TEACHING UNIT 3

Introduction to Representation and Reasoning 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

Representation and reasoning are two very relevant topics in AI, although they are complete new for students in high school degree. They are both related with processes that take place inside the computational system that makes up the artificial intelligence. Representation is related to how the perceptions of the real world can be encoded and stored in the computer, so we can use them to create intelligent programs. There are many ways of representing perceptions, and it must be done carefully to avoid that our programs become too complicated.

On the other hand, reasoning is the process of selecting the most appropriate action to send to the actuators of our intelligent system in each instant of time. Reasoning responds to the question: What should the AI system do now to reach its goals? Our program has to consider many variables to perform this selection: the current perception, the motivation (goal), the history of past perceptions (memory), the set of possible actions, etc. Depending on the intelligence level we aim to obtain with our AI system, selecting the best action can be very easy in simple situations but very complex in many others.

Perception and reasoning can be studied together or as independent topics, which is the approach we will follow in this curriculum. But, to introduce them, we have decided to work with both in the same practical case. This way, students will understand how a proper representation facilitates reasoning. The objective of this TU is to provide a first approach to these topics through a simple but illustrative case, leaving a more detailed analysis for future TUs.

Specifically, students will develop here a smartphone app that acts as a *path guide* for their school. It will be an improvement of the app developed in the previous TU, and it will allow any visitor to reach a target destination in the school in an easy way, simply by following a route over a map created by students. The AI elements related to perception and actuation presented in the previous TU, will be used here too.

As we will see at the end to the TU, step by step, the apps developed by students will be more "intelligent" as more topics are introduced. In this case, the user will perceive that the app is able to suggest the optimal route in an autonomous way, reasoning over the current location and the target one.

2. Context

In order for the student to adequately meet the learning objectives of this TU, he/she must have the following prior knowledge:

- **Programming**: students must have experience in block-based programming, creating simple algorithms based on conditionals and loops.
- **Basic knowledge of MIT App Inventor:** students should have basic knowledge about the App Inventor application (https://appinventor.mit.edu).
- **Mathematics:** 2D representation of simple maps, floorplans and cartesian coordinates.

- **Perception and actuation in AI:** it is mandatory that students have finished TU2 and they know the basics of perception and actuation in AI.
- **Previous knowledge:** it is recommendable that students watch the following videos before starting the TU, to understand how data is stored and represented in a computer:
  - https://youtu.be/TQCr9RV7twk
  - https://youtu.be/KN8YgJnShPM

3. Learning objectives

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

SPECIFIC

- Fundamentals of computer image representation
- Introduction to topological and metric mapping
- Fundamentals of graph theory
- Introduction to rule-based reasoning
- Reinforcement in human-machine interaction principles

TRANSVERSAL

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

4. Contents

To start working with representation and reasoning in AI, students will develop a smartphone app using App Inventor. The following specific contents will be studied:

1. 2D representation of the environment: introduction to topological and metric maps
2. First view representation of the environment
3. Computer image representation
4. Basic graph definition
5. Optimal route definition over graphs
6. Path following

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 the first activity has been organized into two small tasks, as shown in the scheme below.

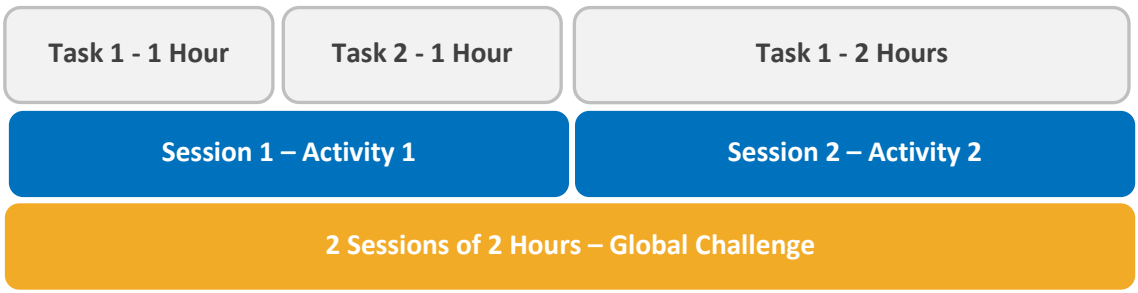


Fig. 1 Temporary organization

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.
3. An Android Smartphone (preferably from the students') per group, connected to the WI-FI, and with two apps installed:
   - The App Inventor app
   - A QR code reader app
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, and some pre-programmed blocks, so it is possible to develop the application in the available time. It is named "appInventor_template_TU3.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. We recommend teachers to read it.

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 web https://bit.ly/3dweLWe to create the codes and to print them (recommended size 10x10cm). **The QR codes used in this unit are the same of those used in TU2, so there is no need to create them again**.

7. Bibliography

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

Knowledge representation: https://bit.ly/2Bg9zXL

Computational thinking: https://bit.ly/2MZWAfL

8. Groups

The project has been designed to be carried out in groups of 2 students. Each member of the team will have a different 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

Students 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 (photos taken by students) and text indications. We will call this app, the *School Path Guide*, and it will be focused in AI representation and reasoning from a simple perspective.

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 should 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 encoded 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.

In the previous TU, we have developed the first part of this app, focused on the perception and actuation parts. Specifically, such version implements the QR code scanning, uses the compass so that the user can orient herself/himself, includes a button through which the voice guidance is activated and performs speech interaction. Although students could carry out the app improvement from their previous project of TU2, it is advisable to start from the template app provided with this TU, so all the class starts from the same point, and the evaluation can be independent from the previous work. In addition, the template includes the programming of some elements in the app that may require too much time.

The app functioning is shown in the video called "TU3_APP.mp4", included in the TU resources. Students 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 which is already programmed in the template, and a second one where the user can select his/her destination and where, once selected, the path to be followed is shown by means of images and text indications. In addition, the second screen contains the compass that provides the orientation and different buttons to advance or return to the previous step in the route.

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 topic, and the first one implies two different tasks. Fig. 2 displays a diagram that must be explained to students and which summarizes the steps that should be followed to develop the program.

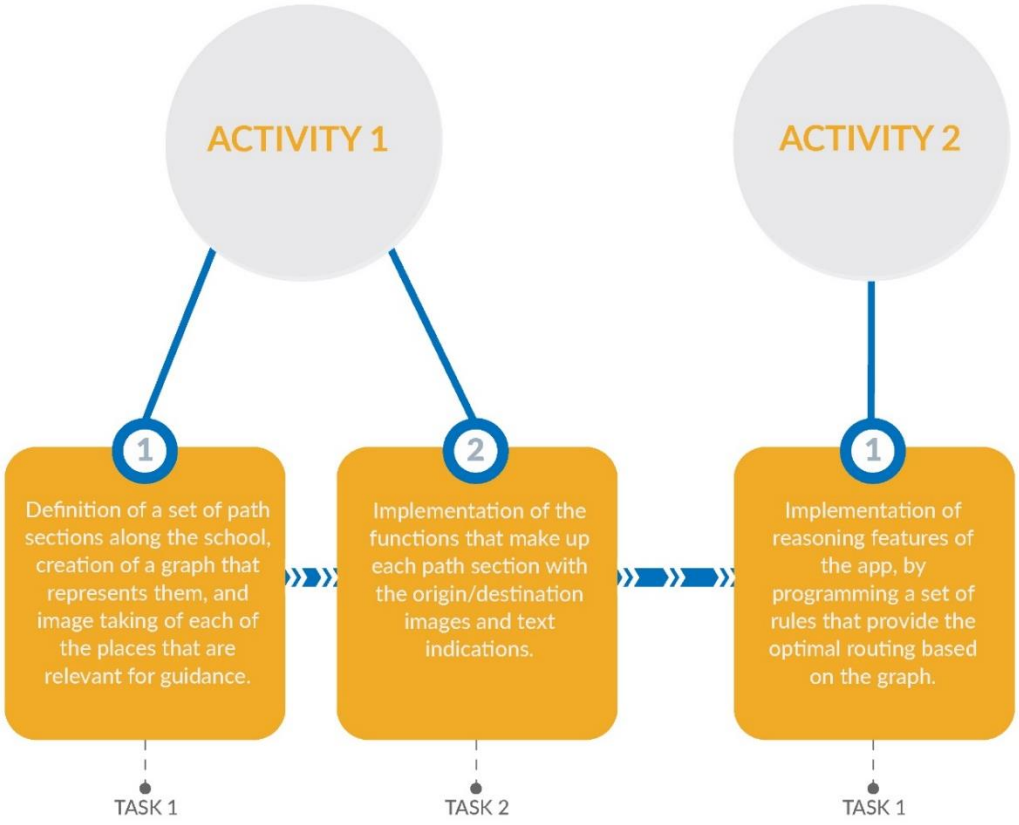


Fig. 2 Flowchart of the application development process

9.2.1 Activity 1

Goal: Map representation

Duration: 2 Hours

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

- ***Activity 1 – Task 1***

Goal: define the path sections in which it is necessary to divide the school map for a correct guidance, create the graph and take the pictures of the relevant points.

Duration: 1 Hour.

The objective of the *School Path Guide* app is to provide guidance to the user so she/he can move around the school without getting lost and can reach her/his destination using the optimal route. To this end, the first step is to define the representation of the school that can be used in the app, so later we can show a path based on it. One of the simplest types of representation for a building is a floorplan, which provides a plan view, typically covering a wide area (here we assume that all students have basic knowledge about 2D maps and floorplans, in other case, an introductory explanation should be provided by teachers). Performing a useful path guidance through a floorplan representation implies knowing the user location to show its position as it moves, but indoors it is not easy (outdoors we could use GPS, as, for instance, Google maps do). Therefore, as a first approach, we propose to use other representation of the school based on *images*, that is, photographs of locations the user can easily identify. This is a “first person” view of the school that covers a more reduced field of view, but it allows us to show the path to follow in a simple way, just by creating an ordered sequence of steps, as it was shown in the video "TU3_APP.mp4". Thus, the user location is not so relevant because he/she can know his/her position by watching the images and comparing them with the real world.

At this point, it is recommendable to explain students the basics of digital image capture and storage. Teachers can find many references about this topic. Here we propose some that could be checked directly by students:

- Images, Pixels and RGB
- Fundamentals of image representation

And some more recommended for teachers:

- Digital Imaging Tutorial
- Introduction to digital images

The digital representation of an image through pixels is automatically performed by the smartphone operating system every time a photo is taken. What is important here is to highlight that, for our app, the pixel-based representation is not the relevant one, because we will not perform the path guidance over the pixels of the image, but over a simpler representation based

on relevant points in the school, which are identified with these images. To understand this type of representation, teachers should review and then explain to students the differences between topological and metric maps:

- The *metric mapping* is the most common for students and considers a two-dimensional space in the objects are placed with precise coordinates, for instance, cartesian coordinates or geographic (Fig. 3 right). We will use metric maps later on in this curriculum, when dealing with robotic coordinates mapping.
- The *topological mapping* only considers relevant places and relations between them. Often, the distances between places are stored too. The map is then a graph, in which the nodes corresponds to places and arcs correspond to the paths. An example of topological map is the London Tube (Fig. 3 left).

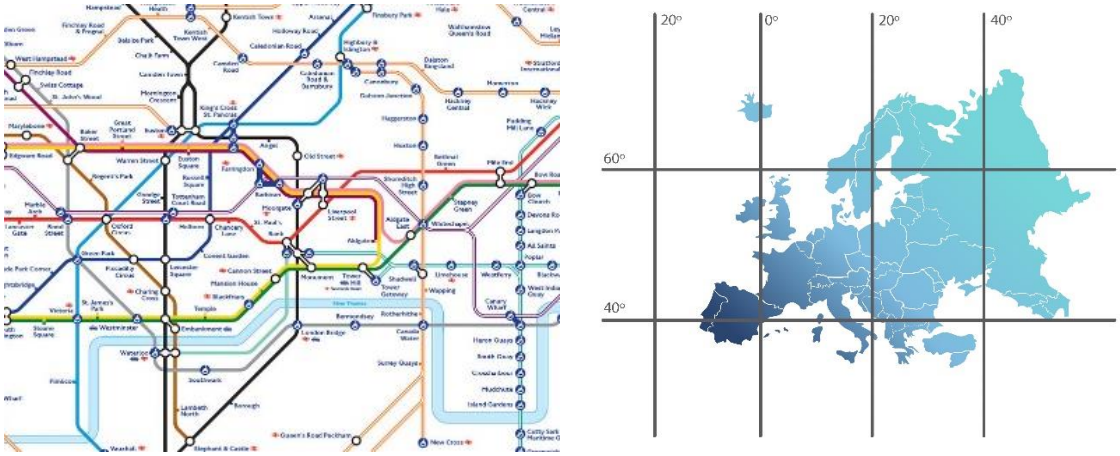


Fig. 3 Left: Topological mapping. Right: Metric mapping

In this TU, *we will use a simple topological map based on a set of location points* in the school. The first step for students is to define these *location points*, which will be marked with a QR code placed in a wall at a visible position. These location points make up the possible origins and destinations of the users in the school. They will appear listed in the app and it is very important to select them carefully. It is recommended, for the first version of the app, to define a maximum of **4 points distributed in two floors** (main and first floor).

With this set of 4 location points, the maximum number of possible routes is 12, corresponding to each location point towards 3 possible destinations. One could define 12 different paths and create 12 different sequences of images to represent them, but there are many sections that are coincident, mainly in small building like a school. Thus, to simplify programming, it has been decided to create a division of the school into *path sections*. They are defined by **crossing points**, which are relevant points in the school where more than one path coincides, or by location points. Consequently, the possible routes are the result of the union of path sections.

To clarify this representation, we describe now a specific example, so teachers can understand the type of path section and location point that students should define:

We will use a building in the UDC as the target case to develop the *School Path Guide* app. It is a three floor building, shown in Fig. 4, but only two of them (ground and first floor) will be considered in this first version. A floorplan of the building is shown in Fig. 5.

Fig. 4 Exterior appearance of the UDC building used for the example

We have defined the 4 location points in the following places (marked in blue colour in Fig. 5):

1. Main entrance (ground floor)
2. Secondary entrance (ground floor)
3. Seminars (first floor)
4. Laboratories (first floor)

As a consequence, the following possible routes arise:

1. Main entrance → Secondary entrance
2. Main entrance → Seminars
3. Main entrance → Laboratories
4. Secondary entrance → Main entrance
5. Secondary entrance → Seminars
6. Secondary entrance → Laboratories
7. Seminars → Main entrance
8. Seminars → Secondary entrance
9. Seminars → Laboratories
10. Laboratories → Main entrance
11. Laboratories → Secondary entrance
12. Laboratories → Seminars

But if we take a look at them using Fig. 5, we easily detect that they share common points and sections. This is why we propose to define path sections to organize and simplify the routes. To do it, we have to identify crossing points, that is, new location points in the building that are relevant because different routes cross them. In this case, these crossing points are:

1. Main Stairs

2. Secondary Stairs
3. First Floor Entrance
4. First Floor Secondary Entrance

They have been represented in Fig. 5 with orange circles.

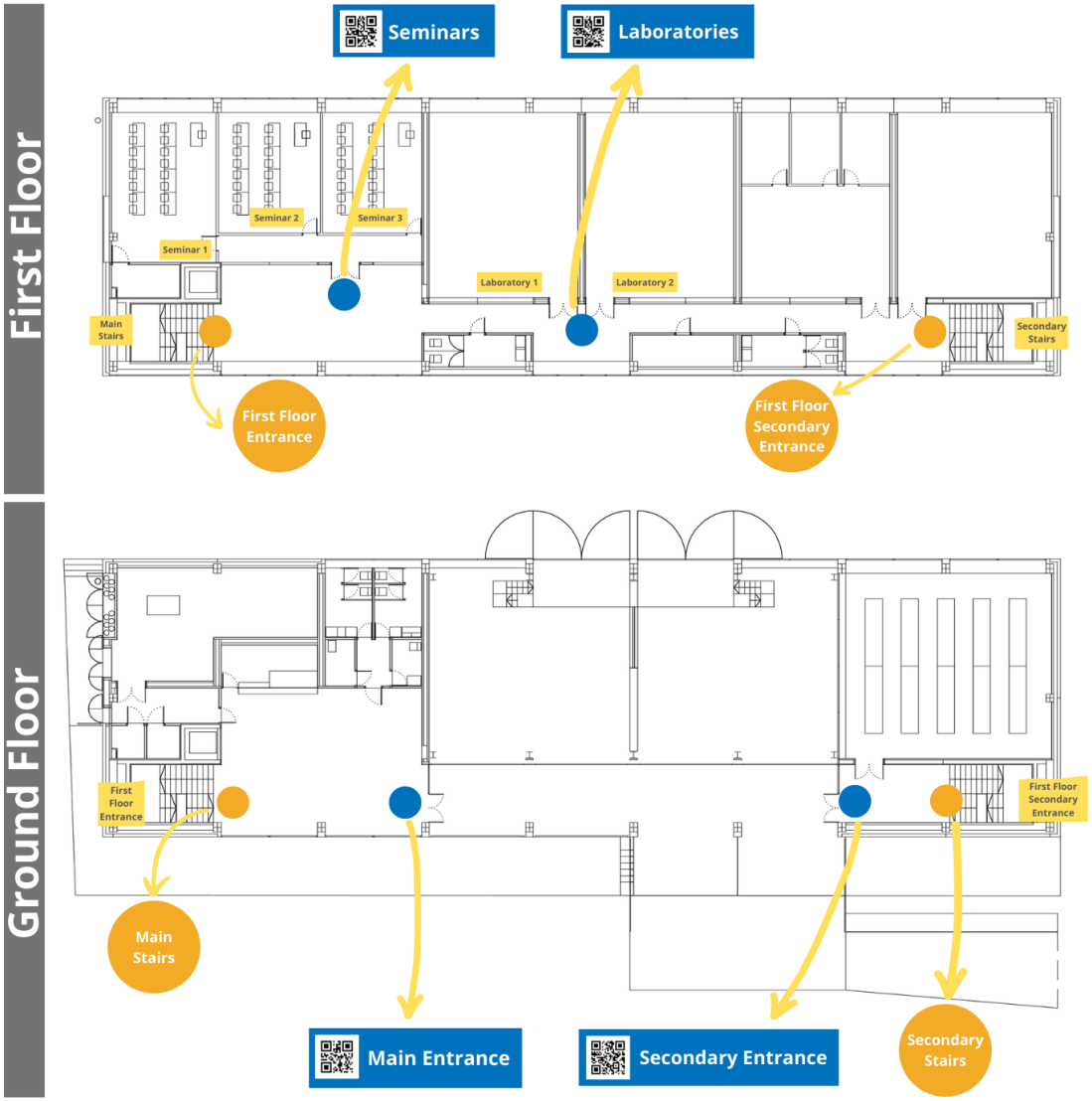


Fig. 5 UDC building map with location and crossing points

Considering now the 4 location points and the 4 crossing points, 8 path sections have been established (16 if we consider the sense of movement), which are displayed in Fig. 6 in green colour:

1. Main Entrance ↔ Secondary Entrance
2. Main Entrance ↔ Main Stairs
3. Secondary Entrance ↔ Secondary Stairs
4. Main Stairs ↔ First Floor Entrance

5. Secondary Stairs ↔ First Floor Secondary entrance
6. First Floor Entrance ↔ Seminars
7. Seminars ↔ Laboratories
8. Laboratories ↔ Secondary First Floor Entrance

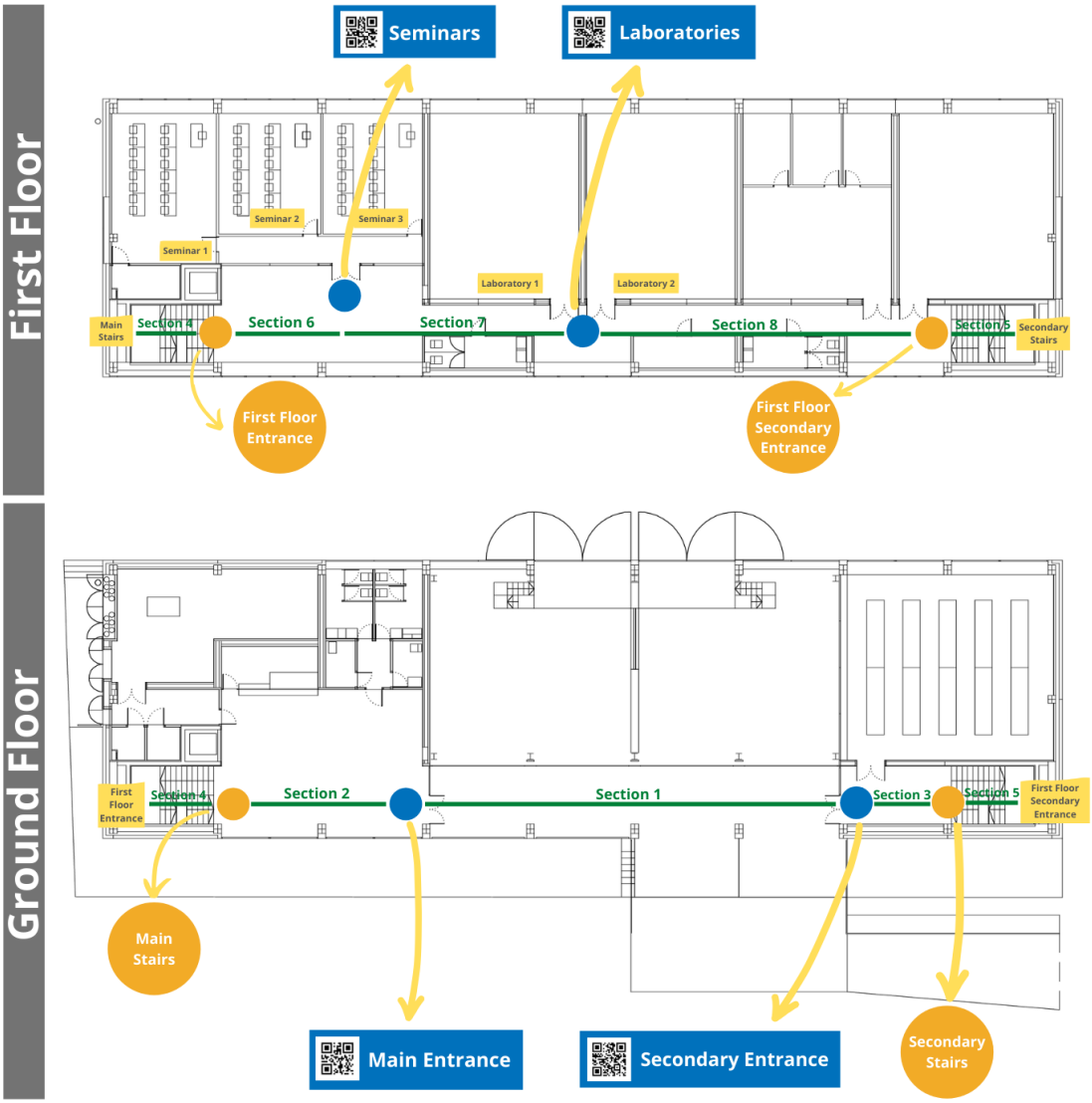


Fig. 6 UDC building map with path sections

The set of location points, crossing points and path sections make up the topological map representation that will be used in this TU. In order to simplify it, a *graph-based* representation is typically used, which reduces the 2D map (floorplan) to a graph based on nodes and links. Graph theory fundamentals should be revised by teachers to introduce it to students. The following references could be used:

- Introduction to graph theory
- Graph theory basics

In this example, the graph that arises is that shown in Fig. 7, where the location points have been represented in blue, the crossing points in orange and the arrows simulate the path sections. This graph includes the relevant information of the map of Fig. 6, but it is simpler because the elements that are not necessary for guidance have been removed, like walls or details of the rooms.

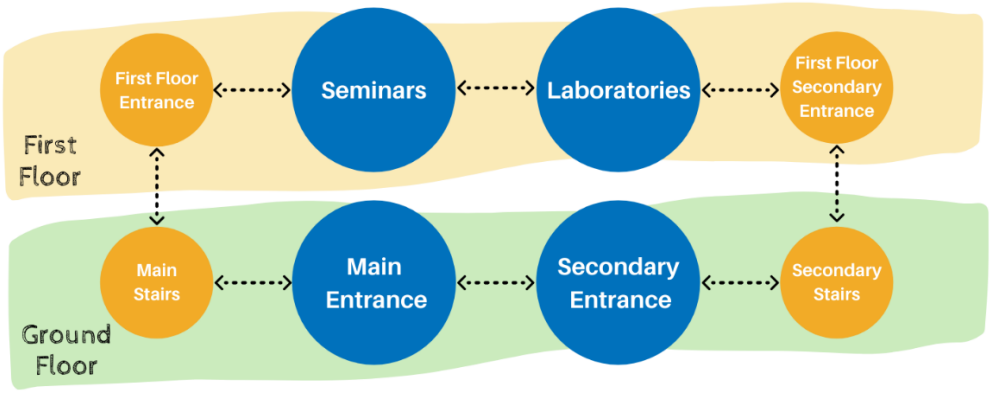


Fig. 7 Graph representation of the UDC building topological map

Bases on this graph representation, it is very easy to guide the user in the app. For instance, let's assume that the user wants to go from the Main Entrance to the Seminars. For this route, as we can see in the Fig. 7, there are two possible paths, being the shorter one: Main Entrance → Main Stairs → First Floor Entrance → Seminars. The app should guide the user from path section to path section by showing images of the location and crossing points (Main Entrance, Main Stairs, First Floor Entrance and Seminars), and indications on the directions to follow (see video "TU3_APP.mp4").

The task the students have to carry out is to **create a graph,** similar to that of Fig. 7. To do it, we recommend to use a floorplan of the school, if it is possible, and to perform the process “by hand” (in a printed paper or using a tablet), so the concept of representation change becomes clearer.

The first thing they have to do to create the graph is **to define the 4 location points** in their school. It is up to the teacher to decide if these points are agreed with all the groups, so the same are used in all cases, or if they are predefined by the teacher and the school. Once the location points are selected, students have to **define the crossing points and the path sections.** In general, there can be several combinations for changing between floors. To reduce them, we recommend avoiding the use of ***the lift in this version,*** so users can only use the stairs, as we have done in this case.

With the path sections defined, each group can create the graph representation.

Finally, once the graph is created, students have **to take photographs of the beginning and end of each section with the smartphone**. It is important that the photographs are taken in horizontal format, so that they fit correctly into the design of the application. These images will be downloaded to the computer later, so at these moment students just have to take them and

make sure that they are stored. For the example of the UDC building, the images we have taken are those shown in Fig. 8.

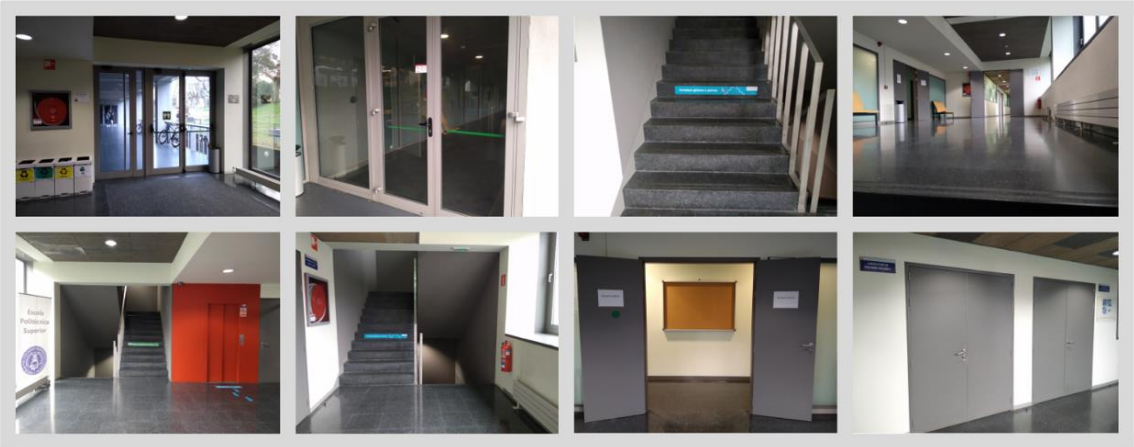


Fig. 8 Images of the locations and crossing points of the centre used for the UDC example

In order to evaluate the completion of the task, it is proposed that the students create a **small report** in which they include a **description of the selection of location points, crossing points and path sections** according to the school configurations. It is important that they justify why these points have been chosen keeping in mind that the app has to be useful, so main school places should be used. In addition, the **report must include the graph** that makes up the final representation used. Finally, they should **include the images taken** for each location and crossing point in the report.

- ***Activity 1 – Task 2***

Goal: implement the functions that represent each of the path sections.

Duration: 1 Hour.

The first step is for each group is to upload the provided template in their App Inventor account. To do it, they must click on “my projects” -> “import project (.aia) from my computer”, as shown in Fig. 9, and select the file “students_template_TU3.aia” provided with this TU.

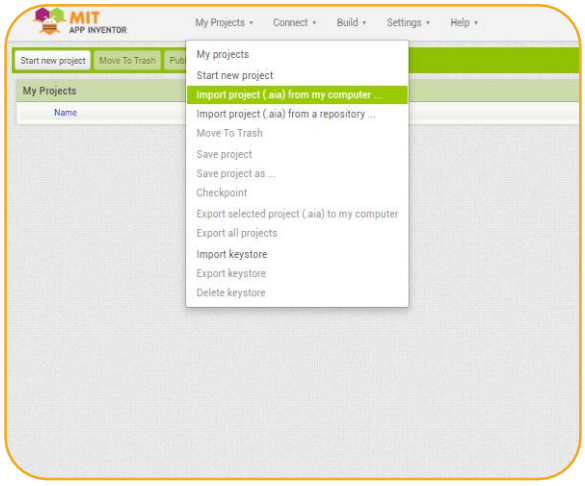


Fig. 9 Screen capture showing the menu to import the project template

If loading is correct, the screen displayed in Fig. 10 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”.

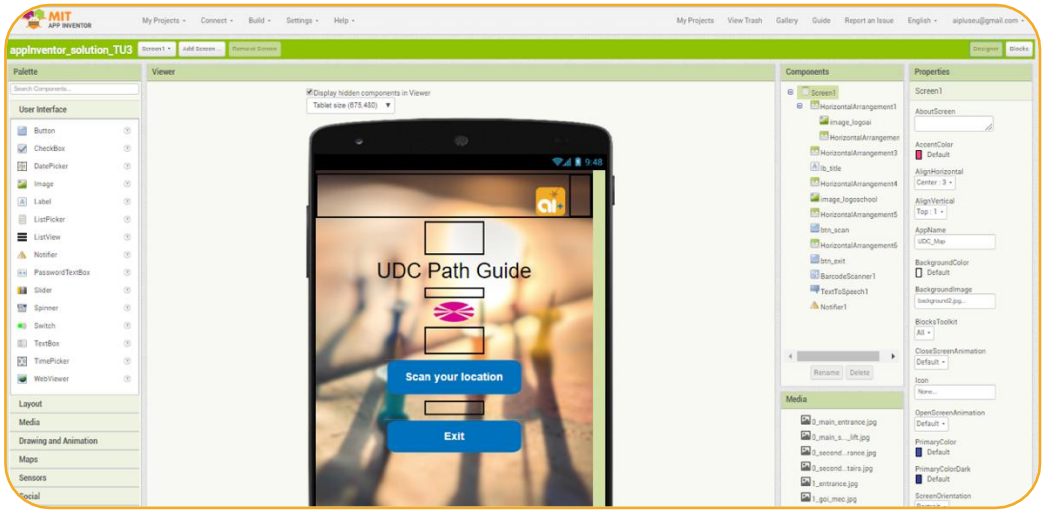


Fig. 10 Appearance of inventor app just after uploading the template

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. 11 should appear.

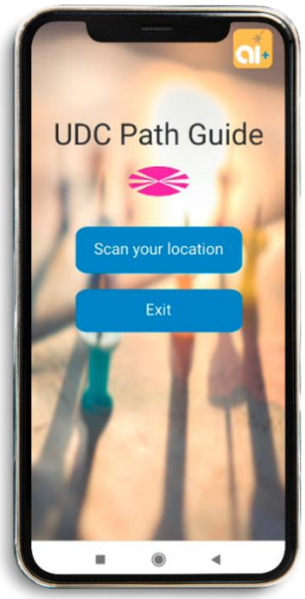


Fig. 11 Appearance of the application on the smartphone

This app, according to the layout created in TU2, has two screens. In the first one, there are two buttons: one to scan the user's position and other one to exit from the app. The code associated to these buttons in screen1 is already completed in the template file "students_template_TU3.aia". On the other hand, on screen2, the images and the guidance steps will be shown, so this is the screen where the student's programming must be performed.

The first step to carry out by students is simply to **update the name of the location points** in the project template, according to the selection performed in the previous task for each school. They are stored in each screen in a list type variable called *location_points*. For screen1, this variable is used to check that the scanned value corresponds to one of the defined location points of the centre. In screen2, it is used to store the set of possible destinations. Remember that the location point names must be the same as the ones in the QR codes. Fig. 12 shows an example of how to add, for instance, the Main Entrance location point. Students can start from this example to finish this step by adding the remaining names in both screens.

Fig. 12 Variable called *location_points* with a Main Entrance point already included

The next step is to **upload the images to the App Inventor project**. To do it, students should download the photographs taken in the previous task to their computer. It is important to *reduce the size of these images* because App Inventor limits each project to 10 Mb. Hence, it is necessary that all the images do not exceed 8 Mb as maximum. Moreover, it is highly recommended to rename the images, because later we will use the specific name of each one in the program, and it is mandatory to easily identify them. One option for name convention could be to start the

name with the floor number and continue with the crossing point name (for example, the Main Entrance photo can be called *0_Main_Entrance*).

Once this is done by all groups, it's time to upload the images to the App Inventor project. To do it, they have to go to the design part and, in the media palette (Fig. 13 Left), click on upload file. Then, a window will open, in which the first image must be loaded (Fig. 13 Right). These steps must be repeated for all images.

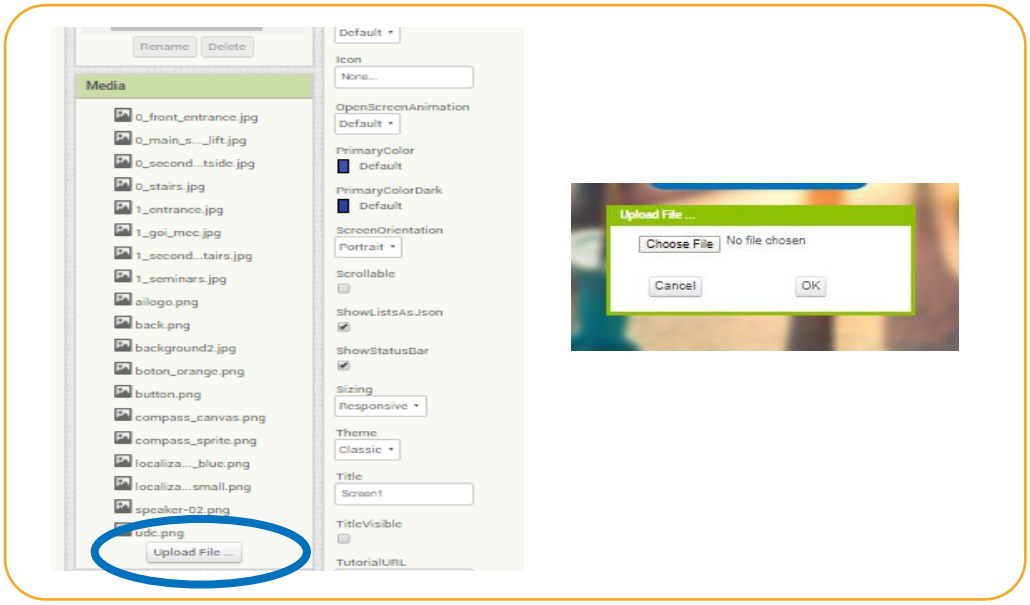


Fig. 13 Left: Media palette Right: Picture of the loading window

Once all the images are uploaded, we have all the elements to program the representation of our AI system, based on the graph previously defined. To understand what the students should do, it is important to understand the app functioning. The programming will be carried out on screen2. Fig. 14 shows the basic aspect of this screen when it is opened, which contains, from top to bottom: a button with an arrow to return to screen1, the AI+ logo, the name of the origin location point (which has been scanned in screen1), and a button that “Select your destination” that will show the list with the possible destinations, list which we can see in Fig. 14.

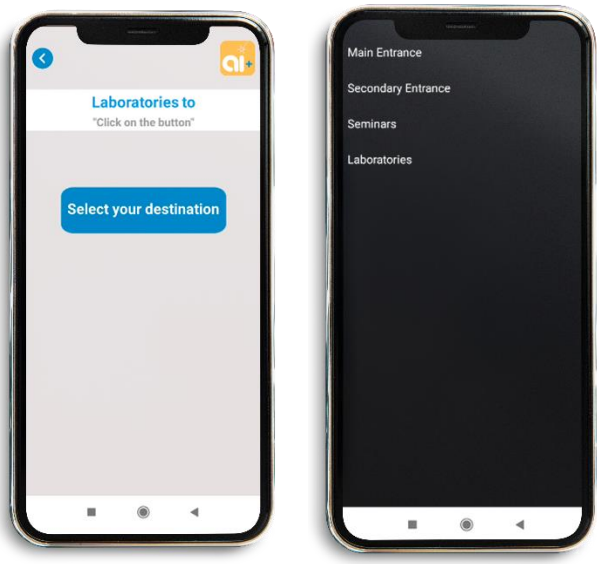


Fig. 14 Left: Aspect of screen2 before selecting destination. Right: Destination list appearance when the "select your destination" button is clicked

Once the destination is selected, the aspect of screen2 included in the template will be that shown in Fig. 15 (these screen captures correspond to the example of the UDC building presented above). As it can be observed, now the top text label includes the whole route. Moreover, there are two images with two names above them, which correspond to the crossing point of the path section. Below them, there is a text with the indications to follow to move from along the path section. On its left, the speaker button is placed, which enables the option to speech the same indications by voice. Below the indications, we can see the compass representation we developed in the previous TU, which shows the user orientation. Finally, at the bottom of screen2, two buttons appear, that allow the user to move between path sections. This layout of screen2 could be designed by students but, in order to focus the TU in the AI concepts, we have decided to provide it in the template, so all groups must respect the visual aspect.

To clarify the operation of the application, the possible screens for the same route are shown in Fig. 15. To clarify the app functioning, Fig. 15 shows the possible screens that may be available for a route. In the first one, left, the initial aspect of the route is shown. In this case, only the button that allows you to go to the next section appears. Clicking on it would load the next section, shown in the Fig. 15 middle. As it can be observed, the left image showing First Floor Secondary Entrance being the same as the right one in the previous path section (they are sequential). Fig. 15 middle shows at the bottom that a new button appears, allowing user to return to the previous path section. Finally, Fig. 15 right displays the aspect of the app when the final path section is reached (final location point). In this case, a new button appears with the text “Arrival”. When clicked, it asks by voice if the user wants to scan a new location and activates the speech recognizer. We recommend here to watch the video "TU3_APP.mp4" again to understand how this interaction with the user works, and how this app creates a representation based on images that allows to perform the user guidance in a simple way.

Obviously, many other options of user interface and app aspect could be possible for this app, but this is clear enough to reach our learning goals.

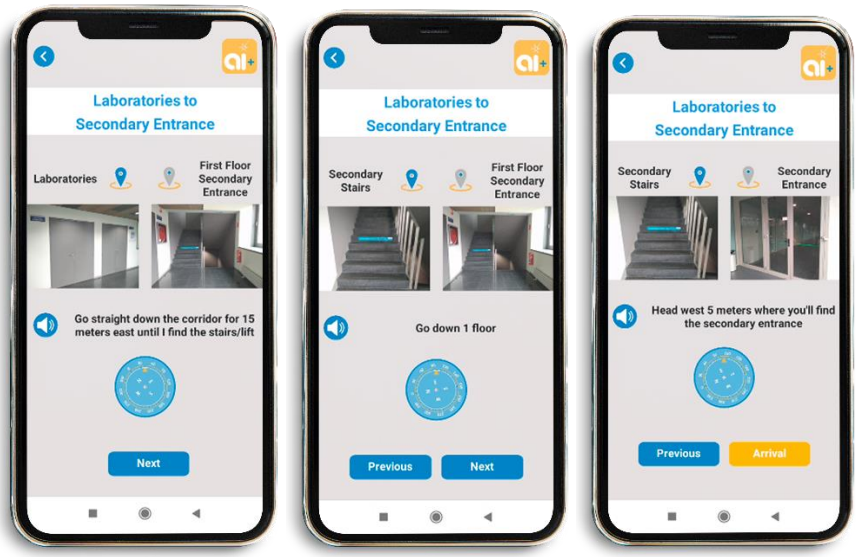


Fig. 15 Left: Appearance of screen2 after select destination. Middle: Appearance of screen2 after clicking next button. Right: Appearance of screen2 on the last section of the route

As it can be observed in the images of Fig. 15, we have translated the graph representation explained in Fig. 7 to the app. Thus, for each path section, it is necessary to show the name and image of the location and crossing points (nodes) and the indications to reach from one to the other (links). Students must understand that this is the representation of the environment we have selected for this AI system. It will become clearer in the next activity that this representation is very useful to perform reasoning over it.

Once the app functioning has been clarified, **students must carry out the programming of the functions in App Inventor that allow to include the graph representation of their school**. To do it, among the pre-programmed blocks in the template, there is a function called "*Print_origin_and_destination_images_names*". It has a total of 5 input parameters (image_origin, name_origin, image_destination, name_destination and indications), corresponding to the information included in the nodes and links of our graph representation. This function displays the parameters in the right places of screen2, so students do not have to do it. Hence, to represent each section in the app, students must create one function for each path section and then put within it the function "*Print_origin_and_destination_images_names*". To achieve it, the first block of the process palette called *to procedure* must be selected (Fig. 16). This is a function that executes everything within it when it is called.

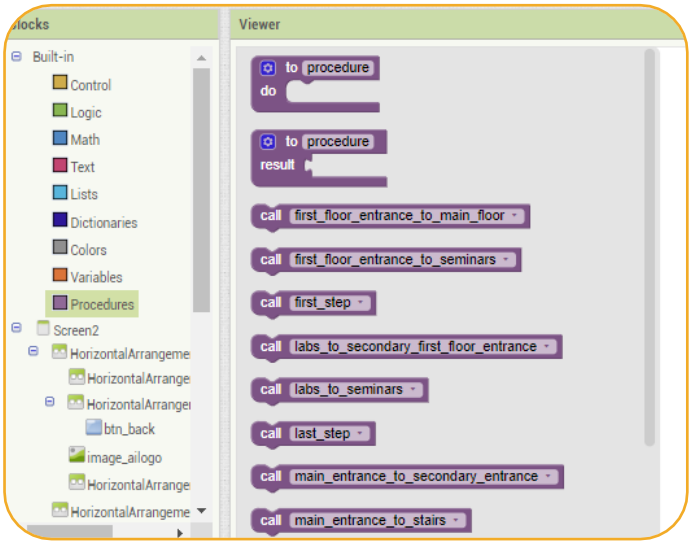


Fig. 16 Procedures programming options

For example, for the UDC building, if we want to create the section that goes from the Main Entrance to the Secondary Entrance, the first thing to do is to create a function with the appropriate name, as shown in Fig. 17.

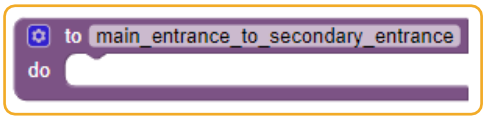


Fig. 17 Example of a section function

Next, this function has to call the function "*Print_origin_and_destination_images_names*", which must include the 5 required parameters (see Fig. 18) In this example, the name of the initial and final crossing point images are *0_main_entrance.jpg* and *0_secondary_entrance.jpg*, while their text names are *Main Entrance* and *Secondary Entrance.* Finally, the indications could be: *go out the main door (to the east) and within 30 meters you will find the secondary entrance.* This last parameter is very important because it implements the **voice interaction between the human and the app.** Such indications must be meditated in order to be concise and natural. To achieve this, one of the things they must contain, whenever necessary, is a reference to the orientation to be followed to reach the destination, since the user always knows his/her orientation due to the app's compass. An example of this can be seen in the Fig. 18, where the indication specifies the user to head *east*.

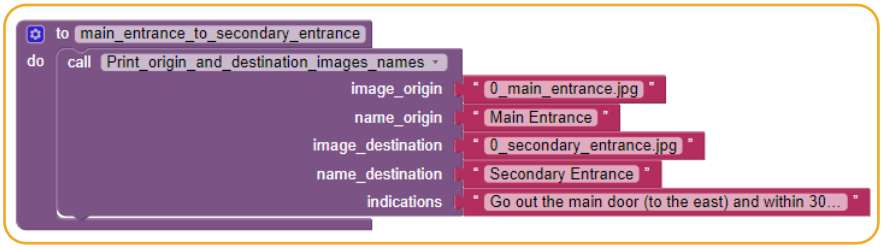


Fig. 18 Complete function of the first section

These two steps (represented in Fig. 17 and Fig. 18) must be repeated for each path section. It is important to remember that students have to implement two functions for each path section of their school, one for each direction.

To evaluate if the students have finished this task correctly, they must call the function they want to check at the end of the screen initialization event (*when screen2.Initialize*), as shown in Fig. 19, where the function *"secondary_entrance_to_main_entrance"* is tested. This will represent the images, names and indications of the path section after selecting a destination (because that's when the components containing them are made visible). This is just a way to check that students are loading the right things, but **they should delete the call after this check**.

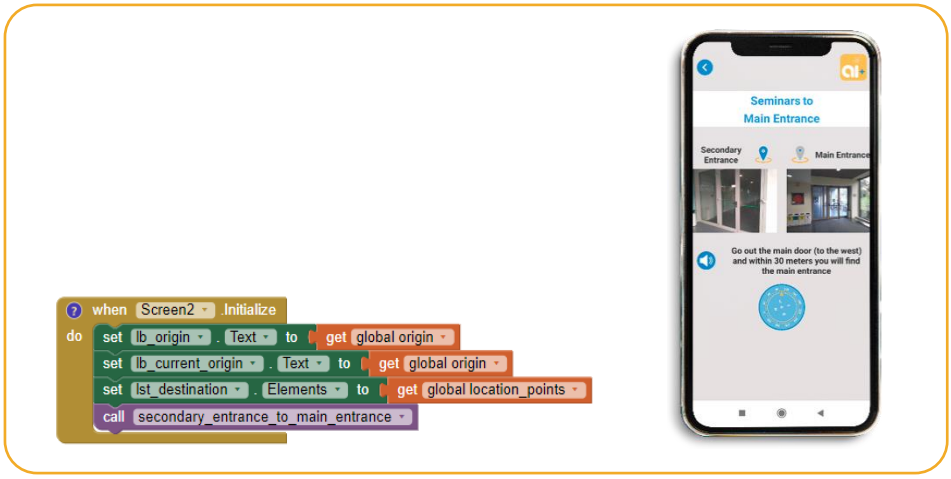


Fig. 19 Left: Programming to test if a path section function has been properly created. Right: Appearance of the application when a path section is tested

9.2.2 Activity 2

Goal: programming the optimal routes between location points.

Duration: 2 Hours.

This activity will be focused on introducing reasoning to students. As commented in the introduction, this is a novel topic for them, which will be treated here with a specific example just to provide them a basic notion of reasoning in the scope of AI. The idea is to develop a program that contains a set of rules based on the graph developed in activity 1, which guides the user for the optimal routes in the school.

At this point, we recommend teachers to introduce students the fundamentals of reasoning. When searching for more information, keep in mind that reasoning, computational thinking, and problem solving are equivalent concepts that differ from the field where they come from, but all of them study the process that must be followed to select the proper action to perform in order to reach a goal or solve a problem. At this level, teachers can use, for instance, the following references:

Computational thinking: https://youtu.be/mUXo-S7gzds

Symbolic representation: https://fyoutu.be/WHCo4m2VOws

Reasoning and acting: https://bit.ly/2Y83BRK

To perform reasoning in this case implies using a symbolic representation based on the topological map defined in the previous activity, and in the graph created by each group, which includes the path sections represented in a very simple way (nodes and links). Each group will program a set of basic rules to guide users through the optimal routes, using the pre-defined path sections. The simplest type of rule is an *if-else structure*, and this is the one we will use in this case.

The *route* function, that is already created in the template app, has several if-else blocks, as shown in Fig. 20, and students must complete it. This function was designed considering 4 location points. In case some group wants to add more, new conditionals should be added.

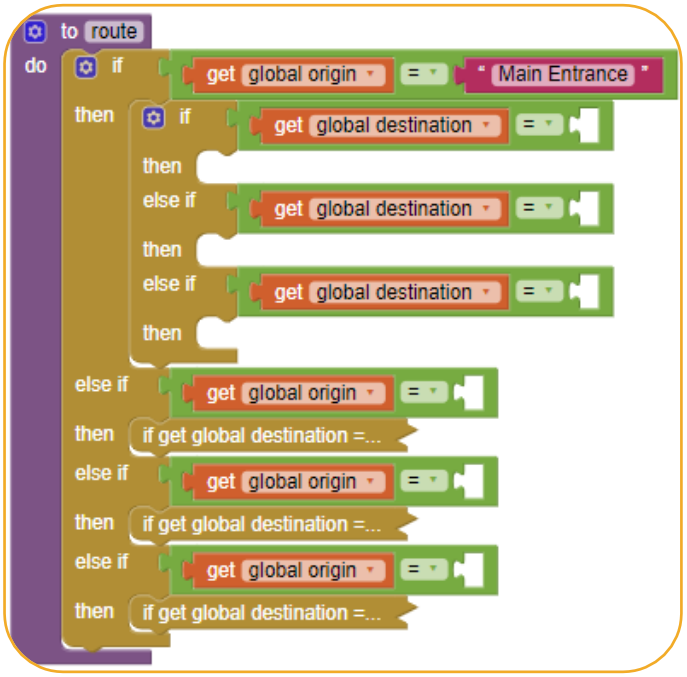


Fig. 20 Route function

To solve this activity 2, students will have to use the App Inventor functions implemented at activity1-task2, the route function shown in Fig. 20, and two more that are already included in the template: *first_section* and *last_section*. These last two functions must be used in the first and last section of a route to show the correct buttons at screen2, as we will explain later.

To program reasoning in this app, it is necessary to know how to determine the optimal route for each path. Following the example of the UDC building to guide explanation, it can be seen in the graph of Fig. 7 that, in this case, there are two possible paths for each route. Hence, it is necessary to choose one of them to guide the user. This selection depends on our preferences, because the optimal route could be the shortest, or the fastest, or the one that avoids stairs, for example. In this case, for the sake of simplicity, we will use the distance as the main criterion, so the *best route will be the shortest one*.

In order to know which is the shortest route between an origin and a destination, it is necessary to compare the total distance of each one of them. To calculate it, students should measure the real distance of each path section in the school or provide an approximate value. This distance will be included in the links of the graph, so the calculations will be easy, accurate and comparable between them.

Fig. 21 displays this improved graph in the case of the UDC building. The values shown in the graph correspond to the real distance between location points and crossing points in reality, but they are scaled to a range between 0 and 10, with 10 being the longest distance and 0 the shortest, just to simplify the measurements. It is important to point out to students that **this graph constitutes the final representation that will be used in the TU**, and from it, different types of reasoning can be performed.

At this point, students should modify their own graph representation to include the distances and include it in the report of activity 1.

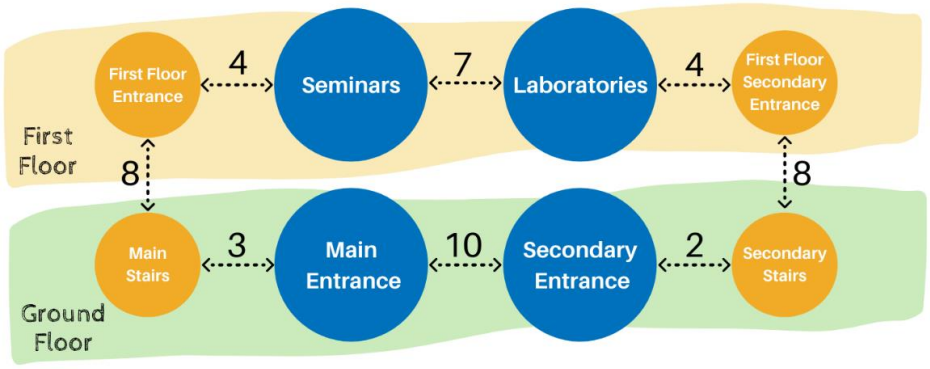


Fig. 21 Final graph with links including the distance between path sections

To clarify how this representation can be used to perform reasoning, in the case of the UDC building, let us imagine that the user wants to go from the Main Entrance to the Secondary Entrance. As shown in Fig. 21, two different routes are possible, one implies moving to the right in the graph, from the Main Entrance to Secondary Entrance, and the other one implies moving left, passing through all the points until reaching again the Secondary Entrance. In this case it is obvious which is the optimal route, but more complex cases could arise, so we have to explain how to calculate it. The only thing we have to do is to sum the distances of each path section and select the smallest, as shown in the following table:

Main Entrance → Secondary Entrance		
	Path Sections	Distance
Route 1	Main Entrance → Secondary Entrance	10
	Total:	10
Route 2	Main Entrance → Main Stairs	3
	Main Stairs → First Floor Entrance	8
	First Floor Entrance → Seminars	4

	Seminars → Laboratories	7
	Laboratories → First Floor Secondary Entrance	4
	First Floor Secondary Entrance → Secondary Stairs	8
	Secondary Stairs → Secondary Entrance	2
	Total:	36

So, as expected, one of them is made up of a single section and has a total distance of 10, while the second one includes 7 path sections, with a total distance of 36. In this case, the optimal route is the first one.

If the user wants to go from the Main entrance to the Laboratories, it is not so evident the optimal route. But using again the graph, it is very simple to calculate it:

Main Entrance → Laboratories		
	Path Sections	Distance
Route 1	Main Entrance → Secondary Entrance	10
	Secondary Entrance → Secondary Stairs	2
	Secondary Stairs → First Floor Secondary Entrance	8
	First Floor Secondary Entrance → Laboratories	4
	Total:	24
Route 2	Main Entrance → Main Stairs	3
	Main Stairs → First Floor Entrance	8
	First Floor Entrance → Seminars	4
	Seminars → Laboratories	7
	Total:	22

Hence, the optimal route in this case would be the second one.

Once we know how to calculate the optimal routes from the graph, it is time to start with the programming. In what follows, an example using the routes that have just been analysed, will be solved (Main Entrance → Secondary Entrance and Main Entrance → Laboratories). Remember that the objective is to complete the *route* function included in the template and displayed in Fig. 20.

The first conditional in this function aims to check in which of the possible location points the user is (value stored in the variable "global origin"). The first task of the students will be to fill in the 4 logical conditions with the names of their location points, as shown in Fig. 22 for our case.

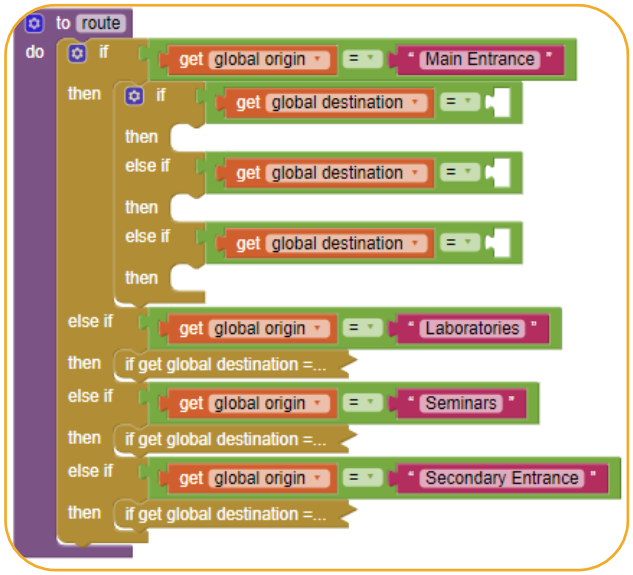


Fig. 22 Route function with the possible origins filled

Within each of the origins, there is a new conditional, through which the destination selected by the user is checked (value stored in the variable "global destination"). As with the origins, it is necessary to complete all the possible destinations of all the origins (Fig. 23).

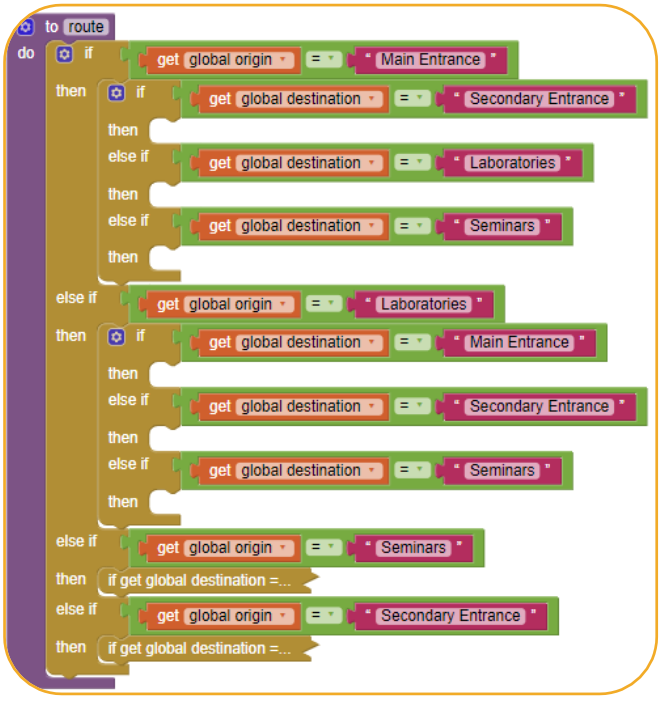


Fig. 23 Route function with the possible destinations filled

Once completed, it is time to program each of the routes, but a previous consideration must be explained to students. As shown in Fig. 15, the app allows the user to move between path sections by means of buttons. These buttons are slightly different depending on whether it is the first section of the route (in this case, only one button is displayed, as in Fig. 15 left), an intermediate section (two buttons appear as in Fig. 15 middle), or the last one (the Arrival button is shown on the right, as in Fig. 15 right). In order to obtain this response on the app, when routes have more than one section, it is necessary to call the *first_section* function in the first section of the route and the *last_section* function in the last one. For routes with only one section, only the function *last_section* needs to be called.

The first route to program in the example displayed in Fig. 23 has as origin the Main Entrance and destination the Secondary Entrance, so the function that must be added in the first conditional is *main_entrance_to_secondary_entrance*, as shown in Fig. 24. This route is composed of a single section, so it is necessary to include the call to the *last_section* function, as shown in Fig. 24.

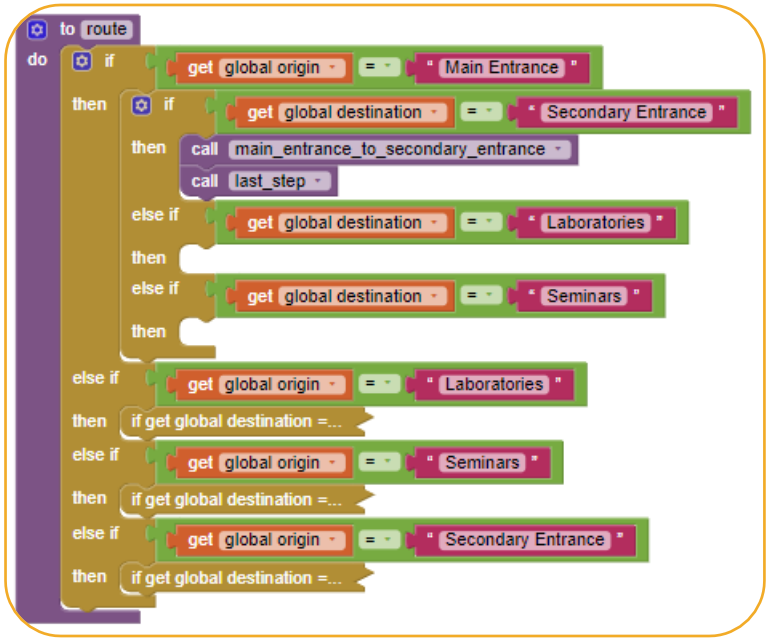


Fig. 24 Route function with the main entrance to secondary entrance route programmed

At this point, **students should program their first conditional in the route function**, in a similar way as we have explained in Fig. 24. To do it, they have to use the graph with the distances to calculate the optimal route and program it. To test this case, it is necessary to launch the app, scan a QR code containing the origin location point and select the corresponding destination point. Then, the app would show the route to follow to reach the destination. In the case of the program of Fig. 24, the app aspect would be that of Fig. 25. It is possible to see that, as expected, only the Arrival button is displayed, according to the use of the *last_step* function.

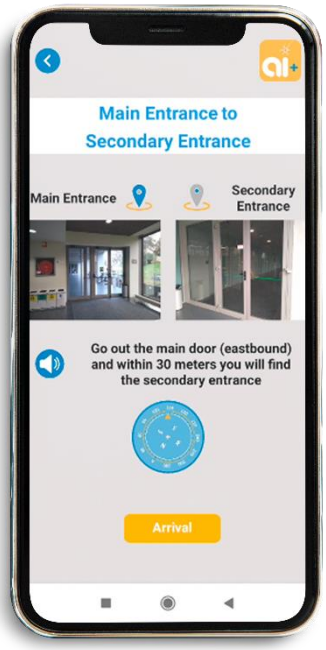


Fig. 25 Appearance of the application with a one section route selected

To continue with the development of the example program shown in Fig. 24, the route from the Main Entrance to the Laboratories must be programmed. This is a more complex route and it serves as an example of a route with several path sections. As explained before, the optimal route in this case passes through 4 different path sections:

1. Main Entrance → Main Stairs
2. Main Stairs → First Floor Entrance
3. First Floor Entrance → Seminars
4. Seminars → Laboratories

Remember that it implies 4 different screens in the app, with the option of moving from one to the other using *Previous* or *Next* buttons (see Fig. 15). To program this response properly, we have included a counter variable called *global_step* in the template that stores the path section number in which the user is for a given route. Fig. 26 displays how the *global_step* variable can be used in a new if-else structure. This variable will increase each time the user clicks on the next button and will decrease if she/he clicks on the previous one (this is already programmed in the template).

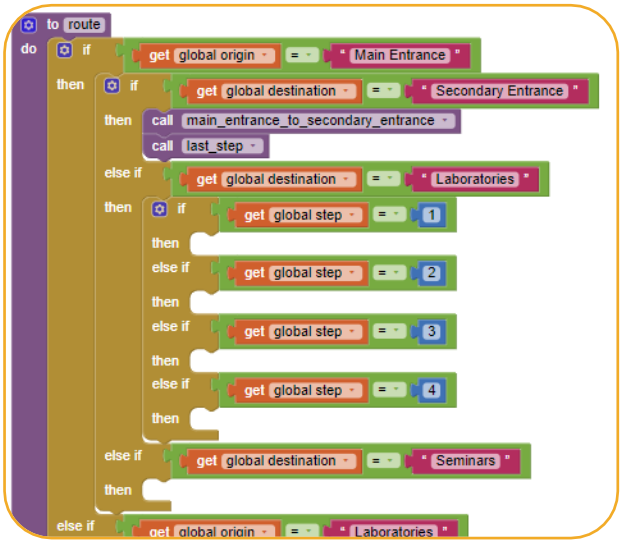


Fig. 26 Route function with the third conditional in the main entrance to seminars route.

Once this is understood, it is time to the call to the functions of the path sections include in each of the branches of this second conditional. Fig. 27 shows the solution for this route, including the different functions that correspond to each path section like "*main_entrance_to_stairs*" or "*seminars_to_labs*". Moreover, the functions *first_step* and *last_step* have been included too, in the first and last steps (1 and 4). It is important that teachers understand this solution in order to explain students how to perform their own ones.

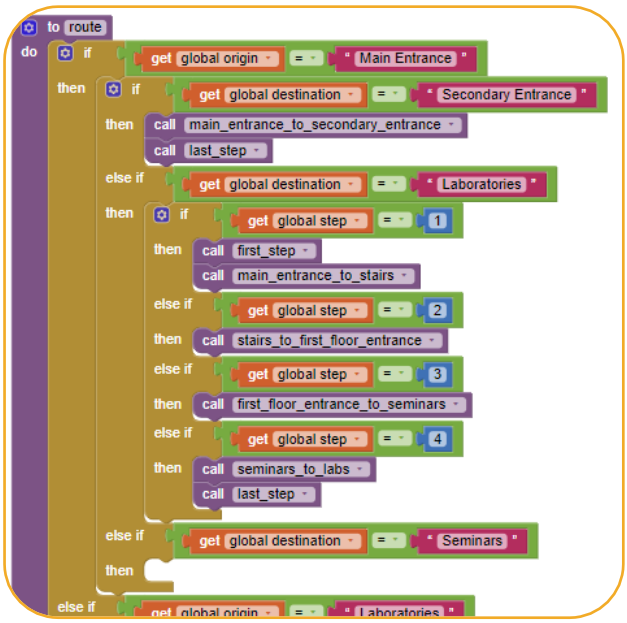


Fig. 27 Route function with the main entrance to seminars route programmed

Fig. 28 shows the final appearance of the 4 path sections that form the route in the application if we execute the code of Fig. 27.

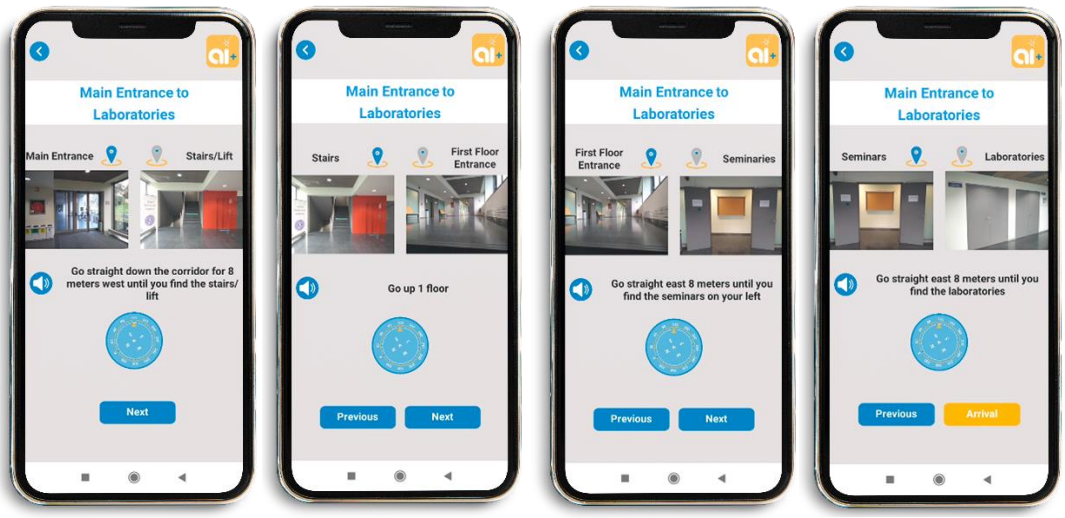


Fig. 28 Captures of each of the path sections of the route between the main entrance and the laboratories

In summary, the steps students have to follow to program the if-else rules that implement reasoning in their app are:

1. Calculate the optimal route using the graph with the distances.
2. Check whether the selected one is a single-section route or a multi-section route.
3. If it is a single-section route, simply call the function corresponding to the path section created in activity1-task2, and the *last_section* function after it.
4. If it is a route with several path sections, create a conditional structure depending on the global_step variable and add each of the path section functions, including *first_section* in the first step and in *last_section* at the end.
5. Repeat the three previous steps for each of the routes.

Once all the routes are implemented, the students should test that their app guides the user correctly by checking all the possible combinations and solving possible “bugs”. They must be aware of providing the right indications, photos and overall working. Once the app has been tested, the *School Path Guide* app is finished and ready to be used.

9.3 Final remarks

Once the TU2 and TU3 have been carried out and the app has, therefore, been completed, it is time to point out the main AI concepts the students have practiced.

This app includes 4 of the 8 topics of AI established in TU1: perception, actuation, representation and reasoning. The first two were the focus of TU2, and the last two in the present unit.

Concerning perception, the app uses the following sensors:

- The camera, to scan the QR code of the location points.
- The gyroscope and magnetometer, to get the spatial orientation of the phone.
- The microphone, to pick up the user's speech.

- The touch screen, where the user can click on buttons.

Regarding actuation, the application includes:

- The LCD screen, on which the user is shown the information.
- The speaker, which communicates indications to the user.

The representation topic was included by means of the topological map we have defined using the graph, the photographs of the location points, and the indications.

Finally, the reasoning topic was explained by means of the implementation of the if-else rules that make up the optimal routes represented in the graph.

To conclude, students should understand that the apps they are developing still not cover all the topics mentioned in the TU1, so they do not are not perceived as "intelligent", but they contain the core elements to reach to that point.

10. Evaluation

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

Assessable activity	Score
1- Program test	40%
2- Activity 1 report	20%
3.1- Group report 3.2- Checklist	30% 10%

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. That is, each group will use the app of other group as it they are visitors in the school and they will provide a brief report to the teacher with their opinion regarding:

    1. *Did the app guide you properly to your destination?* If the answer is NO, please, explain the detected problems
    2. *Are the images useful? (i.e. the images represent the path to follow)*
    3. *Are the routes optimal? (i.e. they follow the shortest path)*
    4. *Are the written instructions clear and true?*

    In addition, the teachers should check that the basic working of the app is correct:

    - The screens display the expected information and the indications, names and images are correct all the time.

- The general app functioning is correct, with no stops or pauses.
- The human-machine interaction aspects are correct: clear and direct sentences.
- The next, previous and arrival buttons appear and disappear when they should (if that doesn't happen it's because they haven't correctly called the first_section and last_section functions).
- 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*: in this part, we include the report created at the end of activity 1, which includes the representation tasks developed by students.

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

11. 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. Lift

Today, all educational centres are accessible and have lifts for those who need them. It would be very interesting to improve the app, so it considers optimal routes using the lift. It is interesting for visitors with limited mobility, or in buildings with several floors. One solution would be to include a new button in the current version that considers the lift when activated. This would imply a change in the reasoning part because the best option is not the fastest or shortest route, but the one that goes by the lift and not by stairs.

2. Add new floors

Most centres have more than one floor. If so, it would be interesting to add it to the application. This complicates a bit both the reasoning and the representation part because the number of path sections is much higher and you have to take into account that you can go up or down more than one floor, so the function is to go up and down stairs is a bit complicated.

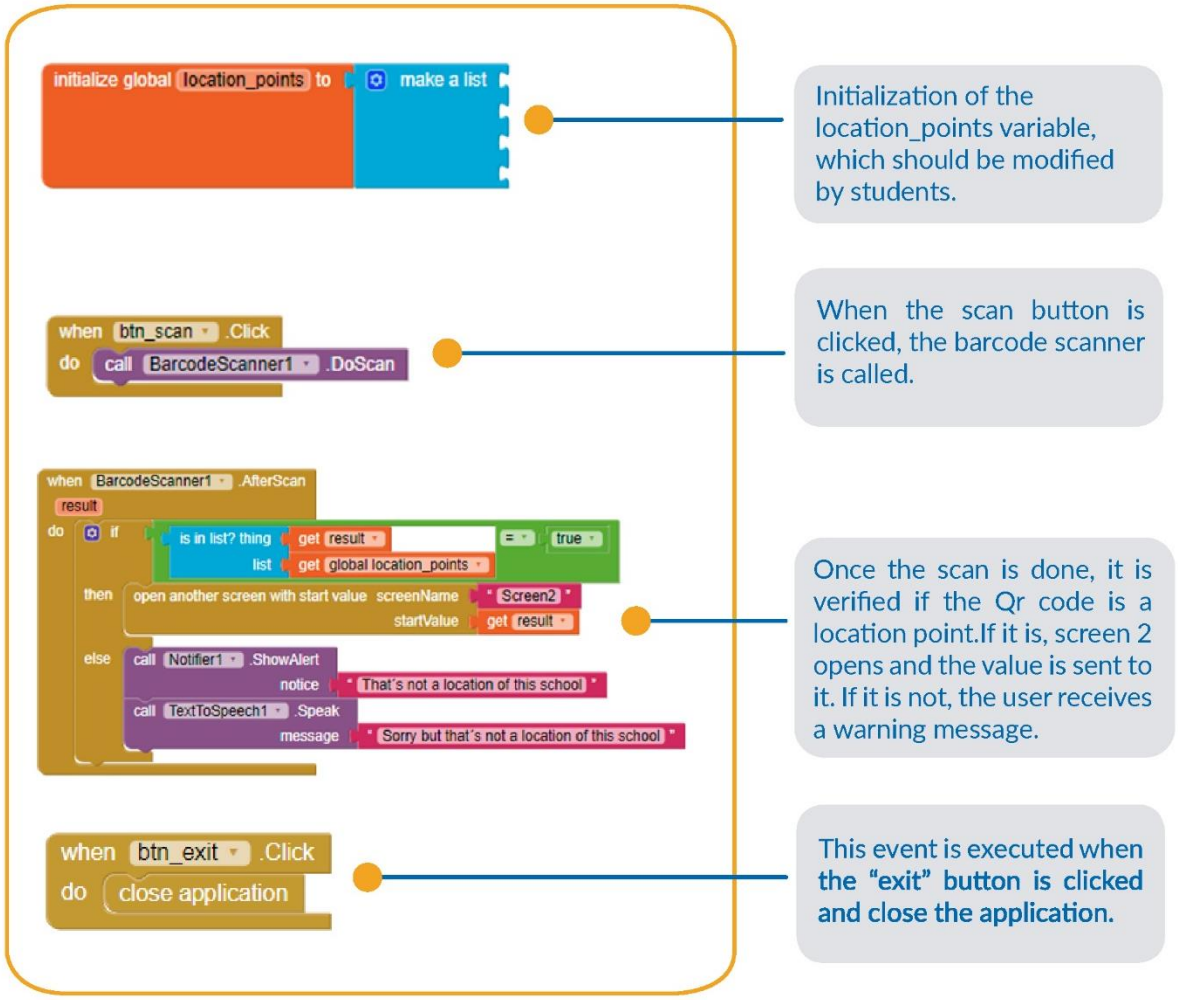
12. Annexes

Annex I: Pre-programmed blocks

To adjust the TU duration, a template is provided to focus the attention in AI aspects. This template includes the graphic part of the app too. This appendix aims to describe all the pre-programmed blocks of the template, organized in screen1 and screen2.

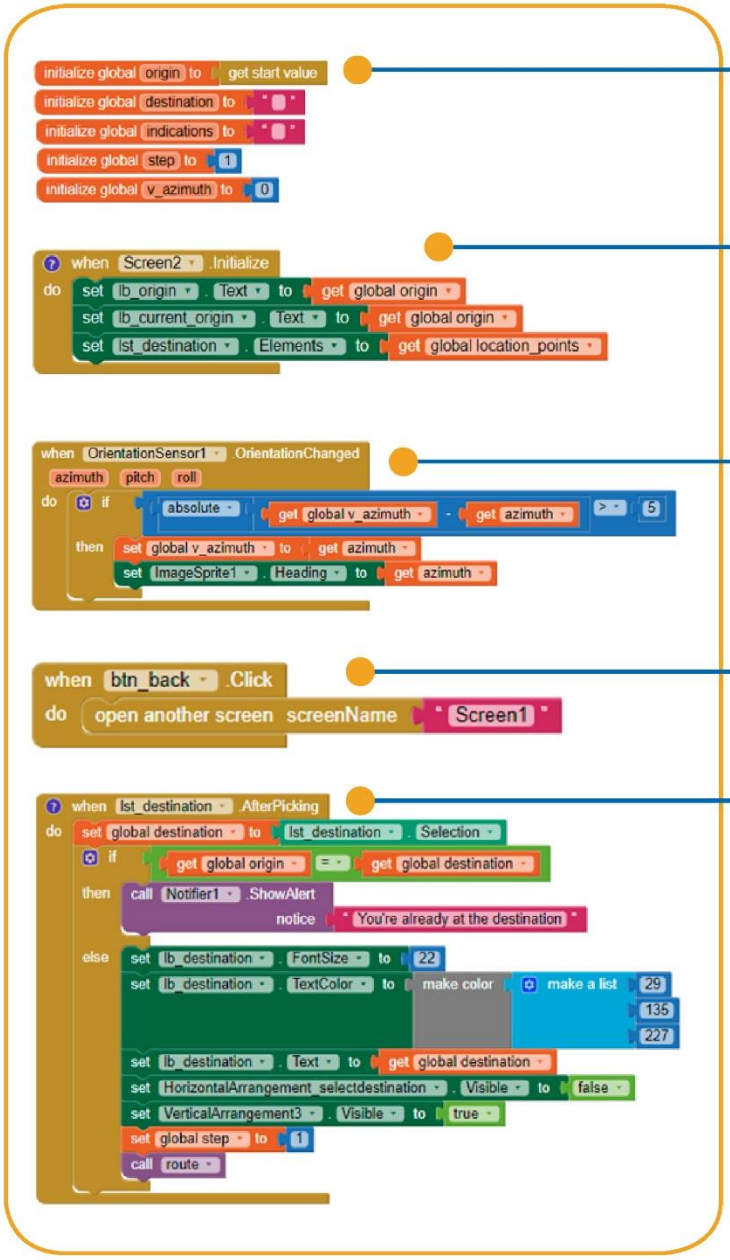
- **Screen1**

In this first window, all the blocks are completely programmed except for the first one, which the students must complete with the names of the location points of their school.



- **Screen2**

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



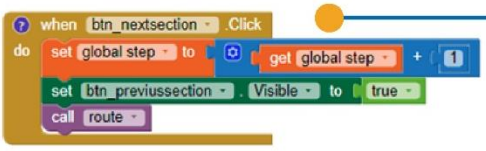
Initialization of variables.

When Screen2 is initialized, the name of the origin location is displayed on the corresponding text labels and the possible destinations are loaded into the list.

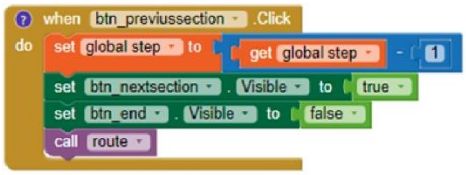
Compass programming.

Back button programming.

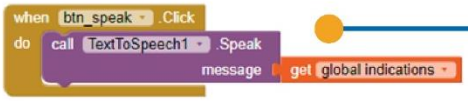
When the user selects the destination, this block checks if it is different from the current location. If it is the same, it warns the user. If it is not, it is saved in a variable called destination and represented in the corresponding text labels (adjusting also its color and size). In addition, the components in which the sections of the route are displayed (images and text labels for names and indications) are made visible, and the button for selecting the destination is made invisible. Finally, the step variable is set to 1 (because all journeys must begin with section 1) and the route function is called up to start the guide.



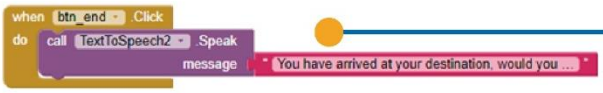
When the user clicks on the next section button:
- The event adds 1 to the step variable.
- A button is activated to go to the previous step.
- The route function is called to show the following section.



When the user clicks on the previous step button:
- The event subtracts 1 from the step variable.
- A button to go to the next step turns on, the end of route button turns off, and the route function is called.
- The route function is called to show the previous section.



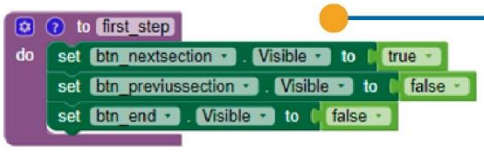
When the speak button is clicked, the directions are provided by voice through the speaker.



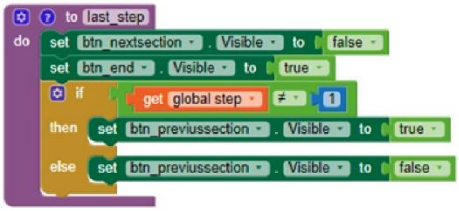
When this button is clicked, the user is asked if he/she wants to scan a new location.

When the user clicks on the arrival button, he is asked if he wants to scan his new location. This event handler runs when the sentence has been completed and calls the voice recogniser.

Check the captured word and execute the appropriate action. If the result is YES, screen 1 opens.

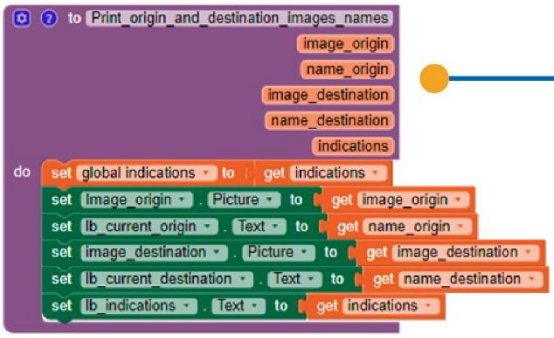


At the first step of each route it is necessary:
- To make visible the button to go to the next step
- Hide the button to go to the previous step
- Turn off the end button



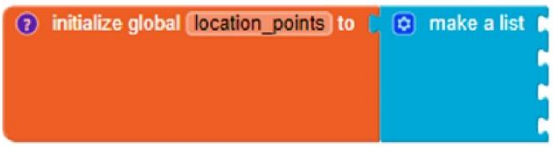
At the last step of each route it is necessary:
- Make visible the button through which the user passes to the previous step (if there is more than one step).
- Hide the button that takes the user to the next step.
- Activate the end of route button.

In addition, if the total number of steps is one, the previous section button must be turned off.

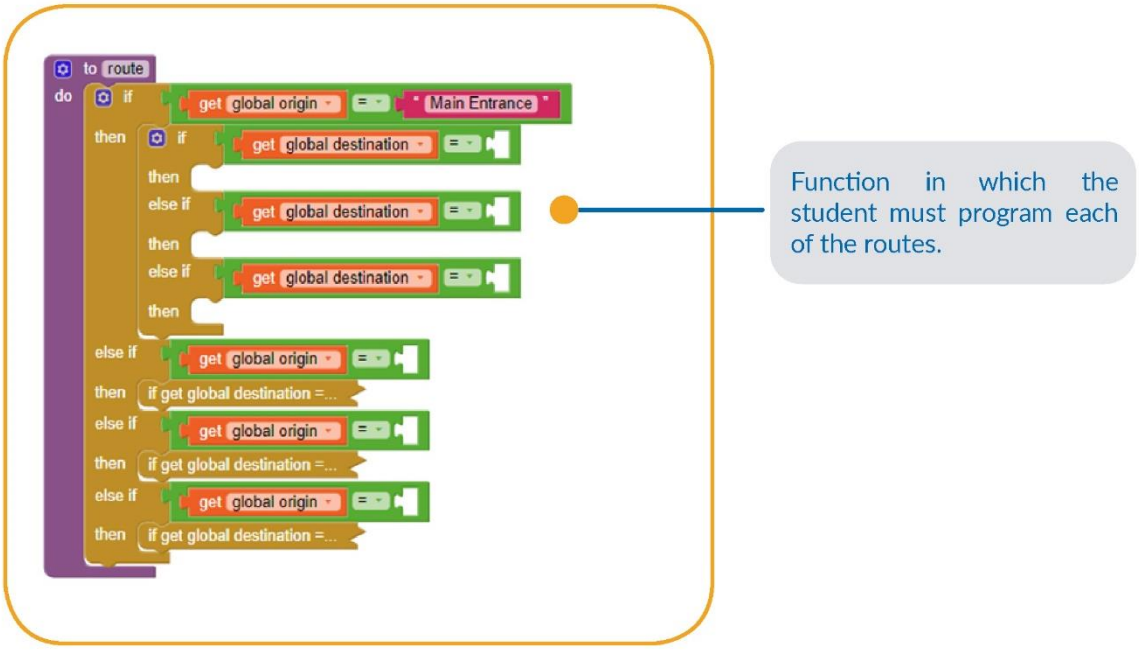


This function represents the instructions to be followed in each section, as well as the source and destination names and images. This function serves to simplify the code.

The blocks that should be modified will be:



Initialization of the location_points variable, which should be modified by students.



Function in which the student must program each of the routes.

Annex II: Teacher rubrics

Level (score) / Aspects to be evaluated	Expert (4)	Competent (3)	Partially competent (2)	Not yet competent (1)
Adequate selection of information (Graph definition, 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 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 3 – ACTUATION AND PERCEPTION</th></tr>
<tr><td>Students</td><td colspan="4"></td></tr>
<tr><td colspan="5"></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>Graph/ map</td><td colspan="2"></td><td>Optimal routing</td><td></td></tr>
</table>