TEACHING UNIT 2

Introduction to Perception and Actuation 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

Perception and actuation are the two first topics of AI we will cover in these introductory Teaching Units (TUs) from a practical perspective. They are, probably, the two topics that students are already more familiar with, because they are treated in other technological subjects like robotics or programming (for instance, in languages like Scratch).

The objective of this TU is not to show all the possible sensors and actuators that can be used in AI, but to understand how to apply some of the most relevant and start thinking about the type of smart applications that can be developed with them.

To this end, in the current TU, students will develop a smartphone app that uses the three most important sensors in AI, cameras, microphones and tactile screens, as well as two very relevant actuators, speakers and LCD screens. With them, students will create a simple but useful app that exploits human-machine interaction to capture and show information to the user in a natural way. This app will be improved in the next TU, so its real utility will be clear then.

2. Context

For the student 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. They should have learned the following basic topics: conditionals, loops, functions, and variables. There are specific recommendations about this in section 4.3 of the introductory TU, so we assume that students have followed them.
- **Basic knowledge of MIT App Inventor**: students should have basic knowledge about the App Inventor application (https://appinventor.mit.edu). To start programming with App Inventor, they should follow two main steps:
    1. *Understanding the interface*: App Inventor consists of two main parts, the Designer and the Blocks Editor. To learn how to use them at classes, you should read carefully the contents described at this link.
    2. *Learning the basics*: we highly recommend carrying out the beginner tutorials that can be found at this link. Specifically, students should perform those called: "Hello Codi!", "Talk to Me", "Ball Bounce" and "Digital Doodle".

3. Learning objectives

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

SPECIFIC

- QR code sensing fundamentals
- Speech production fundamentals
- Speech recognition fundamentals
- Human-machine interface fundamentals
- Gyroscope and magnetometer sensors fundamentals

TRANSVERSAL

- App Inventor software usage
- Computational thinking: conditionals, variables, and functions

4. Contents

To start working with perception and actuation in AI, students will develop a smartphone app using App Inventor. Specifically, they will use the following smartphone sensors:

Sensor	Information provided
Camera	School place name by scanning a QR code
Gyroscope and magnetometer	Phone's spatial orientation
Microphone	User's speech
Tactile screen	Button that is clicked by the user

Regarding smartphone actuators, in this TU, we will use:

Actuator	Action performed
Speaker	To show app information by voice
LCD Screen	To show app information through the smartphone display

5. Temporary organization

The TU proposes a main challenge to be solved by students. To achieve it, it has been divided into 2 activities of 2 hours each, and each activity has been organized into a set of small tasks, as shown in the scheme below.

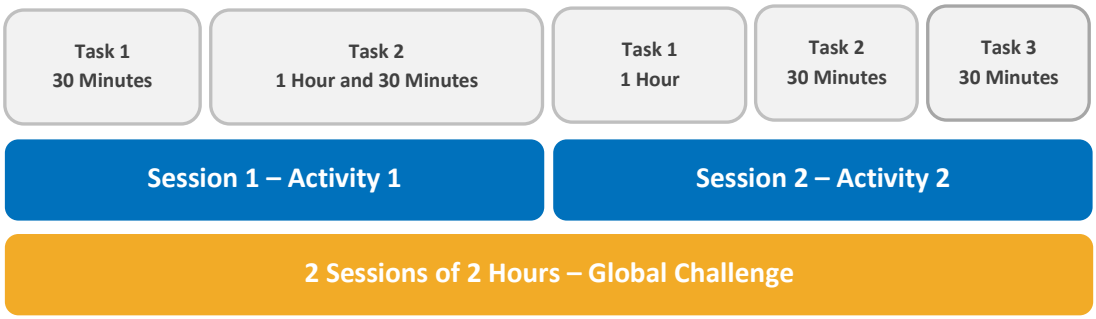


Fig. 1 Temporary organization

6. Necessary resources

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

1. A WI-FI network, with internet connection.
2. A laptop or computer per group connected to the WI-FI, and with a text editor software installed.
3. An Android Smartphone (preferably from the students') per group, connected to the WI-FI, and with two apps installed:
   - The App Inventor app called "MIT AI2 Companion" which can be obtained at Google Play in this link.
   - A QR code reader app called "Barcode Scanner" which can be downloaded following this link.

   The Smartphone used in each group should have gyroscope and magnetometer. To know it, we recommend to install an app like Sensor Box or similar.
4. Recommended: a projector to show the TU material (App Inventor screen, multimedia resources, etc.) to all students in case it is necessary.

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:* To adjust the TU duration, we provide an App Inventor project as a template to focus the attention in AI aspects and not in programming ones.
   This template includes the graphic part of the app. It is named "appInventor_template_TU2.aia" and it must be stored in the computer of each group to be loaded from the App Inventor web application.
3. *QR Codes:* QR codes created with the names of the location points will be used in this unit. It is proposed to use the page https://bit.ly/3dweLWe to create the codes and to print them (recommended size 10x10cm).

7. Bibliography

App Inventor documentation: https://bit.ly/2Wt7Khr

QR code history: https://bit.ly/2ypFp3p

What sensors are in a smartphone?: https://bit.ly/2KZYTOW

Guidelines for interface design: Annex I

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 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. Challenge / Project

9.1 Final Objective

The ultimate goal of this TU is for students to learn in a practical way to distinguish the most relevant sensors and actuators used in AI. To do this, **they will have to develop a smartphone app using App Inventor that helps to follow the optimal path from one location to other inside the school**. Specifically, the app will guide users (visitor, teacher, student...) from their location to the destination they have selected, based on images and indications. We will call this app, the *School Path Guide*. The app will not contain all the AI topics explained in TU1, so some students maybe do not understand why it is an intelligent app. And it is not, so far. It is focused in one or two of those AI topics. The important aspect to explain here is that it is a first step towards future teaching units where we will develop complete AI solutions.

The *School Path Guide* is an app suitable for any type of educational center, and useful for the daily life of the entire educational community, but especially for those who do not know the center, i.e., visitors. It must work as follows: it is assumed that in the center there are different location points in different places (main hall, classrooms, library...) identified by a QR code, which has been coded with the location name. When the user arrives at one of these points, he/she scans the QR code through the app using the smartphone's camera. Once scanned, the app shows a list of possible destinations and the user selects the desired one. From this moment on, the app shows the optimal path to follow to reach the destination through photos and instructions displayed in the screen.

This app includes more AI topics than just perception and actuation, like representation or reasoning. Hence, it has been divided in two parts, which will be performed in two different units. In the current one, only the part that related to sensing and actuation will be carried out, so the *School Path Guide* will not be fully functional until the next TU will be finished.

Summarizing, in this TU, the app that students have to develop must perform the QR code scanning, the smartphone orientation sensing, and show such information to the user in a natural manner. In fact, the main feature that must contain the app is a **natural interaction with the user**, a very relevant topic of AI systems.

The app functioning is shown in the video called "TU2_APP.mp4" that is included in the TU resources. Student must watch it to understand the kind of solution they have to achieve. As

shown in the video, the application contains two different screens: a first one where the QR scan button is located, and a second one where the sensed information is displayed, including a compass. In addition, in the second screen, the user has a button where it can activate the voice guidance and perform speech interaction. The details of the app development will be explained in the following sections.

9.2 Activities

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 of the two screens, and each of them implies a different number of tasks. Fig. 2 displays a diagram that must be explained to students, and which summarizes the steps that must be followed to develop the program.

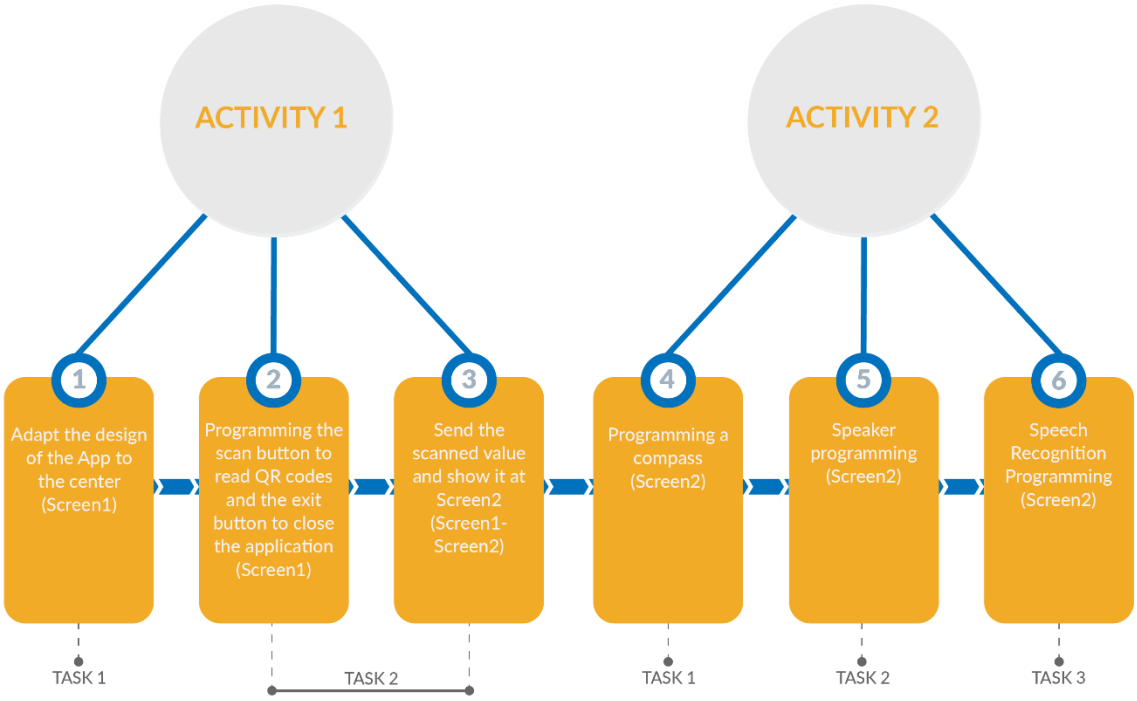


Fig. 2 Flowchart of the application development process

9.2.1 Activity 1

Goal: Programming the button actions of the first screen of the app and sending a variable to the second screen. To do it, the students will start with a project already created that includes the visual design of the screens.

Duration: 2 Hours.

Tasks: This activity has been divided into 2 tasks to be accomplished by students.

- ***Activity 1 – Task 1***

Goal: Change the UDC logo and the name "UDC Path Guide" to the specific school in Screen1.

Duration: 30 minutes.

The objective of this first task is just to load the template app so students can see the visual aspect of the app they will develop. The different elements included in the screens will be revised, and two of them must be modified to adapt the app to each specific school.

To start, each group must login in the App Inventor page (http://ai2.appinventor.mit.edu/) in their computer. The first time, some warning messages appear that must be closed. Once the main window is open, it is necessary to import the project template. To do it, they must click on "my projects" -> "import project (.aia) from my computer", as shown in Fig. 3, and select the file "appInventor_template_TU2.aia" provided with this TU.

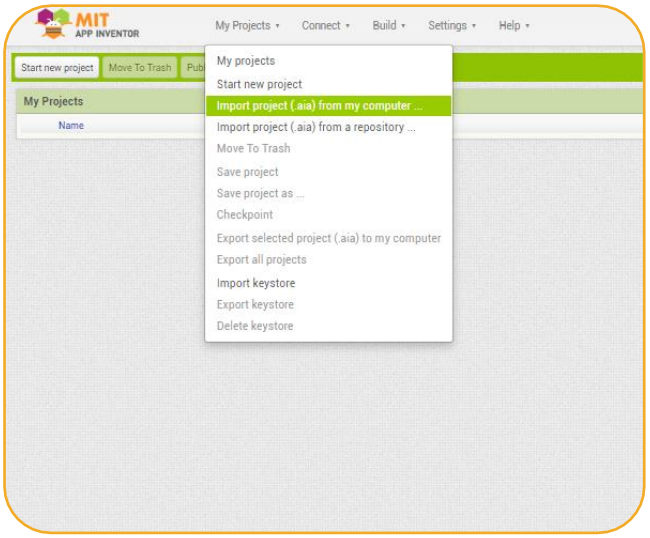


Fig. 3 Screen capture showing the menu to import the project template

If loading is correct, the screen displayed in Fig. 4 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". As usual in App Inventor, the screen shows the component palette on the left, the smartphone screen viewer in the center and, on the right, the list of added components and the properties of the selected one. The template contains the graphic elements with a basic design of the app and some pre-programmed blocks, which will be directly used or slightly modified to be applied in the TU.

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://appinventor.mit.edu/explore/ai2/setup. 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. 5 should appear.

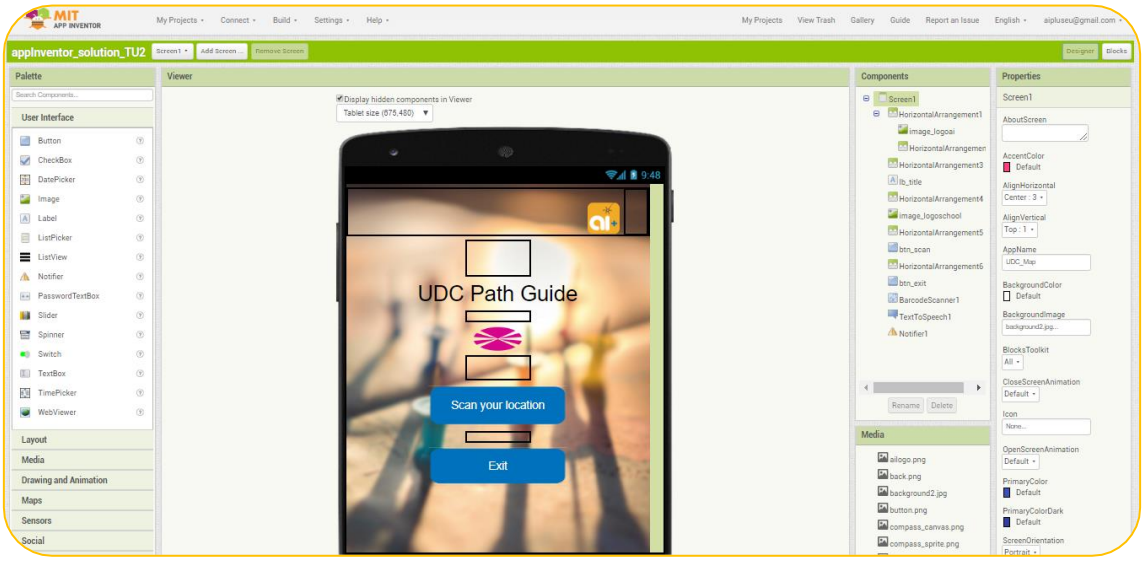


Fig. 4 Appearance of inventor app just after uploading the template

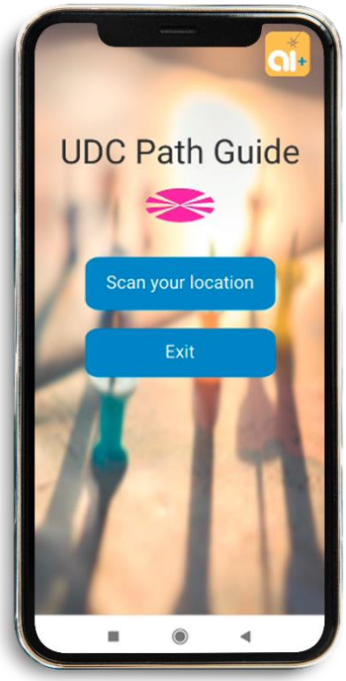


Fig. 5 Appearance of the application on the smartphone

The app design that can be seen in Fig. 4 includes, from top to bottom: an horizontal space with the AI+ logo on the right, a vertical space, a text box where you can read "UDC Path Guide", the UDC logo, a first button to start scanning, a second one to exit the app, and a final horizontal space. The visual aspect of the app (Fig. 5) is important, and students must know its relevance. Although the objective of this TU is not on creating user interfaces, the human-machine interaction depends on how the information is shown to the user, so **some basic guidelines are provided at Annex I**: Guidelines for interface design**.** It is recommended that students read this annex on their own so they can develop new interfaces in the future.

The task students must carry out in this task1 is simply to change the UDC logo by the one of their school, as well as the text “UDC Path Guide” for “*school name* Path Guide”. If they have developed the recommended App Inventor tutorials, it should be very simple for them, and no help should be required.

The following steps are include to guide the teacher in the correction: the first step is to modify the object properties in the “Properties panel” (shown in Fig. 6 and Fig. 7). To change the school name, it is necessary to select the component by clicking on the text label or by selecting it directly in the component panel (Label2). Then, in the right panel, all the properties of the component will be shown and it will be necessary to change the “Text” property, as shown in Fig. 6).

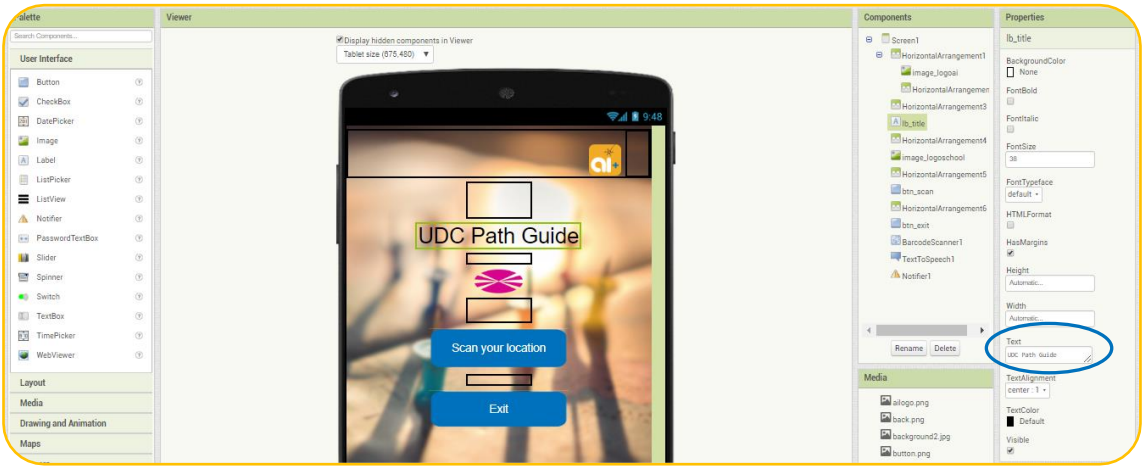


Fig. 6 Screen shot that shows the property of the label that must be changed to modify the text

On the other hand, to change the logo, the first step is to select the component (by clicking on the logo or by selecting “Image2” in the components menu) and upload the new logo in the “Picture” property. This logo should have the proper dimensions in order to maintain the same aspect as the original, so students should adjust both width and height.

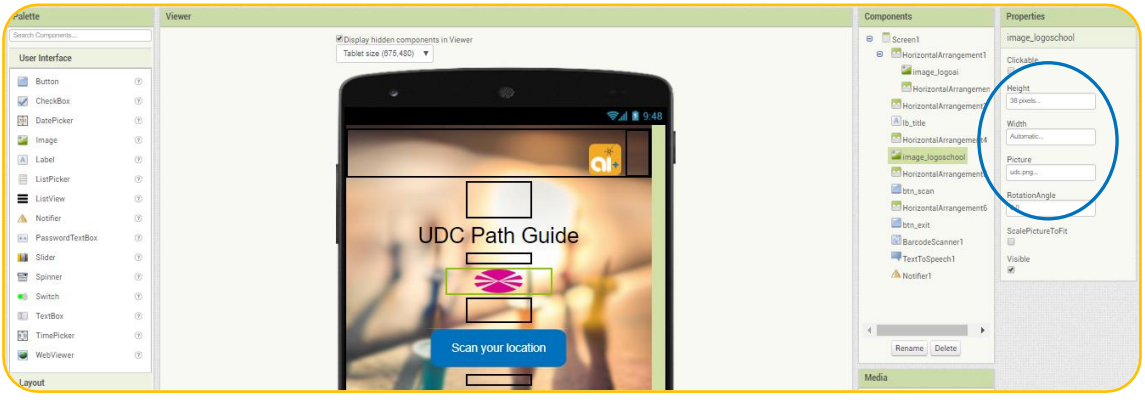


Fig. 7 Screen capture displaying the image properties that must be modified in order to resize and change the image

Fig. 8 displays a possible solution to this task with a new text and logo, although the original UDC logo and text will be used throughout the TU.

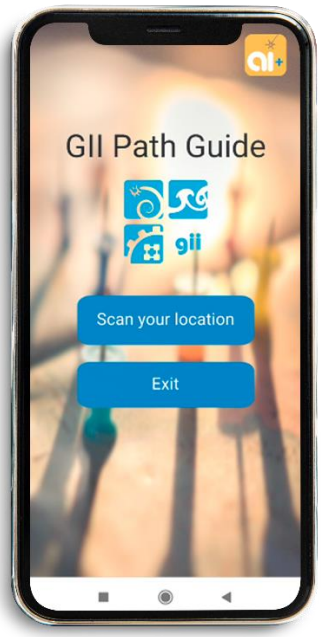


Fig. 8 Aspect of the app after changing the text and logo

- ***Activity 1 – Task 2***

Goal: Program the scanning button, perform the QR code scan, and send the read classroom name to Screen2. To check the successful implementation of this task, the scanned value must be displayed on Screen1 and Screen2 in text labels.

Duration: 1h. 30min.

To make the app functional, it is necessary to provide action to the buttons. Specifically, in the first screen there are two possible buttons: *Scan your location* and *exit* the application.

We can start with the *exit* button, because its programming is much simpler. The programming of this button action can be carried out by students autonomously. In what follow, we provide a step by step explanation just in case it is needed.

The first step is to know the name of this button (the text displayed over the button is not the name of the component) in order to access to its options. To do it, go to the design editor, click on the button and check the highlighted name in the components' menu (Fig. 9).

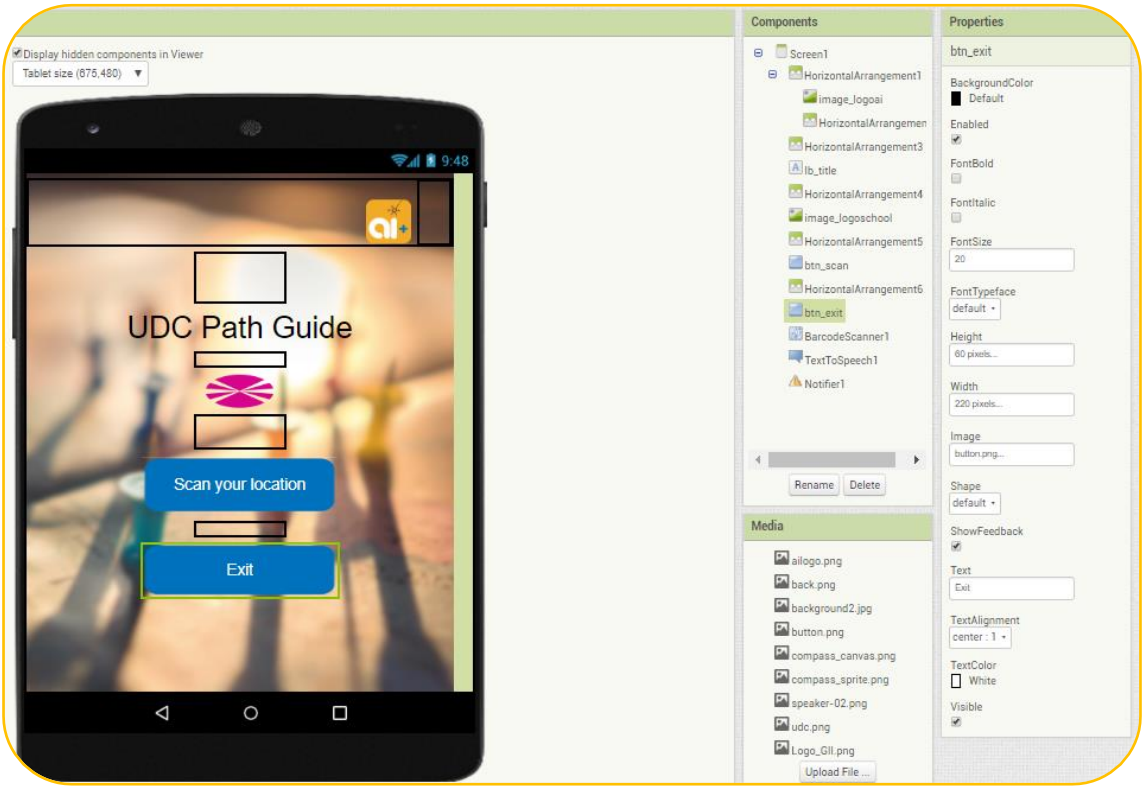


Fig. 9 Capture showing how to know the name of the *Exit* button

Now, if we click on the *exit* button in the block editor, it is possible to see that there are different options associated with this component (Fig. 10). In this case, what is needed is that when the exit button is clicked, the application is closed. Therefore, we must use the event handler that reacts to the click "when btn_exit.Click" (top one in Fig. 10).

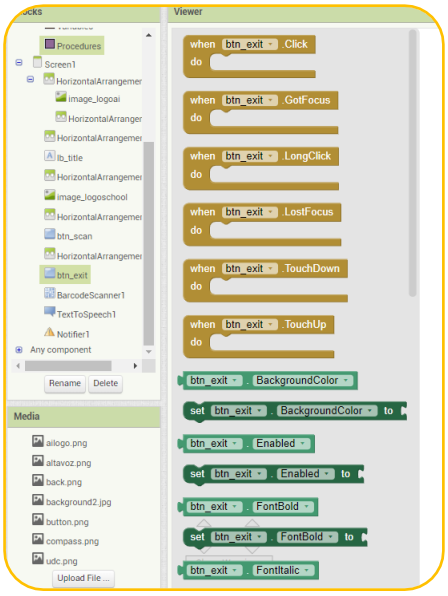


Fig. 10 Programming options of the exit button component

Once the event is known, it is necessary to add a block to close the application. To do it, we move to the control blocks (Fig. 11), select the "close application" one and insert it into the event handler as displayed in Fig. 12.

Fig. 11 Control programming options

Fig. 12 *Exit* button programming

Secondly, to program the *Scan your location* button, students must understand its functioning: when clicked, it should load the *barcodescanner* component of App Inventor, which opens a scanning app in the smartphone and performs the reading. The captured information must then be transferred to Screen2 to display the name of the location point where the user is placed.

At this point, students should revise some basic information about QR codes. To do it, we recommend using web resources and multimedia material as:

- https://bit.ly/3fjJJlu
- https://bit.ly/3b8kL5n
- https://bit.ly/35rv7vQ

Teachers should previously check these resources and select the most appropriate for their groups, or provide new ones better adapted. It is not required a deep knowledge of QR codes, but a proper understanding of what they encode and which information they provide.

Once revised, students should perform autonomously the basic programming of the *Scan your location* button as the result of this task2. To do it, they should apply the basic knowledge gained

by programming the *Exit* button to associate actions to buttons in App Inventor. Moreover, they must remember the basics of App Inventor on how to send values between screens.

A possible solution to this task is now provided:

When the button is clicked, the barcodescanner component of App Inventor must be called to perform the scan. In the blocks editor, if we click on this component of the Blocks panel, the different options associated to it may appear (Fig. 13). Specifically, in this case, only the first two blocks will be needed: the "when BarcodeScanner1.AfterScan" event handler and the "call BarcodeScanner1.DoScan" function call.

The scanner must be called first to launch the scanning app. This call will be made in the event handler of the *Scan your location* button. To do it, again, it is necessary to know the button name (in this case “btn_scanner”). Then, we must select the event handler of the component that responds to the button click (“when btn_scanner.Click”), and insert the scanner call on it (“call BarcodeScanner1.DoScan”), as indicated in Fig. 14**.**

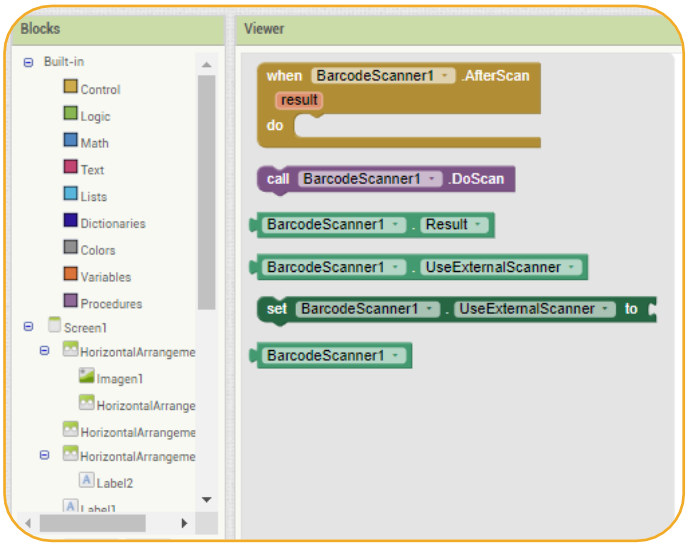


Fig. 13 Programming options of the BarcodeScanner1 component

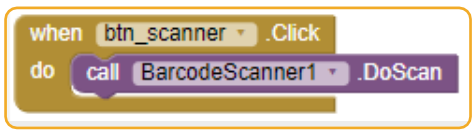


Fig. 14 This block calls the scanner when the *Scan your location* button is clicked

As a result, now the button has an associated action. When it is clicked, the QR scanning app opens, so the user can perform scan. Remember that, as established in section 6, this step will fail if a scanning app is not installed on the smartphone. To try scanning, we recommend that students create a QR code with a typical school place, like Main Hall, Classroom 1, Library, etc., so the reading process can be tested. In Fig. 15, an example of creation of a QR code on the website recommended in section 6 is shown. The code that appears there contains the name Main Hall.

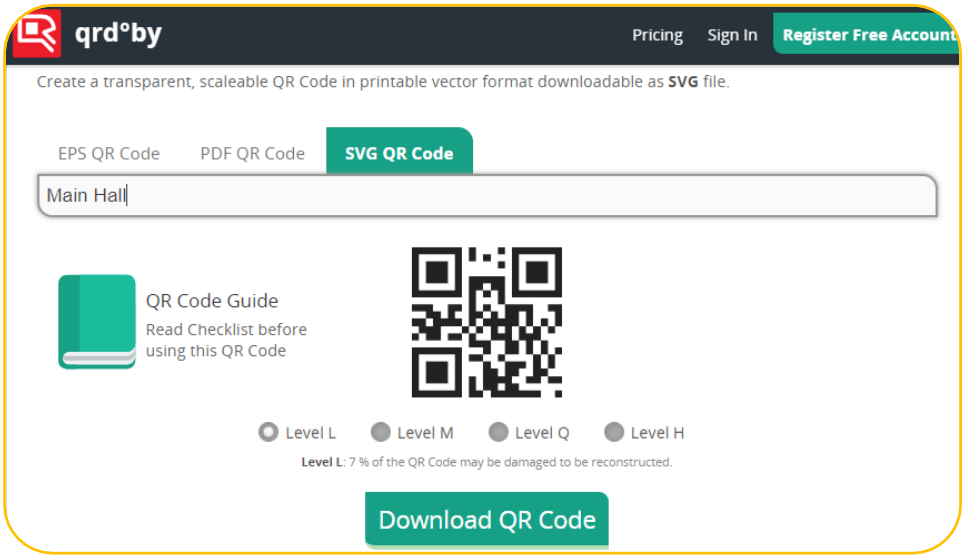


Fig. 15 Capture of the website where the QR codes are created

The next step in the program is to save the value read with the barcodescanner. The event handler "When BarcodeScanner1.AfterScan" should be used for this purpose. It is interesting that the students display the obtained value in the screen to better understand the information provided by this sensor. To do it, they should add a label to Screen1, as shown in Fig. 16, and load the result in it. Now, when students click on the Scan your location button, the read value will be displayed on this label, as shown in Fig. 17 for the QR code created in Fig. 15. Students can now create different QR codes with other texts in order to understand the type of textual information that can be used, which will be displayed in the text label.

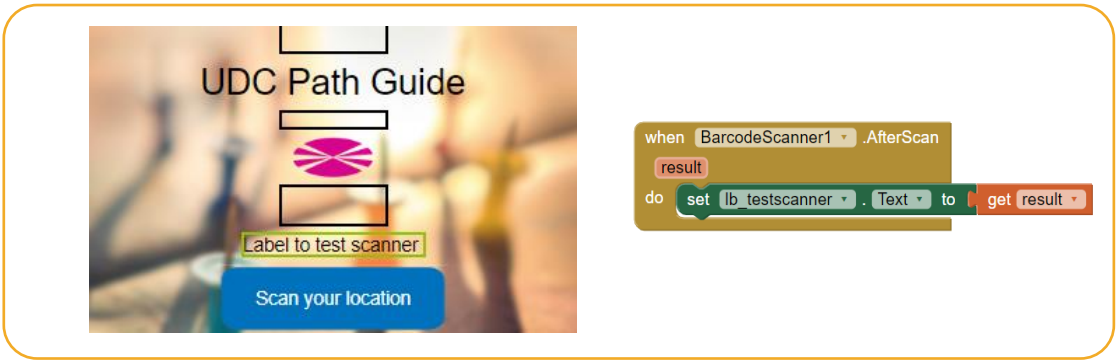


Fig. 16 Left: test label added to Screen1. Right: code needed to show the scanned information

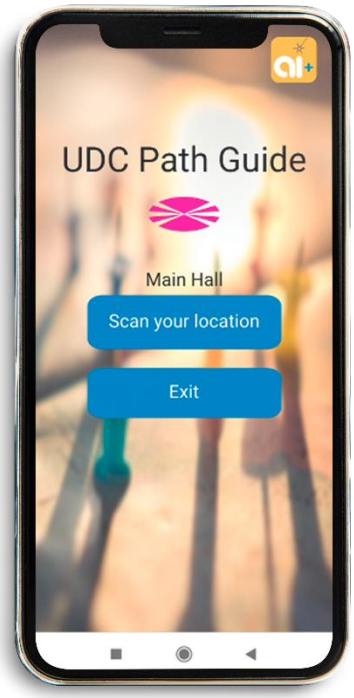


Fig. 17 Screenshot of the Smartphone after scanning QR. The read value is displayed above the *Scan your location* button

Now that the scanning process has been understood, the next step is to send the read value to Screen2 and store it in a variable to keep its value. To verify that is has been done properly, we recommend displaying the variable content in a text label, in a similar way as performed for Screen1. Programming this step requires changes in both Screen1 and Screen2. In Screen1, we will have to include the block "open another screen with start value...", in the "When BarcodeScanner1.AfterScan" event. Then, we must add the name of the screen we aim to open (Screen2), and send the scanned result as initial value (Fig. 18).

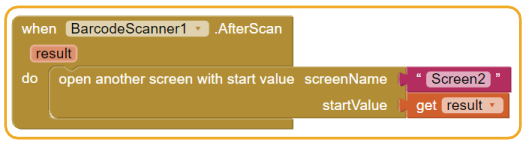


Fig. 18 Open a new window and send the scanned value

At this point, it is time to show how the Screen2 looks like to better understand what will be done. In Fig. 19 we can see both its format and the components that make it up. There are a total of 5 important elements (without considering the logo and the background image): a back button on the top left that allows the user to return to Screen1, a label on the middle where the user's location is represented (called *lb_origin*), a compass and a label below it where the smartphone's orientation is displayed (called *lb_azimuth*), and at the bottom, a button that, when clicked, performs 3 actions:

1. Notifies the user location by voice
2. Notifies the user orientation by voice
3. Asks the user if she/he wants to scan another location

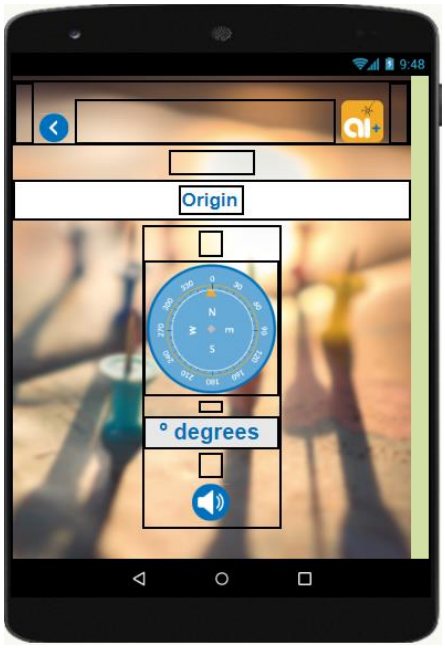


Fig. 19 Screen2 design capture

To capture on Screen2 the initial value sent from Screen1 in a new variable (named “origin” in Fig. 20), we can use the "get start value" block, which is located in the control blocks. Moreover, the variable content can be represented in the *lb_origin* label, which is shown when Screen2 is initialized, so it must be placed in the event handler "when Screen2. Initialize" (as shown in the Fig. 20).

Fig. 20 Code that stores the value sent from Screen1 in a variable and shows its in a label at Screen2

Now the scanning button can be tested again with some of the QR codes previously created. The result of this test should be like that displayed in Fig. 21 with the “EPS” text shown.

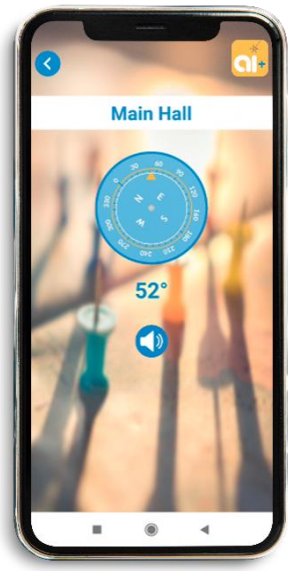


Fig. 21 Final result of Activity1

To return to Screen1 in a simple way, students should program the "back" button in the top left corner of this second window. They should be able to do this by themselves because it is very similar to what was developed to open Screen2, as indicated in the Fig. 22. Once the programming of the back button has been carried out, and it has been checked with several QRs that the application responds correctly, this task 2 can be evaluated as finished.

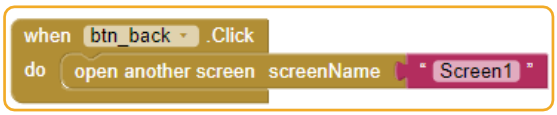


Fig. 22 Programming of the back button

9.2.2 Activity 2

Goal: To program the response of three elements in Screen2:

1. A compass that indicates the smartphone's orientation
2. A button to enable voice information of the user location and orientation
3. A button to enable speech recognition, so the user can say if she/he wants to scan a new location.

Duration: 2 hours.

Tasks: This activity has been divided into 3 tasks to be accomplished by students.

- ***Activity 2 – Task 1***

Goal: Programming a compass.

Duration: 1h. 20min.

Fig. 19 shows all the elements included in Screen2. One of them has been already programmed in Activity1, the central label showing the location name, and the others are the objective of the current activity. In this first task, we will focus on showing the orientation information in the compass image and in the text label that are placed in the center of the window (see Fig. 19).

Remember that the final objective of the smartphone app that is being developed in this TU and the next one, is to guide the user towards a destination in the school. For any guidance to be effective, it is necessary to provide the user with information regarding its orientation, so she/he can follow the right direction and sense. Outdoors, this type of information can be obtained from the GPS sensor, but indoors it is not available. The sensors we can use to obtain an equivalent data are the gyroscope and the magnetometer included in most of modern smartphones. With them, we will program a *compass* in our app, which will show the user direction relative to the geographic cardinal points. It is very important to remark here that **if the student's smartphone do not have these sensors this task1 cannot be carried out**. This is the reason why it was recommended in section 6.

At this point, students should learn the basics of how gyroscopes and magnetometers work. We recommend that they watch the following multimedia material, although many others could be selected by teachers:

How gyroscope works: https://bit.ly/35whjjq

How magnetometer works: https://bit.ly/3c3y346

App Inventor includes the information from these two real sensors in a virtual *orientation sensor*, which can be used to create our own compass. This sensor returns the following three measurements from the smartphone, in degrees (see Fig. 23 for a better understanding):

- *Roll*: 0 degrees when the device is level, increasing to 90 degrees as the device is tilted up onto its left side, and decreasing to −90 degrees when the device is tilted up onto its right side.
- *Pitch*: 0 degrees when the device is level, increasing to 90 degrees as the device is tilted so its top is pointing down, then decreasing to 0 degrees as it gets turned over. Similarly, as the device is tilted so its bottom points down, pitch decreases to −90 degrees, then increases to 0 degrees as it gets turned all the way over.
- *Azimuth*: 0 degrees when the top of the device is pointing north, 90 degrees when it is pointing east, 180 degrees when it is pointing south, 270 degrees when it is pointing west, etc.

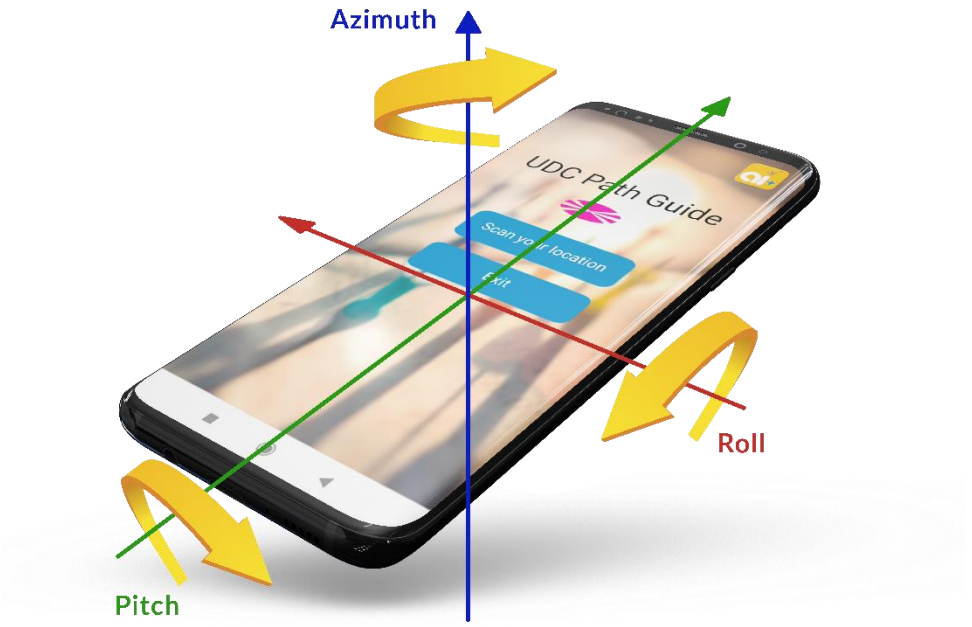


Fig. 23 Demonstration of roll, pitch and azimuth

To create a compass-like sensor, we just need to show the Azimuth information of the orientation sensor on Screen2. This can be carried out by students autonomously, but we recommend explaining them how to do it for the compass image, and then they solve the text label case:

In the blocks editor, if we click on the *orientation sensor* in the Blocks panel, we can see the different options associated to it (Fig. 24). The azimuth readings will be represented, and they must be refreshed each time the user moves, in other words, each time there is a change in the orientation sensor. Therefore, it will be necessary to use the first of the blocks "when OrientationSensor1.OrientationChanged".

The information will be displayed in the compass image on the middle part of Screen2, which must work as a real compass. Hence, it is necessary to understand how this type of instrument works in order to program it. We recommend students to observe the parts of a digital compass (for instance the one that comes with the smartphone, or other in a video), to realize that it is composed by two types of elements (see Fig. 25). Fixed ones, like the mark that indicates where the centre of the compass is and the arrow that shows where the smartphone is oriented, and moving ones, a wheel with the rotation degrees, and the cardinal points (N, S, E, W).

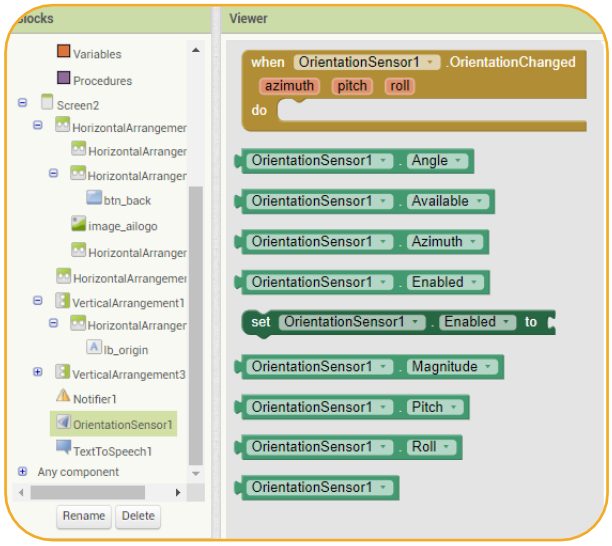


Fig. 24 Programming options of Orientation Sensor

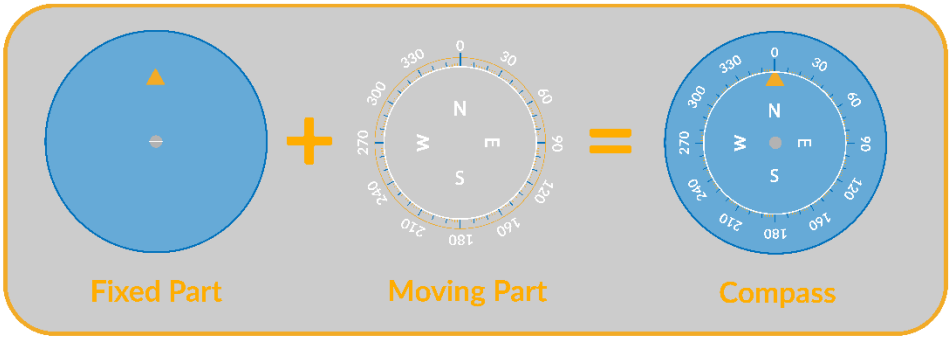


Fig. 25 Compass format

To represent these two parts properly in App Inventor, the fixed part will be loaded on a canvas and the moving part on a sprite, being this last the one where the orientation sensor data will be displayed. To program it, the important point is to understand that the sprite should always move to the orientation angle detected by the orientation sensor. As we can see in Fig. 25, the sprite component (which in this case has the name *ImageSprite1*), has many options, but here only the "set ImageSprite1.Heading to" will be used.

Students should copy the blocks shown in Fig. 27, understand what they do, and try them. When executed, the top of the sprite always points north, and the wheel value changes as the smartphone moves.

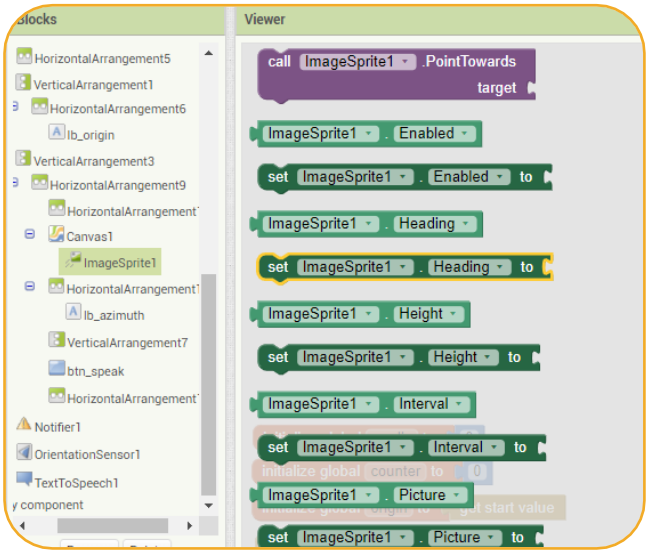


Fig. 26 Programming options of Sprite

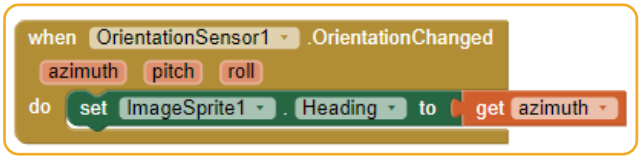


Fig. 27 Programming the compass Sprite

Starting from this code, students must be able to finish this task autonomously and represent the orientation degrees on the corresponding label as shown at Screen2 (named *lb_azimuth*), including the degrees symbol on the right of the reading, and without representing decimals digits. A solution to task1 program is displayed in Fig. 28.

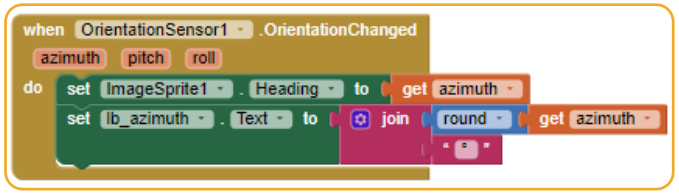


Fig. 28 Compass programming

To test if the students' implementation works properly, the value shown in the compass image and the one in the text label must be the same and change every time they move. Fig. 28 contains a screen capture of the app aspect with an orientation of 120º.

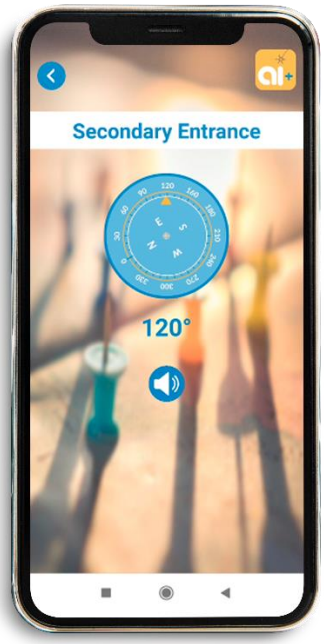


Fig. 29 Screen2 capture

It is possible to see that, although the compass works, it does not respond perfectly, because even if the mobile is not moving, the degrees always vary a little. At the level of realistic operation of this app it does not influence the result, but from a visual level, it is annoying and somewhat disconcerting. For this reason, an improvement of the compass response to make it more stable is proposed as a complementary activity for students (Compass Adjustment).

- ***Activity 2 – Task 2***

Goal: Programming the voice help button.

Duration: 20 Minutes.

As commented in the introduction of this TU, one of the most important topics in AI is that of human-machine interaction, and throughout the curriculum, students will learn its fundamentals. Regarding perception and actuation, the most important is to understand that a proper interaction with humans must be carried out using the sensors and actuators more similar to human ones. That is, a natural interaction with humans must use *cameras* to detect faces and objects as humans do. Moreover, it must use *microphones* and speech recognition techniques to detect sounds and words. Finally, AI systems should communicate with humans through *speakers* using speech and showing written information on *screens*.

Considering these aspects, in this task2 we aim to improve the app usability by including a button in Screen2 that enables voice information of the user location and orientation. To do it, we will use the smartphone's speaker. This is a well-known element for students, so we leave to the teacher's consideration to explain students the fundamentals of how speakers work. What is more interesting for students to understand in this scope is how voice synthesis works

(automatic text to speech). In the following resources, teachers can find fundamental concepts that we recommend students to check with attention:

- Text to Speech, how does it work?
- Text-to-Speech Technology: What It Is and How It Works
- Speech Synthesizers

To do it, it will be necessary to use the *TextToSpeech* component of App Inventor (Fig. 30). This component has been already included in the project template for students. As shown in Fig. 31, the component has several blocks, but in this case only the "call TextToSpeech.Speak" will be used. This block has one input called “message” in which we must add the text we want to transform into speech. With this information about the *TextToSpeech* component, students should program its response autonomously. An important point here to be explained to students is that **the text spoken to users must be clear, respectful and brief**, that is, the programmer must focus simply in transmitting the desired information to the user.

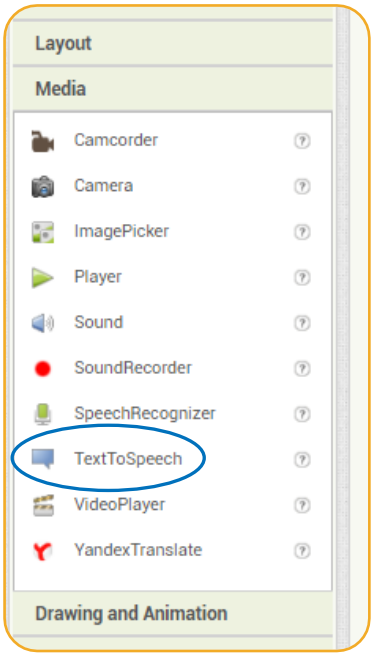


Fig. 30 Media Palette

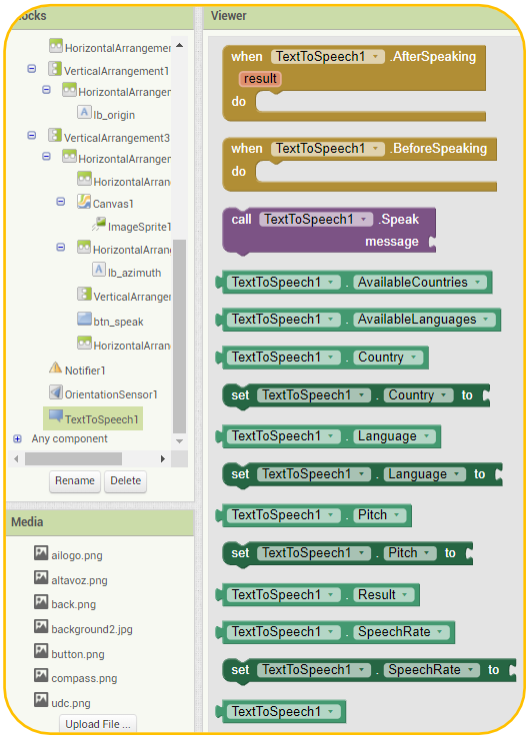


Fig. 31 Programming options of Text To Speech

In what follows, we include a possible solution just in case it is necessary:

The module call must be placed into the event handler "when btn_speak.click". The spoken message should include the user location (saved in the *origin* variable) and orientation (using the OrientationSensor1.Azimuth block which is located between the Orientation Sensor blocks). It is important to highlight here the relevance of using variables to store perception data, because now we can access to them without needing to read them again. Since we are dealing with two different variables, it is necessary to use the join block in order to put them together, as shown in Fig. 31. In addition, it can be observed that we have added the text “You’re in… and you are oriented towards …”, with the aim of communicating with the user in a simple and straightforward way.

To test the student’s implementation is very easy: each time the speaker button is touched, the smartphone “says” the location (last QR read) and the orientation, which must be the same value that is displayed in the compass. It is important to check the message spoken to the users avoiding inappropriate texts.

Fig. 32 Voice help button programming

- ***Activity 2 – Task 3***

Goal: Programming the speech recognizer.

Duration: 20 Minutes.

With the same objective of the previous task, in this one we aim to improve the app usability and the natural interaction with the user by including speech recognition, so the user can speak to the app to return to start scanning again. The *SpeechRecognizer* component of App Inventor takes the smartphone's microphone to capture the user voice and then it transforms this audio into text that can be used in the program. At this point, teachers can introduce the basics of microphones to students in case they consider it interesting for them, although it is a well-known sensor and most of them are familiar with it. What is relevant here is that students understand how the speech recognition process works, so the following material should be checked carefully:

- Behind the Mic: The Science of Talking with Computers
- Speech recognition software

The *SpeechRecognizer* component, shown in Fig. 33, has many associated blocks. In this case, we only need the "call SpeechRcognizer1.GetText" and the "when SpeechRecognizer1. AfterGettingText". The program that students must develop should provide the following response: each time the user touches the speaker button programmed in task2, the app says the location and orientation. Then it will ask the user by voice if she/he wants to scan a new location. If the answer is YES, Screen1 should be opened. If it is NO, nothing happens, and the app remains in Screen2. Now, students should be able to program this response autonomously.

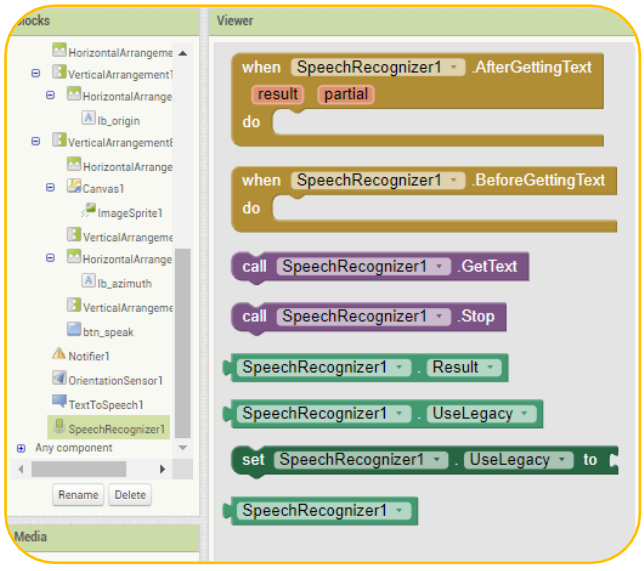


Fig. 33 Programming options of Speech Recognizer

A possible solution is provided in what follows: the first step is to modify the TextToSpeech component programmed in the previous task, adding a text to ask the user if she/he wants to

scan a new location. This can be useful, for instance, to know the way back to the origin (the practical use of the app will be clearer in the next TU). It is important to highlight again to students that the question to be pronounce by the app to be clear and simple. This modification is shown in Fig. 33.

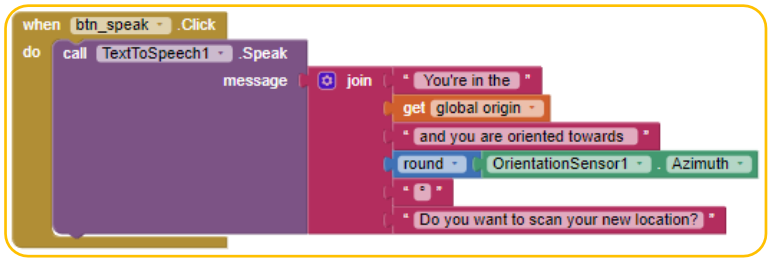


Fig. 34 Speak button programming with question to the user

Following the previously explained logic of the app response, once the question has been spoken to the user, the SpeechRecognizer call must be executed. If we put it right after the TextToSpeech block, it will not wait for the question pronunciation to finish, leading to a wrong functioning. To solve this problem, in the TextToSpeech component, there is an event handler (as you can see in the Fig. 31), which is executed when TextToSpeech finishes (when TextToSpeech1.AfterSpeaking), and it must be used here, as show in Fig. 35.

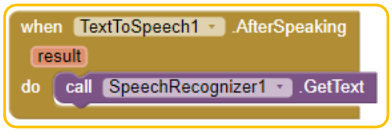


Fig. 35 Calling the voice recognition function

This “trick” can be previously explained to students or leave them find the problem, try some solutions, and then provide them with the right one.

Once the SpeechRecognizer call has been implemented, the last thing to do is to check the captured word and execute the proper action. This can be performed using the event handler "when SpeechRcognizer1.AfterGettingText" and a conditional, as shown in Fig. 35. If the result is YES, Screen1 should be opened (Fig. 36), simply by using the “open another screen” block. If the result is NO, nothing has to be done, so this option is not included in the conditional.

To check the students’ implementation of this task3 is simple. After saying the user location and orientation, the app should ask if the user wants to scan another QR and return to the Screen1 if the answer is YES.

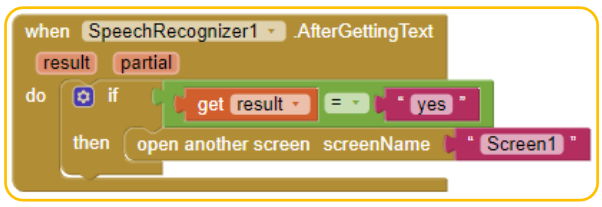


Fig. 36 Last step after voice recognition

10. Evaluation

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

Assessable activity	Score
1- Program test	30%
2- Kahoot	10%
3.1- Group report 3.2- Checklist	30% 30%

1. *Final test of the program:* at the end of the TU, each group must show the operation of the developed app to the teacher. To do it, we recommend using different QR codes with real school places, like the Main Hall or the Library, although it is not required to fix them to the real places for this TU. The following features should be verified by the teacher:

    - The scanning and exit button work properly.
    - The compass works properly.
    - If the speaker button is touched, it says both the user's location point and orientation.
    - When the previous speech finishes, the app asks the user by voice if he/she want to scan a new location, captures the answer and performs the corresponding action.
    - The general app functioning is correct, with no stops or pauses.
    - The human-machine interaction aspects are correct: clear and direct sentences, adequate positioning of buttons, etc
    - OPTIONAL: if the group has performed optional activities and the app has been improved, it should be considered in this part of the qualification.

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*: at the end of the TU, the **students must fill a kahoot survey** individually (Annex II: Student's survey).

3. *Ongoing work during the TU*. This methodology is very important, and it will be evaluated using two different elements:

    3.1 Individual rubric that the teacher must fill for every activity (Annex III: Teacher rubrics).

    3.2 Individual work report (Annex IV: Student's worksheet), which will be filled by students throughout the working sessions.

11. Complementary activities

Error Control

It is important to include an error control logic to the app in order to verify that the scanned QR contains a valid school place name, because in other case, the app will not be capable of guiding the user. It is always recommendable to develop a reliable smartphone app, which can be used in realistic conditions. We recommend to provide students with basic guidance to solver this activity. Basically, to develop the error control, it is necessary to create a list type variable in which the names of the school places are stored, and modify the event handler "BarcodeScanner1.AfterScan" to include a conditional that checks if the read value corresponds to one of these names, showing a warning message in case of unmatching.

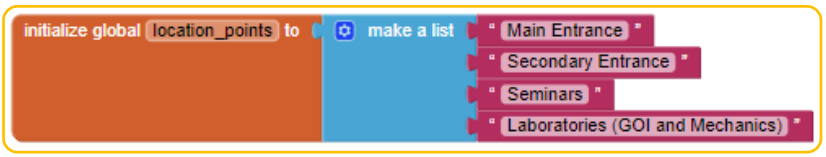


Fig. 37 Initialization of the variable location_points as a list

In what follows, a possible solution to this activity is provided:

Fig. 37 shows how to create the list type variable with the different location points, in this case corresponding to a UDC building. On the other hand, Fig. 38 contains the modifications that must be performed to the solution of Activity 1, displayed in Fig. 18. We have added a conditional in the event handler "when BarcodeScanner1.AfterScan". To check if the scanned value is the list, the "is in list?" block has been used. If it is true, the original code included in the solution of Fig. 18 is executed. If it is false (the read name is not in the list), the user is notified with a message on Screen1 and by voice.

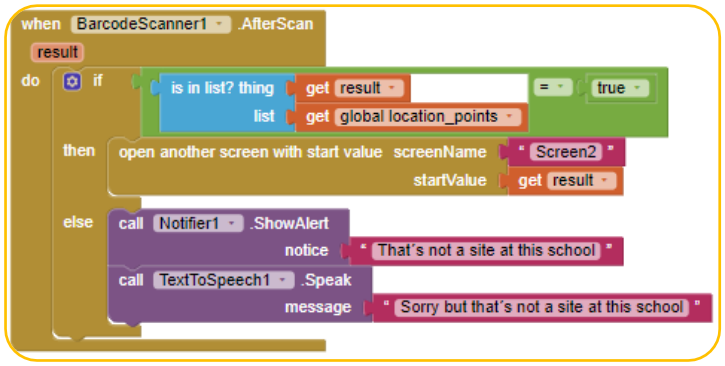


Fig. 38 Possible solution to open another window or report an error with the scanned code

Finally, regarding the programming to be performed in Screen2, nothing changes by including this error control.

Compass Adjustment

In Activity 2 – Task 1, the compass was programmed in such a way that, every time the sensor detects a change of orientation, it is represented, which causes the compass to be in continuous movement and can even confuse the user. In this activity, it is proposed to adjust the compass response to make more stable. This instability occurs because, on the one hand, the sensor is not very accurate and, on the other, it makes many readings per second with small variations between them (around ±5°).

There are many ways to solve the problem, such as computing the average between the measurements obtained through time and representing. But, here, we have decided to update the compass value only when it changes notably (for instance, bigger than 5º), avoiding thus the continuous change of values. To do it, it is necessary to compare two values: the current azimuth value and the previous one. If the difference between them is less than 5, the compass should continue as it is, and when it is greater, the new position should be loaded. This is the task that should be proposed to students.

To program this functioning, it is necessary to create a variable in which the value of the azimuth that is represented will be saved (in this case it has been decided to call it *represented_azimuth* as shown in Fig. 39). Moreover, a conditional must be included to check if the difference (in absolute value) between the two values is greater or smaller than 5. When the condition is true, it will be time to load the compass position, represent the degrees in the label (without decimals and with the degree symbol), and save the azimuth value in the created variable, as displayed in Fig. 40.

Fig. 39 Initialization of the represented_azimuth variable

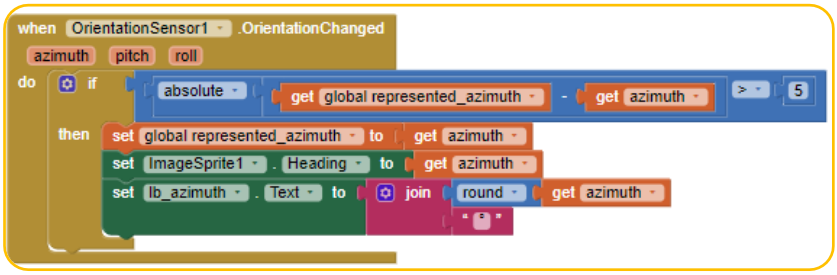


Fig. 40 Possible compass solution

With this program, in the first execution, the value of the variable *represented_azimuth* will be 0 (see initialization in Fig. 39) so, once the orientation is a value higher than 5, the condition will be true, and the reading will be displayed as commented above. From this point onwards, the variable *represented_azimuth* will store the azimuth value that is represented in the compass, and it will be updated only if the new value differs in an amount of 5. To check the student's implementation, just evaluate if the compass response is more stable but still right.

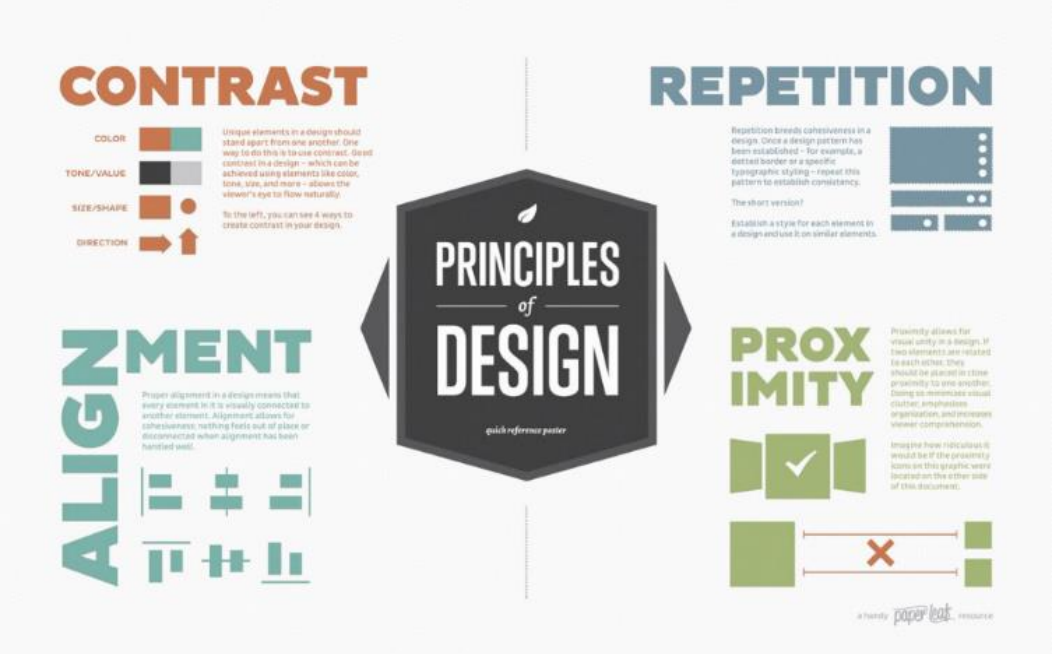
12. Annex

Annex I: Guidelines for interface design

The main guidelines students should know about user interfaces are:

- The brand identity will give us the base colors and typographies.
- The main logos should be placed on the top of the screen.
- The position and size of the buttons should allow the user to see them clearly and to tap them with a finger. Avoid using undersized or oversized buttons and a large number of them in a single screen.
- The colors are important: they should be representative of the app, but not too intense. Colours help us to priorities texts and also highlight phrases or words that can be clicked on, such as links. The background colour should be in line with that chosen for the typography, as a minimum contrast is necessary for legibility and accessibility.
- Minimalist design: The design should not be overloaded with information, the user is looking for a clean app that loads and run quickly.
- Recognized standards: We should remember certain basic standards. For instance, the magnifying glass icon relates to "Search". Likewise, there are colors called: "reserved colors" which use should be limited, as red, which is used for errors or alerts and green for confirmation and success messages.

These are some basic hints to create a good design and ease human-machine interaction. One of the most useful is having a look at the apps on our phone and try to recognize common elements (logos positioning, typography size, the colors based on your corporate identity, using less than 3 colors, etc.).



The image above is representative of the basic design principles. Below there are three links to videos that we recommend you to watch. They are short and enjoyable and will help you to get a sense on the importance of the composition, typography and colour.

- Layout & Composition: https://bit.ly/39qDFDz
- Typography: https://bit.ly/2QVElKy
- Color: https://bit.ly/3avl4aS

Annex II: Student's survey

Teacher can enter the kahoot in the following URL: https://bit.ly/2Us2tGW

Other option is to create a new, using the following questions (correct answer is underlined):

1. Is a QR code scanner a sensor?
   - Yes, it returns sensed information
   - No, it is an app, the sensor is the camera
2. A QR code can contain as much text as desired
   - No, it is limited
   - Yes
3. Voice synthesis is relevant in AI because
   - It simplifies interaction with users
   - It shows information to the user in natural way
   - All answers are correct
4. The tactile interaction with LCD screen in AI is
   - Not really relevant, in the future the interaction will be through voice and image
   - Important, because touching is natural for humans
5. The graphical design of the app screen is
   - Not very relevant
   - Important to simplify user interaction
   - It´s the most important thing
6. The smartphone screen is an actuator?
   - No, it is just part of the system
   - Yes, because it performs an action, showing information
7. The orientation sensor of App Inventor uses
   - The gyroscope
   - The magnetometer
   - Both
8. Speech recognition uses a specific phonetic dictionary for each language
   - False, it works in any language
   - True, to match the recorded sounds with existing words

Annex III: Teacher rubrics

Level (score) / Aspects to be evaluated	Expert (4)	Competent (3)	Partially competent (2)	Not yet competent (1)
Adequate selection of information (QR code creation, 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 IV: Student's worksheet

This worksheet must be filled by each student during the completion of the TU, at the end of each session:

<table>
<tr><th colspan="5">GROUP REPORT TU 2 – ACTUATION AND PERCEPTION</th></tr>
<tr><td>Students</td><td colspan="4"></td></tr>
<tr><th colspan="5">ACTIVITY 1</th></tr>
<tr><td>Time Enough</td><td>☐ Yes</td><td colspan="2">☐ No, It´s necessary ……</td><td>Challenge completed?
☐Yes ☐No</td></tr>
<tr><td>Contents learned</td><td colspan="4"></td></tr>
<tr><td>Difficulties</td><td colspan="4"></td></tr>
<tr><th colspan="5">ACTIVITY 2</th></tr>
<tr><td>Time Enough</td><td>☐Yes</td><td colspan="2">☐No, It´s necessary ……</td><td>Challenge completed?
☐Yes ☐No</td></tr>
<tr><td>Contents learned</td><td colspan="4"></td></tr>
<tr><td>Difficulties</td><td colspan="4"></td></tr>
<tr><th colspan="5">AI ASPECTS</th></tr>
<tr><td>Sensors used</td><td colspan="2"></td><td>Actuators used</td><td></td></tr>
</table>