: https://bit.ly/3kfPhzZ.

NOTE: at the end of the previous video presentation, the Teachable Machine web tool is proposed to train the model, but ***we will not use*** such tool here, although it can be useful for the students to see the explanatory video of that web to better understand image classification.

The whole learning process we will follow in this activity 1 is based on the **Personal Image Classifier** provided by App Inventor. In this link, teachers could find detailed information regarding all the possibilities of this tool: https://bit.ly/3ialHtQ.

Student´s work:

In this task, the only work to be performed is to attend teacher's explanation of the required knowledge, including to watch the recommended videos.

30 minutes

Test:

To evaluate the understanding of the concepts presented in this task, we propose to repeat the same **short discussion** as that of the beginning of the class, in which all the groups compare their initial opinion about how they think this learning should be carried out with the explained steps.

Activity 1 – Task 2

2

Learn a model

1 Hour and 30 minutes

Required knowledge to be introduced to students:

To learn the model that must recognize the objects and to use it in App Inventor, we will use the Personal Image Classifier (PIC) extension introduced above, which has been already included in the project template. The first step students must take is to **decide the objects** that will be used in their own game version. In this case, to simplify the first steps, it is proposed to start only with three objects. We recommend that students choose particular objects that are not easily available in all the classrooms, so that the player is forced to go around the school. Consider that the algorithm can learn any type of object, but students have to take the pictures for training, so the objects must be physically in the school. In addition, they should not be very big, because

one of the most important things in this searching game is how to hide them. For instance, students could select objects like fruits (apple, banana, ...), cups, basketball balls or others they can bring from their houses.

Once each group has selected the objects, the model learning can start. The PIC component web tool that will be used is https://bit.ly/3frwo9D. All groups must enter on this web, which looks as shown in Fig. 5 once loaded.

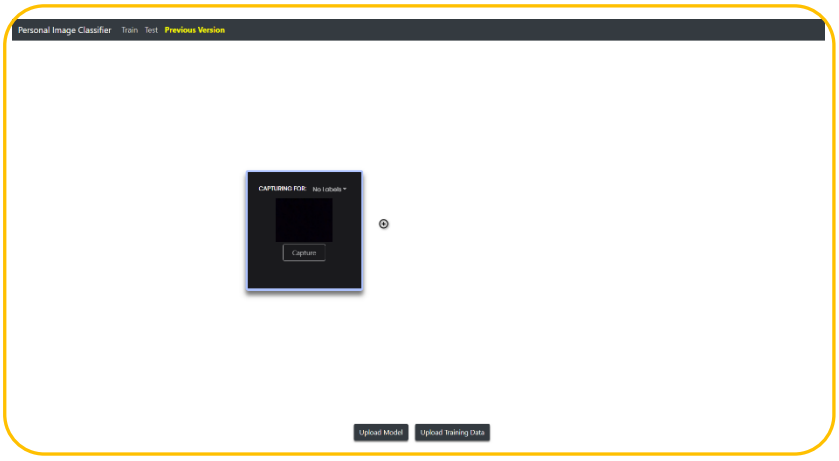


Fig. 4 Initial appearance of the PIC component web tool

NOTE: in the last version of the personal image classifier, it is possible only to create tags and upload images from the webcam. But, in the solution provided for this TU, we will use the previous version of this extension (which can be loaded in the upper left corner of the screen), which allows to use images previously taken with the smartphone's camera. If any teacher prefers to use the new version, the following steps must be adapted to use the webcam.

The appearance of this old version is as shown in the Fig. 5.

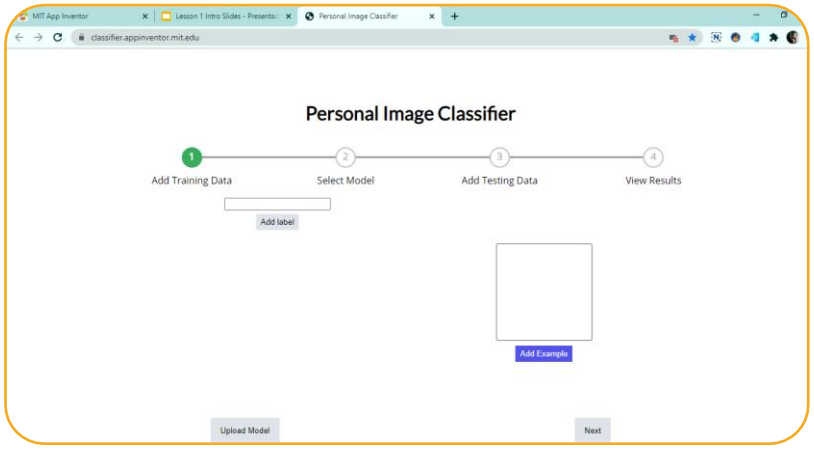


Fig. 5 Initial appearance of the PIC component web tool

In what follows, the four steps displayed in the previous figure are explained:

Add Training Data (Data preparation)

In this case, the input data are pictures of the selected objects and the target output are text labels with the object name. As shown in Fig. 5, in this screen there is a button called Add label and, just above, a text box. In it, each group has to enter the name of each of the selected objects. It is important that students remember exactly the names, because they will be used later in the code. Fig. 6 displays the appearance of the page after entering the names Cup, Ball and Mouse.

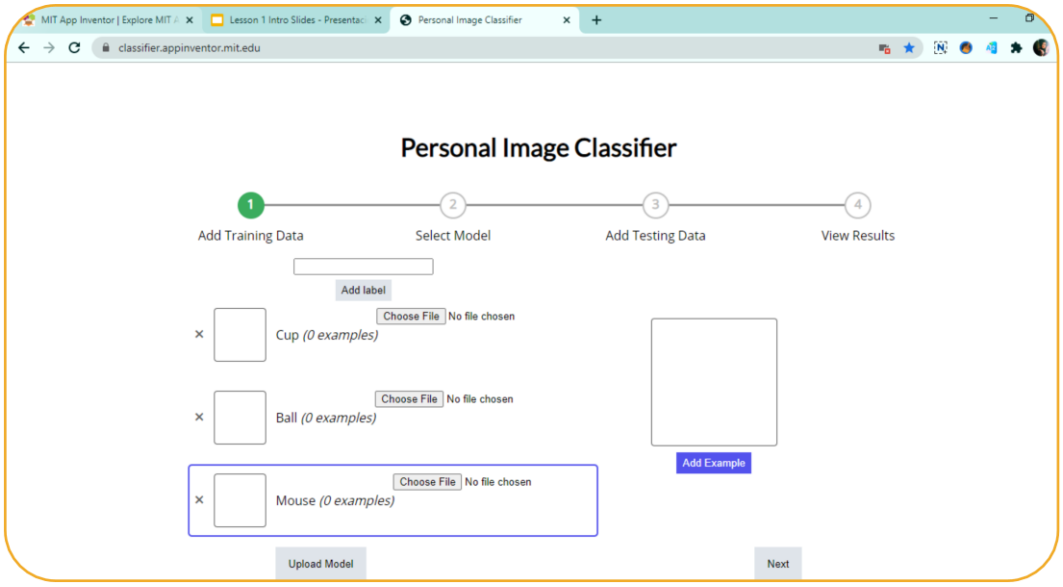


Fig. 6 Appearance of the PIC component web tool after entering the objects name

Now it is necessary to upload several pictures corresponding to each label. To make the recognition simpler, it is recommended to upload images of different objects of the same type. For instance, if the selected object is a basketball ball, recognition will be more reliable if the training images correspond to different balls (colour, brand, etc). In the screen show in Fig. 6, there are two possibilities to upload pictures. The first one is to capture them from the computer webcam, which could be complicated to achieve a representative set. The second is to use pictures previously taken or downloaded from internet. We recommend that students take pictures from the objects using the smartphone's camera, and then transfer them to the computer. It is suggested to upload about 10 images (as in Fig. 7) for each label but, as commented above, the effectiveness of the model is not only related to the number of photos, but also their quality. In this sense, students should take pictures from different perspectives of the object, and where the object can be easily distinguished from the background. Fig. 8 shows an example of 10 pictures from a cup that could be adequate for learning.

Add Training Data (Data preparation)

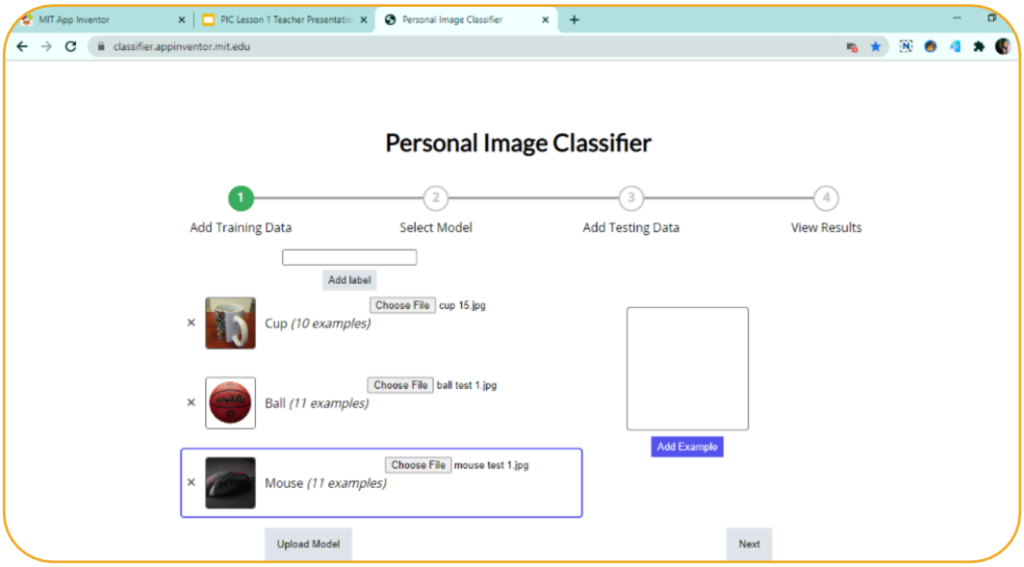


Fig. 7 Appearance of the PIC component web tool after entering the input data

Fig. 8 Examples of cup images suitable for learning

Select Model (Model training)

In this case, an Artificial Neural Network (ANN) model has been selected, which has been trained using a gradient descent algorithm called *Adam*. Although it is out of the scope of this curriculum to get into the details of these specific topics in machine learning because deep understanding requires a higher level in mathematics, we think that the basics of ANNs could be introduced to students. For instance, we recommend watching with them, the following video (or a similar one): https://youtu.be/bfmFfD2RIcg

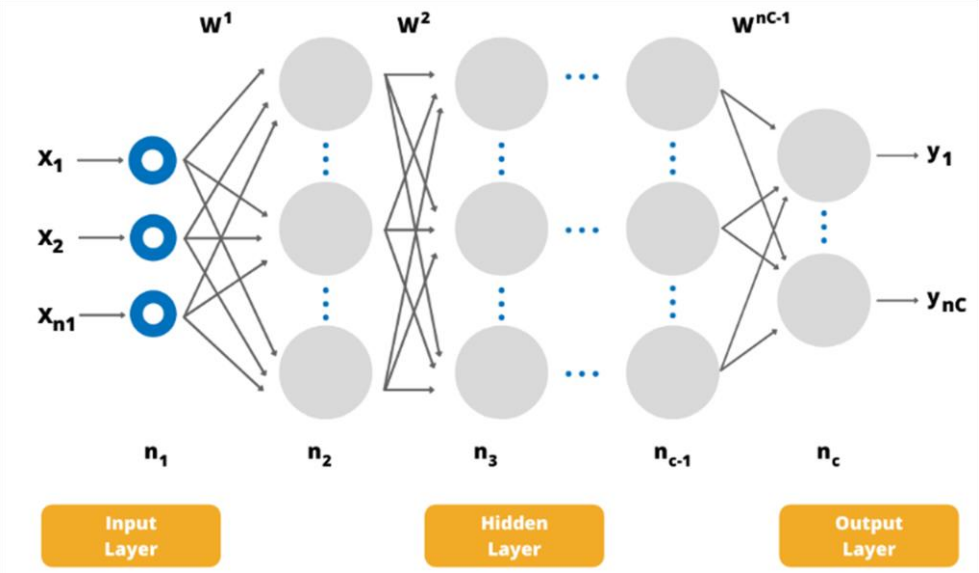


Fig. 9 Representation of ANN model

The core idea is that ANNs are computational models that represent in the networks of biological neurons we all have in our brains. A representation of a typical ANN model is displayed in Fig. 9. As we can observe, the structure is very simple: there is a first layer of neurons (input layer), where the input data is introduced. Then, more layers with neurons are interconnected (hidden layers) in a sequential pattern, which lead to a final layer of output neurons (output layer), that provides the predicted data. The input values (integer of real numbers represented by x_1, x_2, ..., x_n) are transmitted throughout these layers and neurons from left to right. They are multiplied by weight values (real numbers represented by w^1, w^2,.., w^n) and transformed in the neurons by means of activation functions. These concepts are not relevant for students, although they have to understand that the output values (integer of real numbers represented by y_1, y_2, ..., y_n), are a transformation of the input ones, and they depend on the specific values of some parameters, as in the algebra functions they already know. These parameters need to be adjusted, being the most relevant **the number of layers** that make up the network, **the number of neurons** per layer, and **the values of the weights** in each connection. The training algorithm is devoted with adjusting, basically, the weights.

In the case of the problem, we are facing here, Image Classification, Fig. 10 shows an example of a specific structure for the ANN model that should be clearly understood by students. We can observe that the input data are the image pixels, which are directly

Select Model (Model training)

introduced in the input neurons. Then, as an example, only one hidden layer is represented in Fig. 10, with 3 neurons, and one output layer with 3 neurons too. These outputs provide the **confidence** values of the 3 labels we have established in this example. The confidence value is a sort of probability representing how sure is the model in the prediction of each label, as we will see later in detail. In this example, the model has 9 weights (represented by w^{11}, w^{12}, ..., w^{33}) that must be adjusted with the training algorithm.

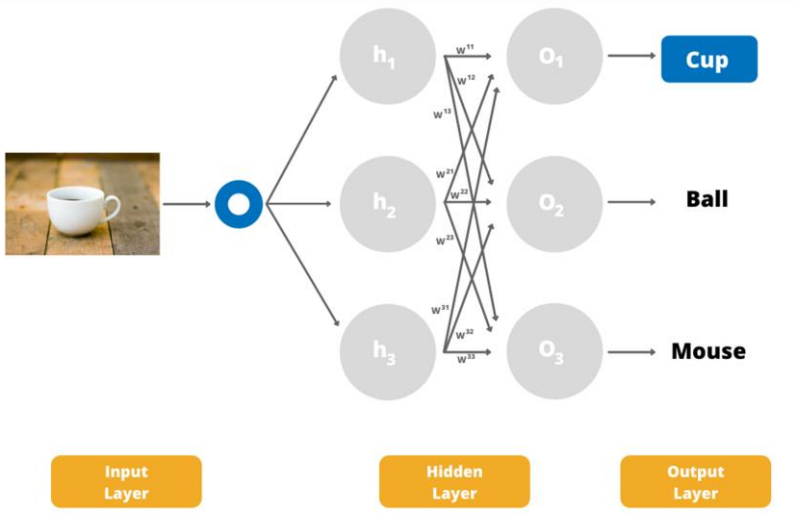


Fig. 10 Specific structure for the ANN model

Now students have to setup their own ANN model and perform training. As we can see in Fig. 11, in the second step of the PIC tool, the different parameters of the ANN and the algorithm can be established to train the model. In this case, the default values should be used, and it will only be necessary for students to click on the *Train model* button. This step implies adjusting the weights using the Adam algorithm until a predefined number of steps (typically called epochs) is achieved, which may take a couple of minutes to complete, depending on the number of images. Once the training process finishes, the set of adjusted weights is available to test the ANN.

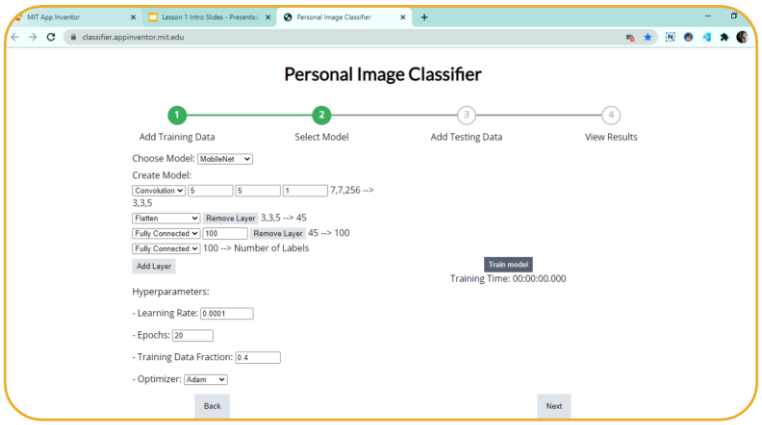


Fig. 11 Appearance of the PIC component web tool in the training step

Add Testing Data (Model testing)

In this step, students just have to upload new images of their selected objects, which must be different from those used for training. We recommend to add more than 5 new images of each type. To do it, students have to use the interface shown in Fig. 7, and the procedure is the same used in step 2 for training. Once all the images are loaded, it's time to click on the Predict button and see the results in step 4.

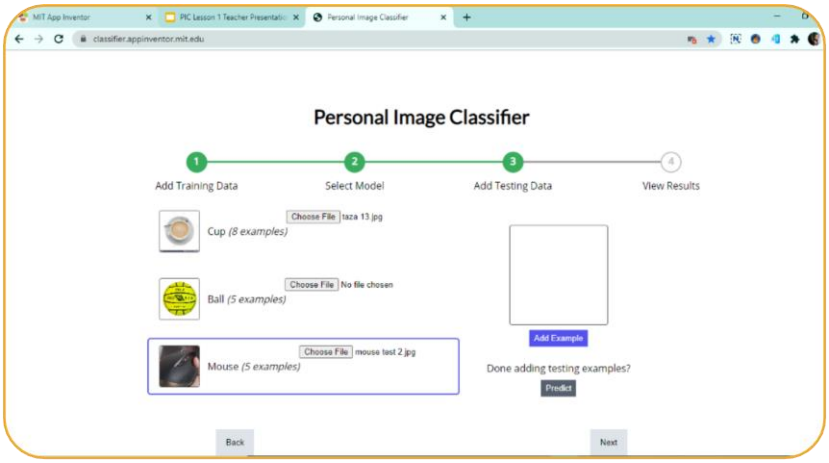


Fig. 12 Appearance of the PIC component web tool after loading the test data

It is recommended to test with images that are like the ones the player can take and considering that there are several external factors that can cause an erroneous classification such as the background, lighting, focus... Therefore, in case of failure, it´s necessary that these factors are not affecting the image.

View Results (Model usage)

In this step, students will see the test results of their model, so they can decide if it is good enough or it must be trained again. Fig. 13 shows and example of how these results are shown. In the central part, the objects are classified in two main categories depending if they have been correctly classified or not. In this case, the model predicted very well all the pictures corresponding to the mouse, but with 2 errors in balls and cups. In the left part of the screen, detailed information regarding the recognition is presented. We can see that, for the red cup, three real numbers below the image, which correspond to the confidence measure. In this example, we can observe that the red cup is predicted as a cup with a 69.771% of confidence, while the other two labels have much less confidence. In general, in this type of model, one of the labels has always a higher confidence value, being this the ANN prediction. For the example shown in Fig. 13, we must return to step 2, include more images of balls and cups, a re-train the model again until at least 90% of the images are correctly labelled.

View Results (Model usage)

As commented above, ANN testing is very fast, because the execution of the model is a direct process: each testing picture is used as input data in the ANN, the adjusted weights are placed in the corresponding connections between neurons, and the model provides as outputs the predicted the label for the picture. Once each group considers that their model works properly, the file with the ANN parameters must be downloaded to include it in the App Inventor template in the next task. This file has a ".mdl" extension and it should be stored in the computer.

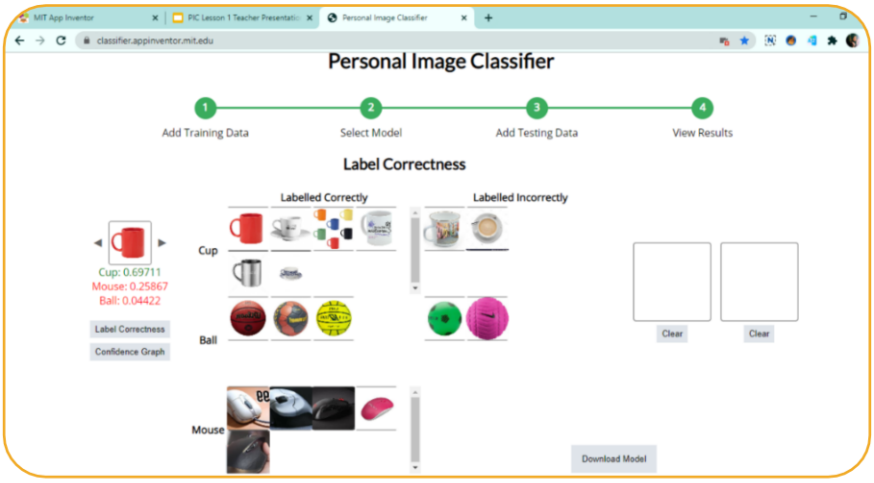


Fig. 13 Appearance of the PIC component web tool after predicting the objects

To better understand these steps, this video shows an example of a model training in a different scope. They try to detect when a person is happy, sad or surprised from a picture of the face taken with the computer's webcam. The video also shows how to load the model in App Inventor (from minute 4:30), but it is not necessary to watch this part.

In addition, in this link, there is a guide for the model creation where the previous are detailed. It is recommended that students follow it when making their models not to forget any step.

Student´s work:

Each group must perform the previous steps by themselves or at the same time the teacher is with the explanation.

1 hour and 30 minutes

Test:

In order to evaluate the task completion, **each group has to fill a brief report** explaining how the learning steps were performed, that is, the selected objects, how they took the correct pictures, how they performed the training, and how they tested the resulting model. This report should include screen captures of the different steps.

Activity 1 – Task 3

3

Upload the model to App Inventor and test it

1 hour

Required knowledge to be introduced to students:

The first step to carry out in this task is to upload the provided template in App Inventor 2. To do it, it´s necessary click on “my projects” -> “import project (.aia) from my computer”, as shown in Fig. 14, and select the file “students_template_TU4.aia” provided with this TU.

If loading is correct, the screen displayed in Fig. 15 should appear. The first time the program is open, it is convenient to use a large size (minimum tablet size) in the *Viewer* window and click on “Display hidden components in viewer”.

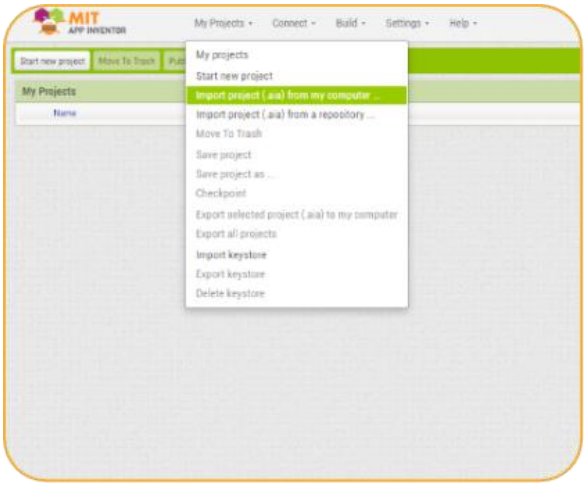


Fig. 14 Screen capture showing the menu to import the project template

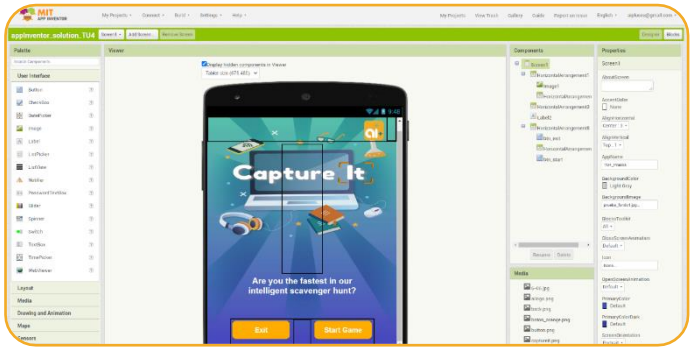


Fig. 15 Appearance of App Inventor just after uploading the template

Moreover, it is recommended that, to obtain an optimal result in the final app aspect, every group connects the smartphone to the computer (by USB cable or WI-FI), as explained at https://bit.ly/2Y84MRn . This way, the resulting screen will be displayed at the smartphone and students can observe the changes on-line. Once the connection has been established and the "MIT AI2 Companion" app is launched in the smartphone, the screen shown in Fig. 16 should appear.

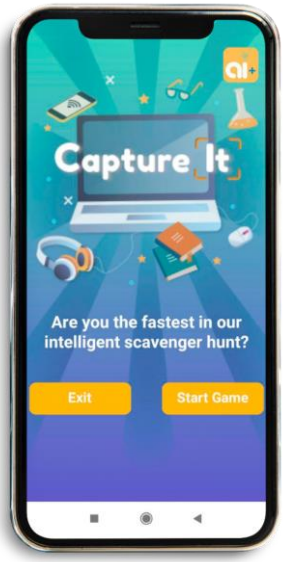


Fig. 16 Appearance of the application on the smartphone

Once the template is loaded, it's time to understand how the app works and the parts it has. As mentioned in the introduction, it has two screens. The first one, shown in Fig. 16, contains the name of the game, a text label with the description of the game, and two buttons: one to start the game (which is carried out in the second window) and other to exit the app. This first screen is already programmed in the template, but students can make any changes they want (change name, description...), considering all the recommendations regarding interface design provided in the Annex 1 of TU2.

By clicking on the button that starts the game, the Screen2 is loaded and, automatically, the game starts. This second screen, as can be shown in the Fig. 17, is divided into 3 sections.

Starting	As shown in the left image of Fig. 17, this first section contains a label within a vertical arrangement. Its function will be to display the countdown that sets the game start.
Object capture	When the countdown is over, a beep sounds and then the second section of Screen2 appears. This indicates that the game has been initialized. This part of the screen, shown in the middle image of Fig. 17, contains in top area a text label indicating the object that must be found, in the central part a camera viewer, below it the time counter and, at the bottom, a button that must be pressed when the object is found to take a picture and check if it is the correct one and makes the third section of the screen visible.
Results	This third section, displayed in Fig. 17 right, will indicate the user whether he/she has found the right object or not. In addition, when all the objects are discovered, this screen shows the time spent in the game, and asks the user if she/he wants to play again (as shown in the demo video).

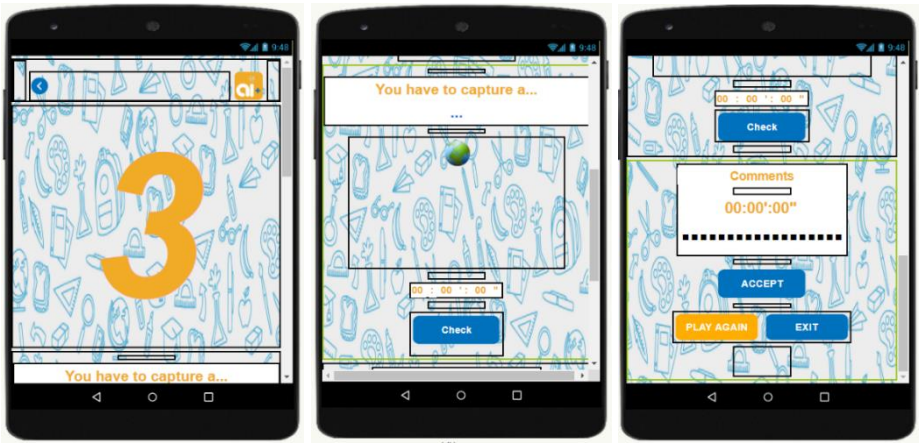


Fig. 17 Left: First section of the Screen2. Middle: Second section of the Screen2. Right: Third section of the Screen2

In order to reduce the work to be carried out by the students in this TU, the project template not only includes the graphical design of the game, but also the programming of Screen1, the change between sections 1 and 2 of Screen2, the chronometer, and some functions that simplify programming in the next activity. All the blocks already included in the template are explained in the "Annex I: Pre-programmed blocks", and it should be checked by the teachers and students if they need it.

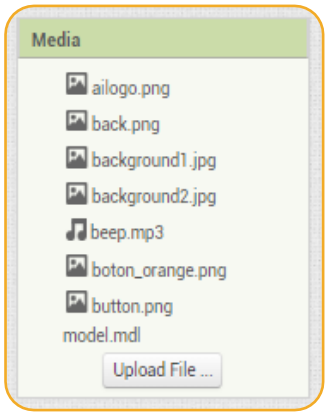


Fig. 18 Capture of media palette

Now is time to load the model learned in the previous task in the project template. First of all, the ".mdl" file must be uploaded to the media palette, shown in the Fig. 18. To do so, just click on the *upload file* button, select the model and accept.

Second, to connect the model to the app, it is necessary to use the extension introduced before, and called here *PersonalImageClassifier1*, which is already included in the template. To indicate the model it should work with, it is necessary that students open Screen2 and select PersonalImageClassifier1 from the non-visible components (Fig. 19).

Among its properties, there is a one called *model*, which must be clicked and the .mdl file of each group can be uploaded, as indicated in the Fig. 20.

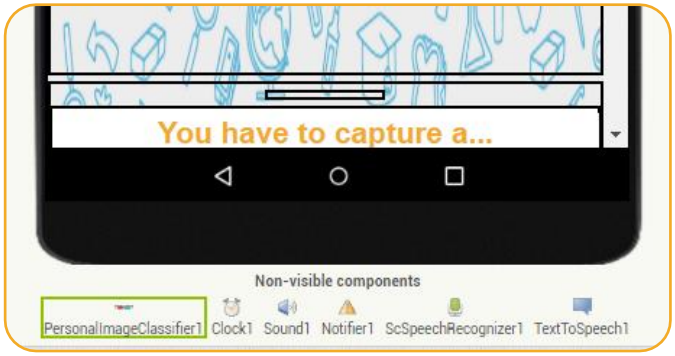


Fig. 19 Capture of the App Inventor where the component PersonalImageClassifier1 appears

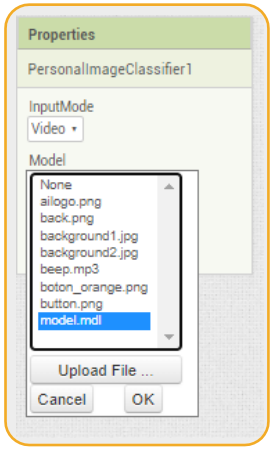


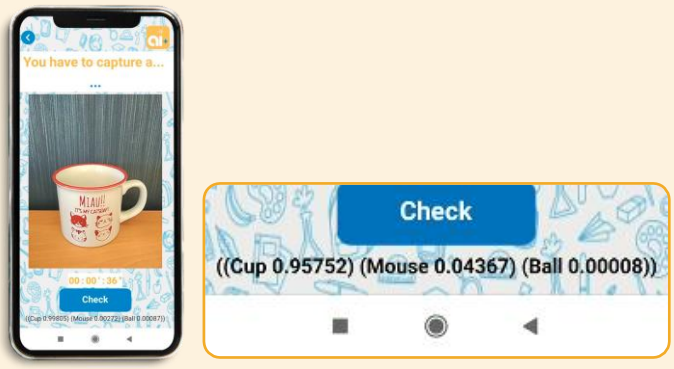
Fig. 20 Properties of the PersonalImageClassifier1

NOTE: It is important that students understand that this model is not continuously learning. Once it has been loaded into the app, the model does not retrain and it performs all its classifications based on the training carried out in the Personal Image Classifier.

Student´s work:

In this task, you have to modify the template to create a program that should:

- Load the learned model correctly
- Display the result obtained after the classification of an object, including the confidence of each label. Screen 2 must show the following information:



NOTE: The correct functioning of the model should be tested using these confidence values. Think about why, when an object other than the three in the model is captured, there is an error in the prediction.

- Once the adequate functioning has been tested, the program must be modified to display only the name of the object with the highest confidence value:

NOTE: Using the pre-programmed function *get_name()* described in Annex I, represent in the added label only the name of the classified object (taking into account that this value will be used later in other EVENTS). You need to revise your knowledge about lists in programming.

40 minutes

Test:

To evaluate the task completion, **each group should complete the report started in task 2** explaining now what results they got from testing the model in the app, and why the model fails when the captured object is not one of the trained ones. Moreover, they must show the teacher that their model works properly in the smartphone app, recognizing their selected objects with no fail or with a very small number of failures. In addition, the teacher must cover **the points 1 and 2 of the checklist in the Annex IV: Teachers program test.**

Teachers solution:

The first step is to add the model to the Personal Image Classifier component, as explained in the *Required Knowledge* section. Next, we recommend teachers to review the different blocks associated to the *PersonalImageClassifier1* component in the block editor (Fig. 21). In this TU, only three of them will be necessary: *when PersonalImageClassifier1. GotClassification* and *call PersonalImageClassifier1. ClassifyVideoData*.

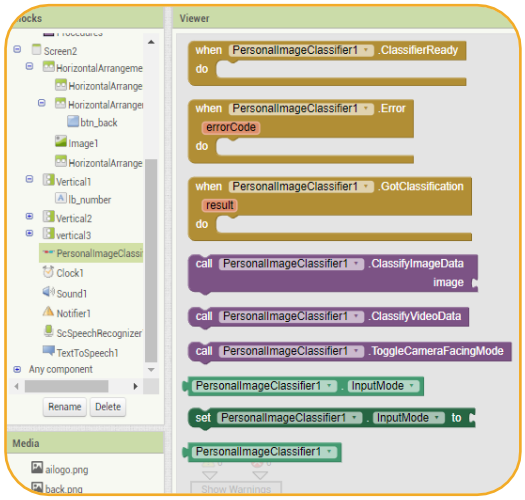


Fig. 21 Programming options of the PersonalImageClassifier1 component

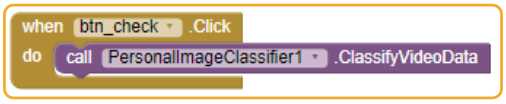


Fig. 22 Solution to call the image classifier

At this point, the functionality to be programmed is: when the user clicks the check button, the learned model is used and the classification must be performed. To do this, there are two options to carry out classification: *ClassifyImageData* and *ClassifyVideoData*. The first one opens the camera and, when a photo is taken, it performs the classification and shows the result. The *ClassifyVideoData* blocks, on the other hand, performs the classification of the focused object in the application's viewer, without requiring to take a photo. In the solution explained in this TU, we will use the second opting.

The idea is that when the player clicks the check button. A possible solution is displayed the one displayed in Fig. 22, where the function call *PersonalImageClassifier.ClassifyVideoData* is used within the event *when btn_check.click*.

The next step is to represent the result provided by the model. The extension contains the event *when PersonalImageClassifier1.GotClassification*, which runs after classification has been completed and returns the result. To see if the model works correctly, and also to see the result format, a temporal auxiliary label will be added under the check button (Fig. 23) to show it.

Fig. 23 Capture of the auxiliary label in App Inventor

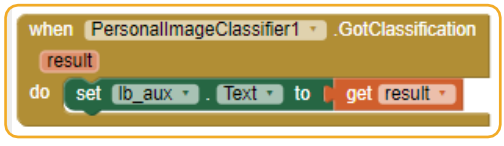


Fig. 24 Solution to display the classification result on the auxiliary label

A possible solution for loading the result on the label is shown in Fig. 24. As can be observed, it is only necessary to load the result variable (event handler output) in the text property of the created label.

As commented above, the result contains three confidence values for the three labels defined by each group. In our example, the object that was classified was a cup, and the model returned a 94.92% confidence that it was the indicated object, a 4.93% that it was a mouse, and with 0.12% that it was a ball. Consequently, the classification was correct.

To check the model in the smartphone, it´s necessary do it with several objects to see if it is working properly. Several tests are recommended with the three types of objects selected: put the same object on different backgrounds, check that it recognizes different objects of the same type (for example, cups of different sizes) and taking into account that there are external factors that can cause an erroneous classification such as the background, lighting, focus...

Fig. 25 Example testing the model in the application

If an object other than the three defined is captured, the model fails. In this case, the model returns the same set of three outputs, but as the object was not included in training, the confidence values will be the most similar objects, which could induce an error in the app. This problem can be solved in several ways. In the case that the object checked is not one of those in the model, the confidence percentage will be low. Thus, a possible solution would be that when the confidence percentage is not at least 60%, it would be taken as a wrong object. Therefore, the errors that can occur in these situations are eliminated. Another possible solution would be to show the model with many objects. In this way, when an incorrect object is scanned, the application would recognize it. Thanks to this, the success rate would be much higher.

Once the learned model has been validated, is the time to modify the program so that it only displays the *name of the object with the highest confidence* in the text label and, in addition, in a variable too, because it will be required later in the app. To do it, we provide a function called *get_name()* in the project template, which returns the name of the object that was detected with a highest confidence. Basically, these function (Fig. 26) search in a list of numbers that is previously ordered from the highest to the lowest, returning the first item.

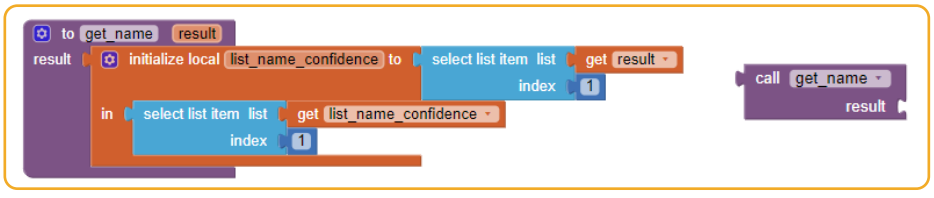


Fig. 26 Left: Get_name function. Right: get_name function call

Fig. 27 shows a final solution in which a variable called "classification_name" is created, the name of the object is saved by the function *get_name(),* and represented in the label.

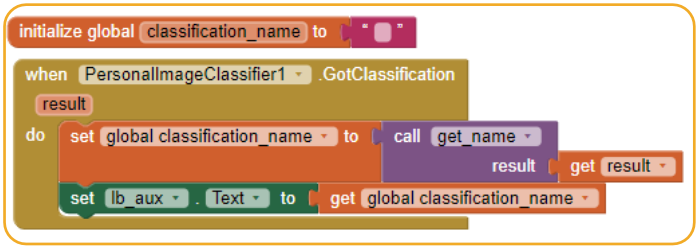


Fig. 27 Solution to the problem of representing only the object name

Activity 2:

Implement the *Capture it* game app

3 Hours

Required knowledge to be introduced to students:

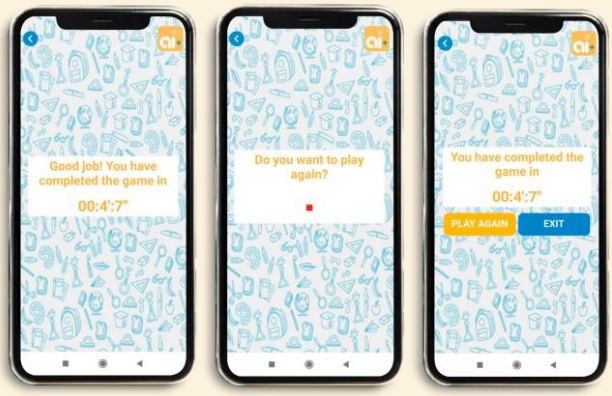
Once the main topic of this TU, that of introducing supervised machine learning from a practical perspective has been carried out by students, it is time to finish the game app so it can be really used. This second activity will be more focused on interface design, human-machine interaction and computational thinking.

Student´s work:

Create a program based on the following specifications:

The global functioning of the game is displayed in the next captures and it can be checked in the demo video. Once the user selects to start the game, the name of the first object to search will be shown in section 2 of Screen2, as shown in first photo for the case of a cup. When the user clicks the check button, the app checks if the object focused in the central frame viewer is the expected one. In this case, the focused object was not the right one and when clicking on check, the application notifies that we must continue searching, as shown in the second capture. In the third, the correct object was already in focus and by clicking on check, a message should appear indicating the player that he/she has found the correct object (fourth capture). When the user clicks the *Accept* button of this screen, the app will back to section 2 and the name of the new object will be shown, as illustrated in the last photo.

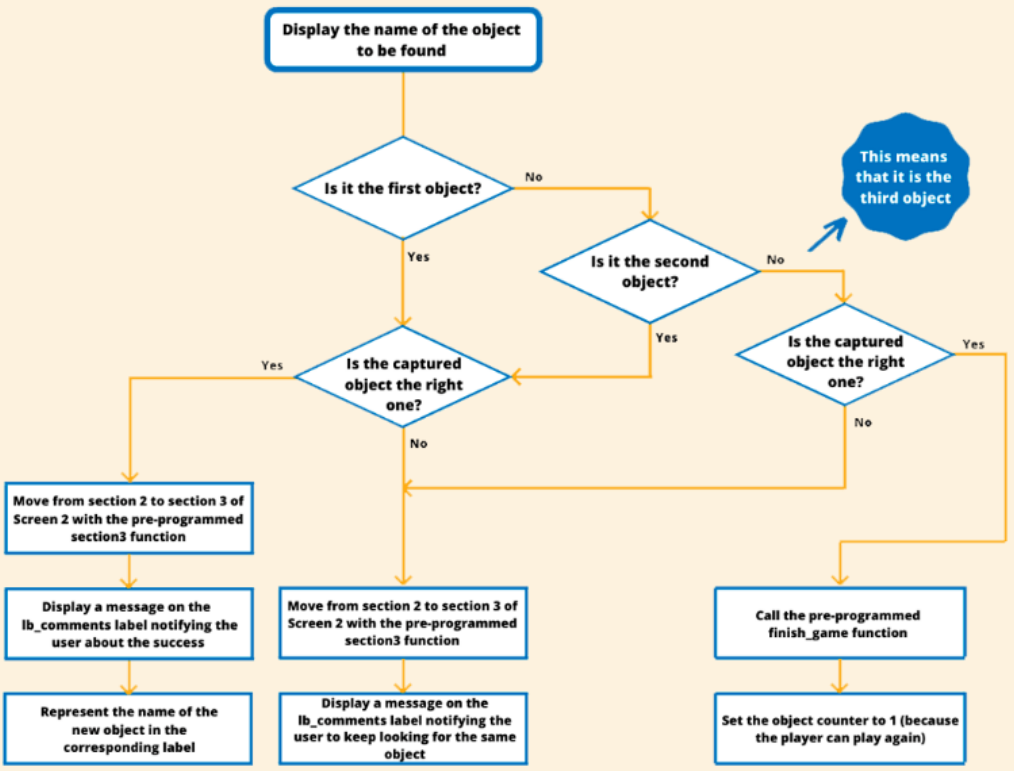
In the course of the game, the players have to move around the school searching for the objects proposed by the app, and the previous screens will be repeated until all of them are found. At this moment, three last screens will be shown. First, as displayed in the left capture of the next photographs, a congratulation message will be *shown and spoken*, including the time required to complete it. Next, a new screen will appear which asks if the user wants to play again (middle capture). The user has to answer by voice. If the answer is *yes*, the game will start again, and if it is *no*, two new buttons will appear on this screen: *play again* and *exit* (right capture).



To guide you in the development of this program, you have to perform the following steps:

- Load the name of the objects their model recognizes in the list variable "objects" pre-created in the template.
- Read the first object from the “objects” list variable and write it in the label called “lb_object” to see it when the game starts.
- Analyse in which event handler the representation of the appropriate message will be carried out after classification.
- For the correct functioning of the app, it will be necessary to carry out the programming of the event handler decided in the previous step following the following flow chart. As indicated in it, it will be necessary to use two of the pre-programmed blocks: finish game and section3. Clue: it may be necessary to create a variable as a counter

NOTE: It is mandatory to create at least one function during the programming of this last point.



3 hours, including 1 hour to test the app

Test:

When students finish their game, it´s time to do the final test of the app. They must prove that their game is working correctly, i.e. that the correct messages are displayed, and that the transition between screens works properly. As this is an entertaining app for students, we recommend that they play in groups using several students the same app during the last one hour.

Teachers solution:

To carry out the first step, the names of the objects trained in the model must be added to the *objects* variable, in the order in which they appear in the game. The *objects* variable is already initialized in the project template as a list type variable with 3 inputs (in the case that any group decided to use more than 3 objects in their app, this variable should be modified including more items). In each of them, each of the objects must be added considering that the name must be

exactly the same as the one used in the model definition. Fig. 28 shows an example, for teachers, of how this variable can be initialized with 3 objects.

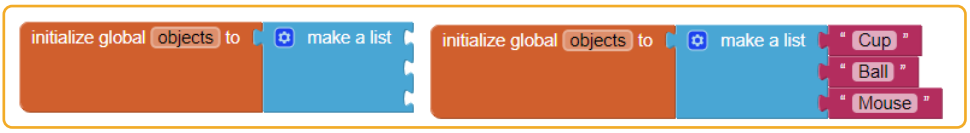


Fig. 28 Left: Global *objects* list. Right: Global *objects* list initialized

Now it´s time to read the objects from the list and show them in the corresponding text label of Screen2. As shown in Fig. 29, there are many option to interact with lists such as adding or removing items, checking if an item is in a list, copying a list into another one, etc.

Fig. 29 Lists programming options

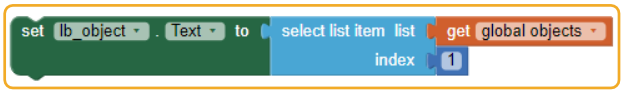


Fig. 30 Solution for writing the first element of the object list in the lb_object label

In this case, as a starting step, we recommend reading the first element in the list and adding it to a text label to show it in Section 2 of Screen 2, as displayed in the left image of Fig. 31.

This can be done by using the "select list item" block. This block has two entries: list and item. In the first one it is necessary to annex the list from which we want to consult an element, and in the second one, to set the position of the element in the list (integer number). The block returns the value stored in the position, that can be shown in the corresponding text label as displayed in Fig. 30.

This code can be place in different parts in order to display the object name at the beginning of the game, such as placing it when Screen 2 is initialized as shown in the Fig. 31 (after the two *call* blocks that are already in the template which must be maintained).

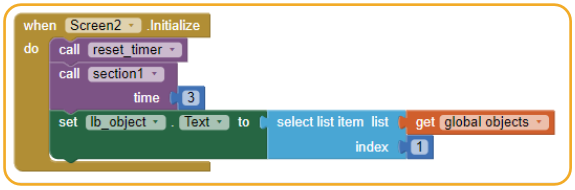


Fig. 31 Representation of the name of the first object when Screen2 is initialized

Following the steps mentioned above, it is time to interpret the result of the classification method and show the messages to the user. Let's start from the program developed in task 3 of activity 1. But now, instead of representing the result of recognition in the text label, we have to display, in section 3 of Screen 2, the appropriate message to the user, and present a new object in case the previous one was found. So, first of all, the auxiliary text label added in the event handler *when PersonalImageClassifier1.GotClassification* must be removed. It is important to understand that **it is in this event where the rest of programming will be carried out**, because all the actions occur after recognition has been performed.

Following the flowchart, the first thing to do is to control the object the user is looking for at any given time. Typically, this is achieved by having a global variable that increases as the objects are found. This way, if the variable is 1, the user is with the first object, if it is 2 with the second one and so on. In this case it has been decided to call it *track*, as can be seen in Fig. 32.

Fig. 32 Global track initialization

According to the flowchart, a conditional based on this variable must be created within the *when PersonalImageClassifier1.GotClassification* event. Fig. 33 shows a possible solution.

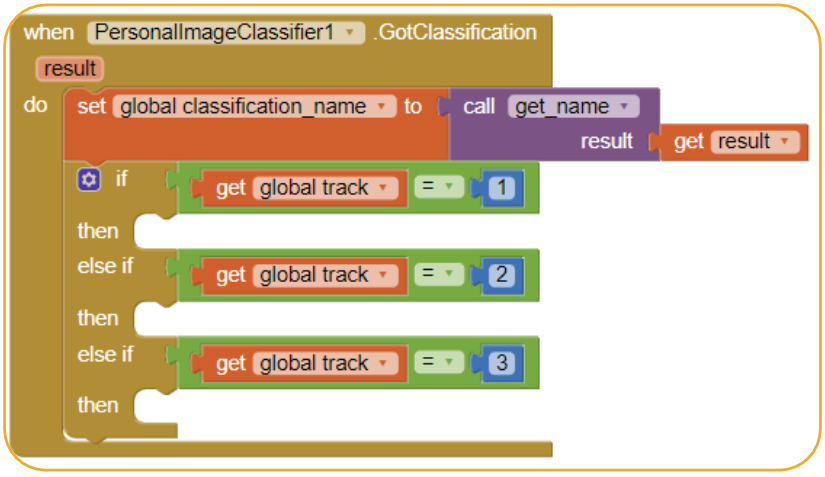


Fig. 33 Track conditional

The next step will be to check if the found object is the right one. To do this, within each of the conditionals, a new branch must be created that simply checks if the content of the variable where the name is saved (*global_classification_name* in Fig. 34) is the same as the one saved in the *objects* list (this block was previously developed).

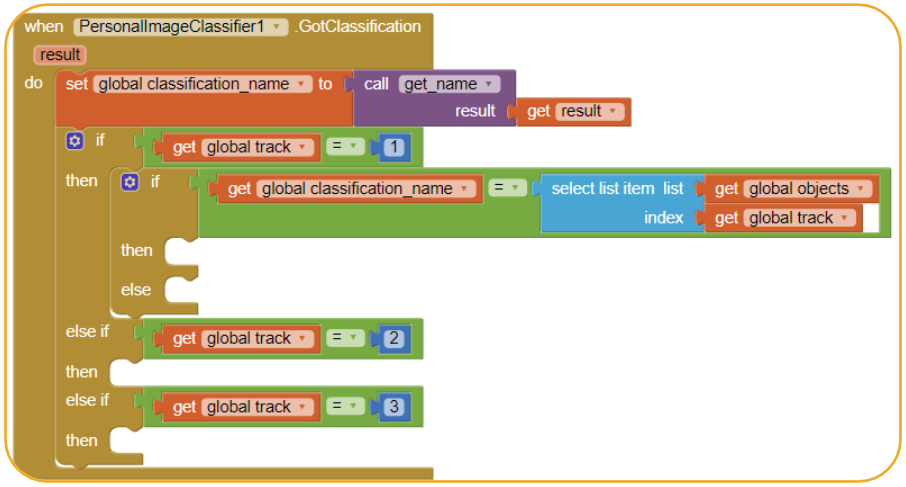


Fig. 34 Code with the conditional that checks if the object is correct or not

According to the flowchart, if the object found is the correct one, the program should perform the following 3 steps:

1. Move from section 2 to section 3 of Screen 2 with the *section3()* function (Fig. 35).
2. Display a message on the *lb_comments* label notifying the user about the success.
3. Represent the name of the new object in the corresponding label.

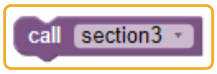


Fig. 35 Section3 function call

On the other hand, as shown in Fig. 36, to show the message in the label called *lb_comments*, it´s only necessary load the message they want in text format in it.

Fig. 36 Block that show a message in the lb_comments label

It is necessary to load the name of the next object into the *lb_object*. This way, when the user presses the *accept* button in section 3, they can see the name of the new object. The best solution for this is that shown in Fig. 37, which is to select the name from the list. As in the previous step the *track* variable has been increased, the correct element will be read.

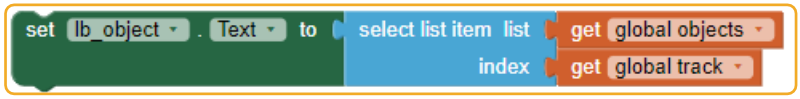


Fig. 37 Code that shows a objects list item in the lb_object label

Finally, it´s necessary to increase the variable track. For this, the sum math block can be used and add 1 to the variable, as shown in the following figure.

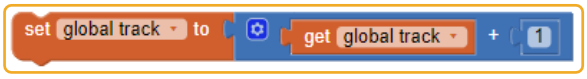


Fig. 38 Code that increase the track variable

The final aspect of the program for the first object would be that displayed in Fig. 39:

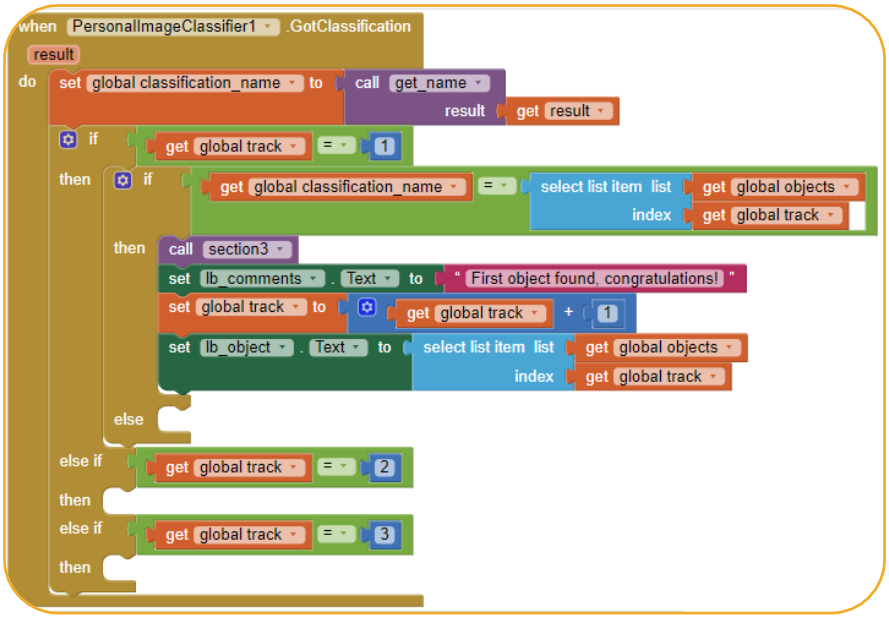


Fig. 39 Code with the found object part programmed

If the object found is not the right one, it would only be necessary to show, in section 3, a comment informing the user that he/she should continue looking for the same object. As a consequence, the solution is that displayed in Fig. 40, where the code shown in Fig. 35 and Fig. 36 has been included.

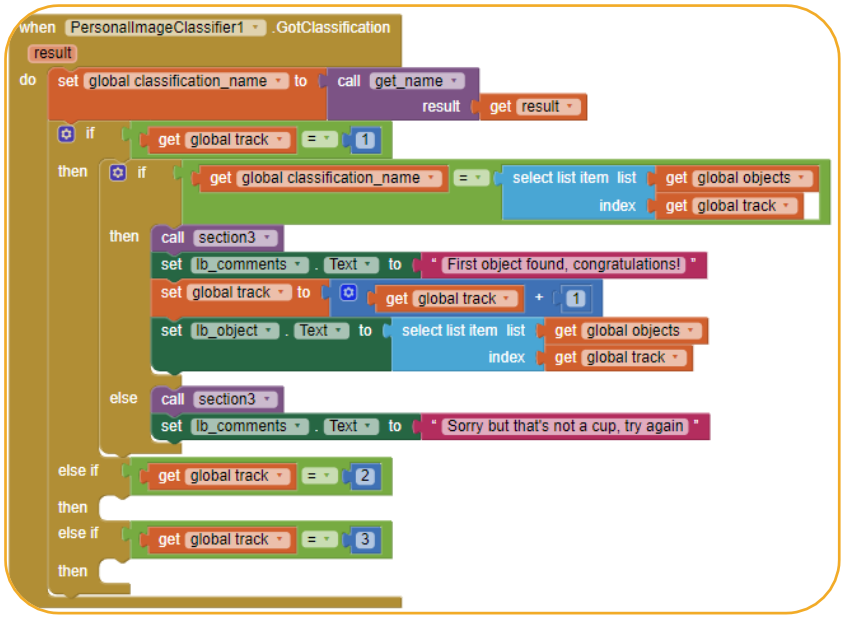


Fig. 40 Code with the not found object part programmed

When *track* is 2, the steps to follow are the same as in the previous case. The only change would be the message to show to the user. Fig. 41 shows a possible solution in this case.

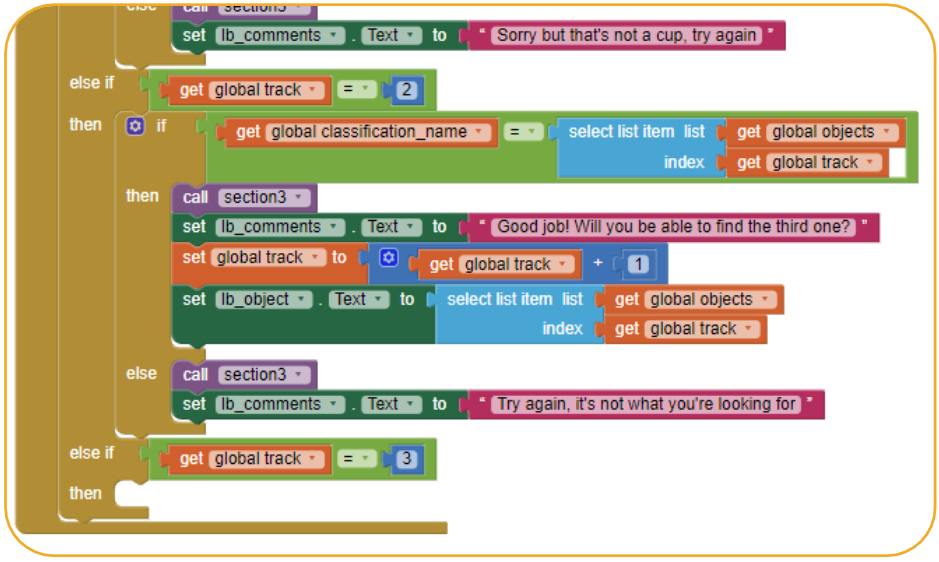


Fig. 41 Code with the second object programmed

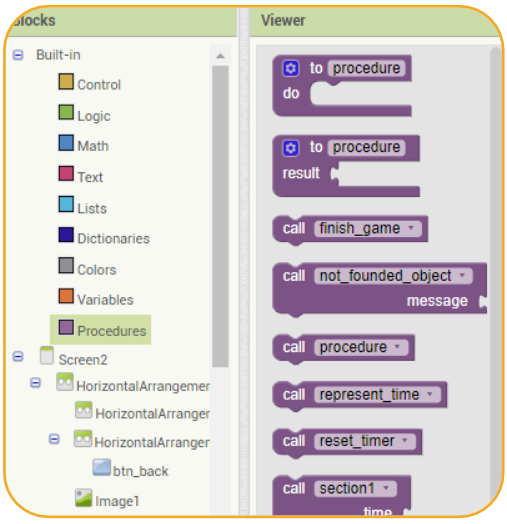


Fig. 42 Procedures programming options

At this point, students should realize that they have to write down the same code, so **they must create functions**, to simplify the program readability and to increase reuse of previous codes. In this case, two functions will be necessary: one if the object is found and other when it is not. Details regarding function creation in App Inventor (called procedures) should be revised by students in case they need (Fig. 42): https://bit.ly/2DoE5QJ

Fig. 43 shows a possible implementation of the *object_found()* function. The only change made from the previous code was to change the message text to a more generic one that is valid for any object.

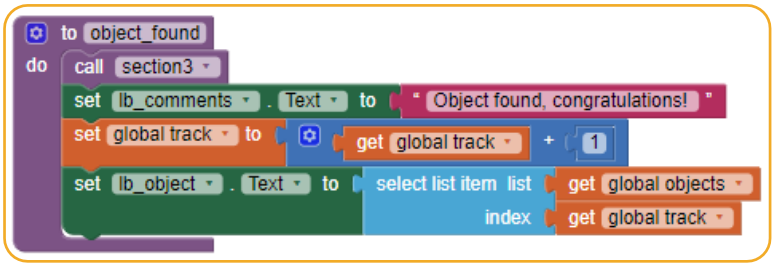


Fig. 43 Solution to object_found function

If the object is not the correct one, a function called *object_not_found* has been created and, as in the previous case, a general message has been written (Fig. 44).

Fig. 44 Solution to object_not_found function

The last step would be to call these functions in their respective places, as indicated in Fig. 45.

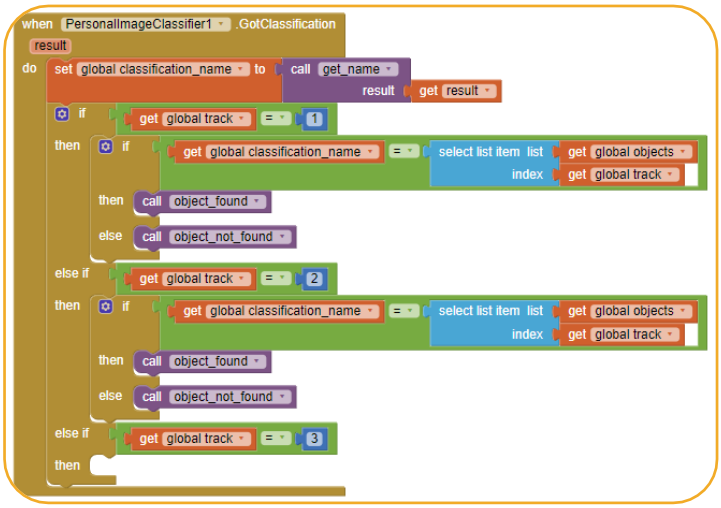


Fig. 45 Code with the first and second object programmed

NOTE: If students would like the messages to change for each of the objects, it is possible to create a function with an input, so that each time it is called it will be necessary to enter the desired message.

Finally, for the last object, if it is not the right one, the *object_not_found* function can be called as before and if it is correct, a provided function called *finish_game* must be used by students, as shown in the flowchart. In Fig. 46, the final aspect of the program is shown with this third part of the conditional already filled.

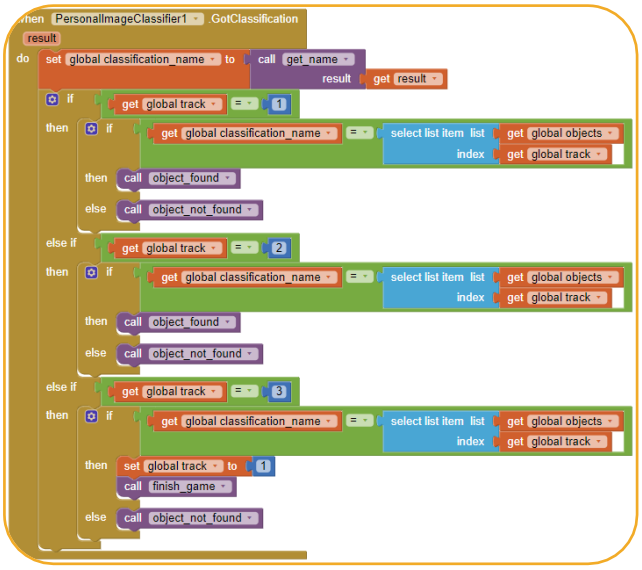


Fig. 46 Code with all objects programmed

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 record the user's speech.
- The touch screen, where the user can click on buttons.

Regarding actuation, the app includes:

- The LCD screen, on which the game information is displayed.
- The speaker, communicating the end of the game.

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

- The creation of a model able to recognize objects from pictures. The model was trained using supervised learning over an Artificial Neural Network that performs image classification. All these concepts, although new for them, are very relevant and they will be included again in future TUs.
- Data preparation, in this case, through the procedure followed to select adequate images for learning.
- Model validation, that is, how to check if a model works properly or if it needs to be improved. In addition, students have learned why testing can fail as a consequence of an inadequate data preparation.

To conclude, teachers should point out to students that this is the first TU where the key property of an intelligent system has introduced, which is learning. The implemented app is able of generalizing from experience, in this case, of recognizing objects which can be slightly different from those used for training.

11. EVALUATION

As established in the introductory TU, three evaluation methodologies are proposed too for this TU, with the following weight:

Assessable activity	Score
1.1- Teacher program test 1.2- Students program test	46 10
2.1- Kahoot 2.2- Students report	14 15
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 by the teacher and by other groups. Therefore, we will evaluate this part following two elements:

   1.1 *Teacher program test:* 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 IV: 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 46).

   1.2 *Students program test:* we propose that students test the applications of their classmates and evaluate them. To do this, each group will be assigned an application (they should not know whose app they are evaluating) and they should cover the table in the **Annex V: Students program test.**
   Each of the points present in the checklist must be added up, and the final evaluation of this part will be obtained (maximum 10).

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*: this methodology is very important, and it will be evaluated using two different elements:

   2.1 At the end of the TU, the **students must fill a kahoot survey** individually (Annex III: Student's survey). This evaluation activity represents 14 points of the unit totals. Because there are a total of 7 questions, each successful question will add up to 2 and failed questions will not subtract.

   2.2 Individual work report, which will be done by students at the end of the tasks 2 and 3 of the activity 1. This report has a total score of 15 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. This report must contain everything mentioned in the **test sections of the tasks 2 and 3** to be perfect. It will be the teacher himself who decides the grade of those who have forgotten some concepts or have put them incorrectly.

3. *Ongoing work during the TU:* the teacher must fill for every activity an individual rubric that (Annex II: 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 3 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. Add More Objects

The first proposed improvement is to create a model that can recognize more objects, and thus, achieve a more challenging game. It is important to highlight to students that adding more objects does not imply to create a program with more conditionals to change between windows in Screen2, because all of the are equivalent. That is, in case they require to change the program, they must try to make it as general as possible, with independence of the number of objects.

2. Make Objects Appear In Random Order

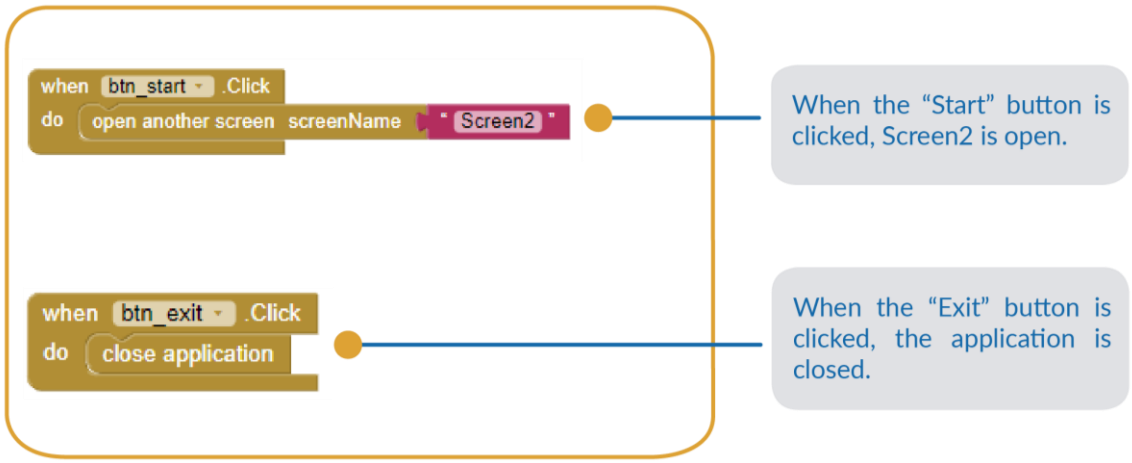
The game created always shows the objects in the same order. This causes that when a player has already played, he will know both the objects and the order in which they appear. In order for this not to happen, it is proposed that the objects are shown in a random way and that they are not always shown. That is, if there are a total of 10 trained objects, the game will only show 8 at random (for example).

13. ANNEX

ANNEX I: PRE-PROGRAMMED BLOCKS

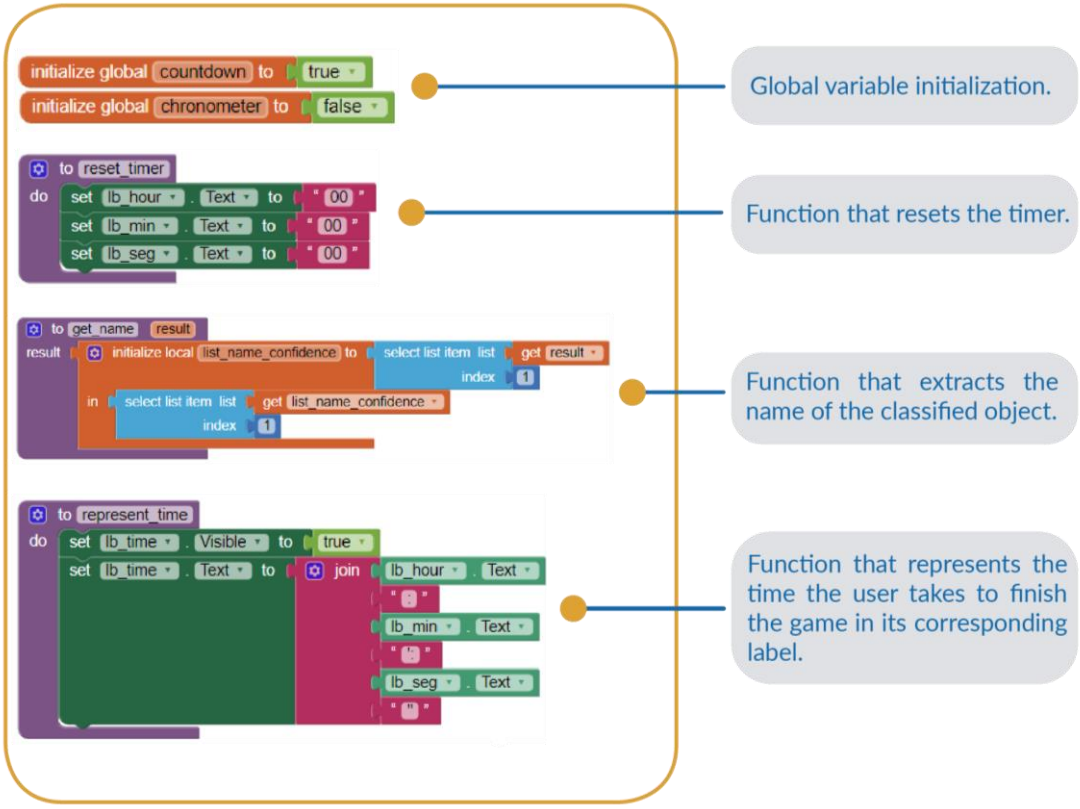
- **SCREEN1:**

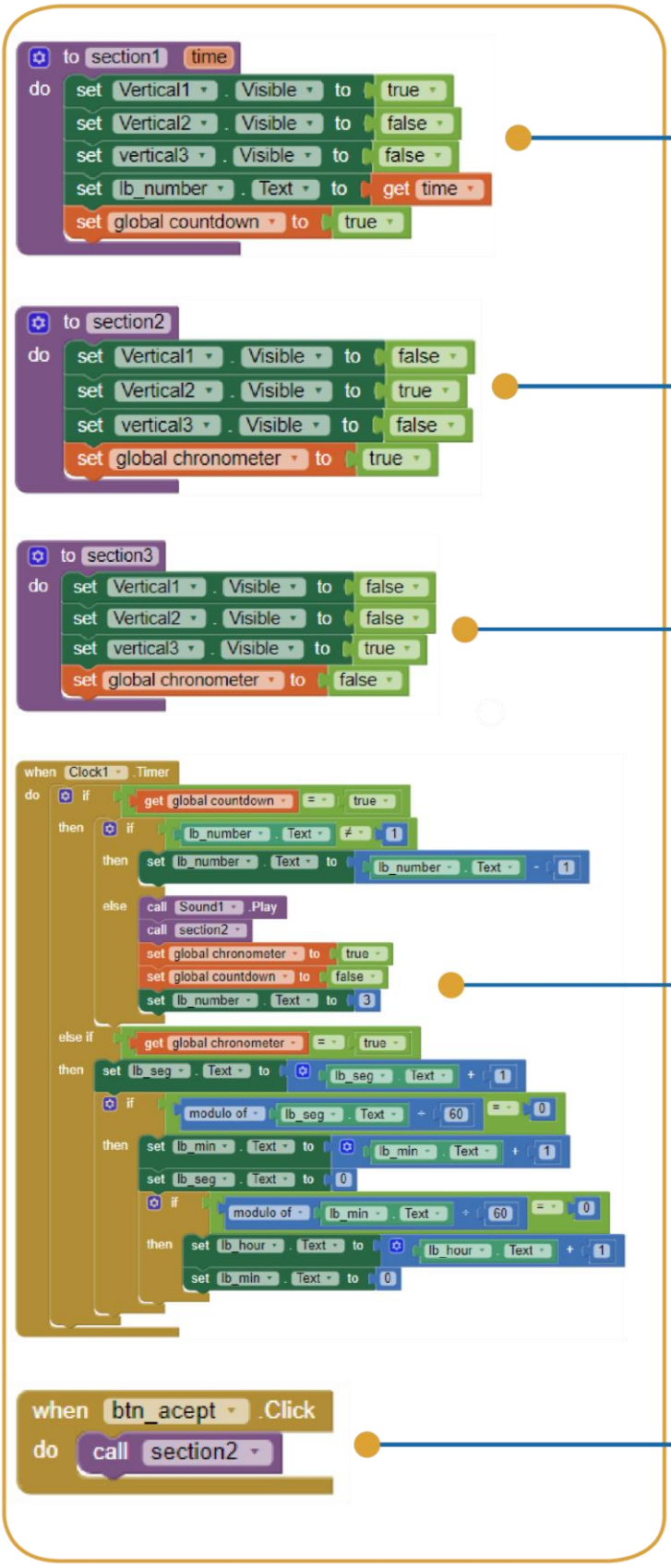
In this first window, all the blocks are completely programmed.



- **SCREEN2:**

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





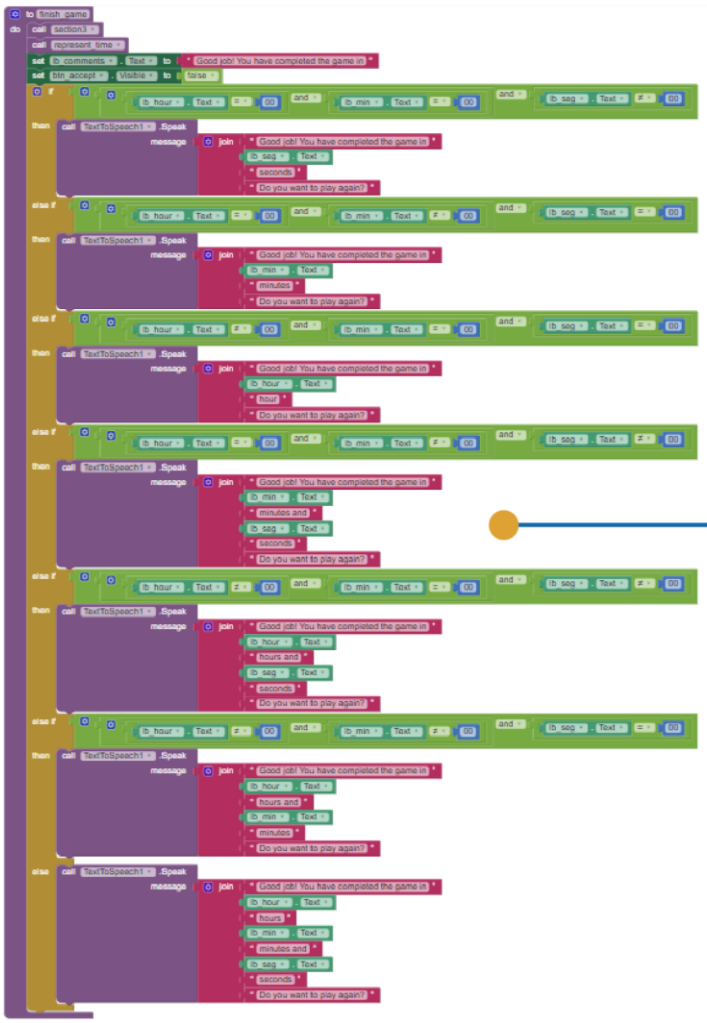
Function that displays section 1 of Screen2 on the smartphone's screen.

Function that displays section 2 of Screen2 on the smartphone's screen.

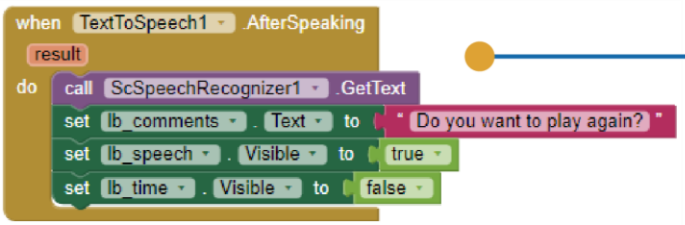
Function that displays section 3 of Screen2 on the smartphone's screen.

Programming of the timer and the countdown.

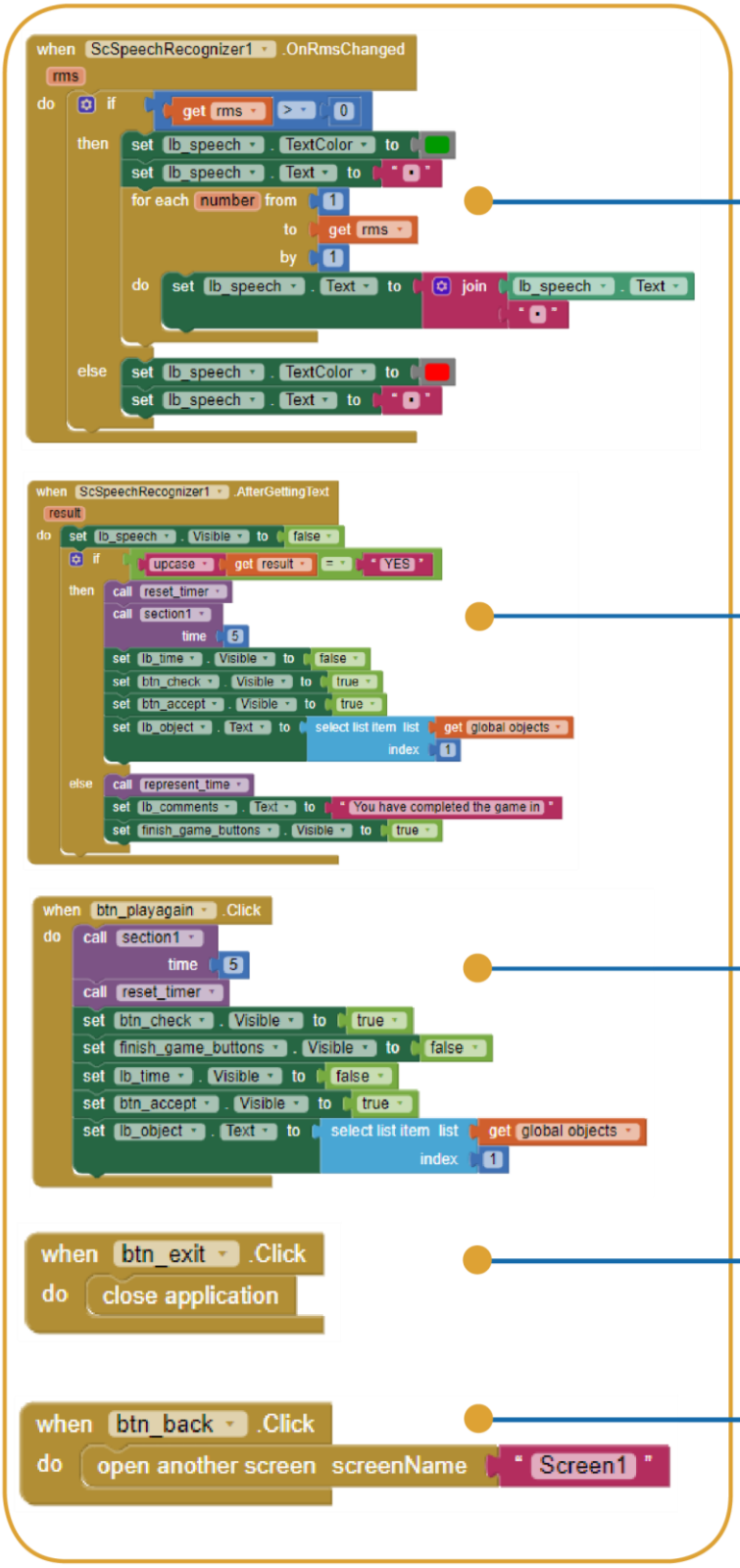
When the "Acept" button is clicked, section 2 is displayed on the smartphone screen.



Function that shows the time the user has taken to complete the game. In addition, the user is asked by voice if he wants to play again.



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.

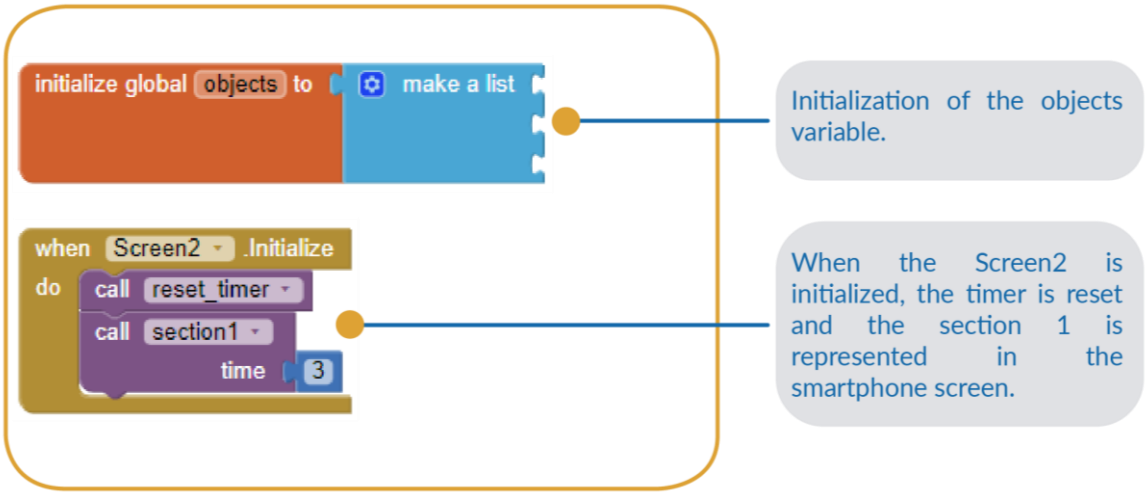
Programming of the voice recognition part to start the game again or not, once the user's answer is received.

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:



ANNEX II: 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 III: STUDENT'S SURVEY

Teacher can enter the kahoot in the following URL: https://bit.ly/32TrGOA

1. The steps to follow to learn a model are:
    - Prepare the data, train the model and use the model
    - Prepare the data, test the model and use the model
    - Prepare the data, train the model, test the model and use the model
2. The property of intelligent systems so they are able of generalizing from experience is:
    - Perception
    - Reasoning
    - Learning
3. The value that provides the probability of recognizing an object is called:
    - Information value
    - Confidence value
    - Error rate
4. The quality of the images in the dataset for training depends on:
    - Their size and format
    - The variety in orientations and the contrast from the background
    - The number of objects included in them
5. The developed game does not use any sensor, and therefore, does not work the perception topic.
    - True
    - False
6. The models learned in this TU are able to detect if an object is not one of the trained ones.
    - No, they always consider it one of the trained objects.
    - Yes, they can detect it.
    - Depends on whether the object looks like one of the trained ones.
7. The model created not only learns when it is trained, but also learns from its mistakes.
    - True
    - False

ANNEX IV: 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	When a scan is performed, the name of the scanned object appears on the smartphone screen.	5		
2	The number of errors in the classification is very low (try a couple of different objects).	6		
3	When the found object is wrong, a message appears indicating it and the accept button.	6		
4	When the found object is correct, a message appears indicating this and the accept button.	6		
5	When the button accept is pressed, if the found object was the correct one, the name of the new object appears and if it was not, the name of the same object.	8		
6	When the last object is found, the finished game message appears with the total time spent.	5		
7	The human-machine interaction aspects are correct: clear and direct sentences	5		
8	The general app functioning is correct, with no stops or pauses	5		

ANNEX V: STUDENTS PROGRAM TEST

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

Available in .docx format at:

https://drive.google.com/drive/folders/1lohSLWg8yRsZQEiM2X2oW6Ne_hDYqI6M?usp=sharing

	Level (score) / Aspects to be evaluated	Score	Yes (total score)	No (0 points)	If the answer is no, explain why
1	The course of the game. If it was correct, with timely messages at all times, and using a natural interaction process.	2,5			
2	The object recognition worked correctly	2,5			
3	The written instructions were clear and true	2,5			
4	All buttons work correctly (accept, play again, check ...)	2,5			