Co-funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.

Introduction to MIT App Inventor – creating an application for remotely control the DIY robotic car

Introducing the 5 Big Ideas in Artificial Intelligence using Internet of Things in STEM education

T2.4 IoT Projects Design & Resources Development

06.10.2023 | EDUMOTIVA
PROJECT NUMBER: 2022-1-FR01-KA220-SCH-000085611

AI4STEM IoT Projects Design & Resources Development
Project: The DIY robotic car
Introduction to MIT App Inventor – creating an application for remotely control the DIY robotic car

Copyright

Co-funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.

Table of Contents

1.Creating an application for remote controlling the DIY robotic car 3
1.1 Introduction 3
1.2 Planning the design of the application 3
1.3 Designing the application 3
Creating the layout for placing/arranging the buttons 4
Adding the ListPicker and the buttons 6
Adding a text notification 9
Adding Extensions 10
1.4 Programming the Application 15
Coding the ListPicker (i.e., Scan_List) component 15
Coding the Label (i.e. Label1) 18
Coding the Disconnect (i.e., Dis) button 19
Coding the buttons for navigating the robotic car 19
1.5 Building the application 22
1.6 Pairing the application with the robotic car 23

Co-funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.

1.Creating an application for remote controlling the DIY robotic car

1.1 Introduction

This document will present a warm up activity for introducing your students to the MIT App Inventor environment. Through this activity, the students will learn how to create an application that will enable controlling remotely the robotic car, through a smart device. Therefore, and towards this scope, they will learn how to design the interface of the application, and how to program the items included therein. Prior to this activity you are highly advised to instruct your students to create the script described in the "T2.4_Programming_the_robotic_car.pdf" file, or download it to the robotic car, by using the corresponding .hex file.

1.2 Planning the design of the application

Before moving on with the design of the application, it is crucial to be aware of the components that will need to be included. One of the main goals is to find a way to establish connection and communication between our smart device and our robotic car. To do that we will set some messages that will be sent/transmitted by our smart device and will be received by our robotic car via Bluetooth. Each time a message is sent/transmitted by the smart device and is received by the micro:bit board, the robotic car will correspond in a different way.

Having this in mind, we need to design an application that will allow us to:

- control our robotic car's movement. To do that we need 5 buttons: 4 of them will move our robotic car forward, backward, on the right and on the left, and 1 that will stop any performing movement
- permit our application to be connected via the Bluetooth of our device
- permit our application to scan/search for available Bluetooth devices and connect to a choosing one
- disconnect our application from the connected Bluetooth device
- optionally, inform us about the current status of the connectivity

All the aforementioned, help us to create a concrete idea regarding the outcome of the design process

1.3 Designing the application

Design is a rather free process, and mostly based on the creator's aesthetics. The following instructions are indicative and present a rather simplified version of the appearance of the interface that our application can have.

Figure 1 presents a preview of the interface that we will create based on the needs that we recorded in the previous section.

Co-funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.

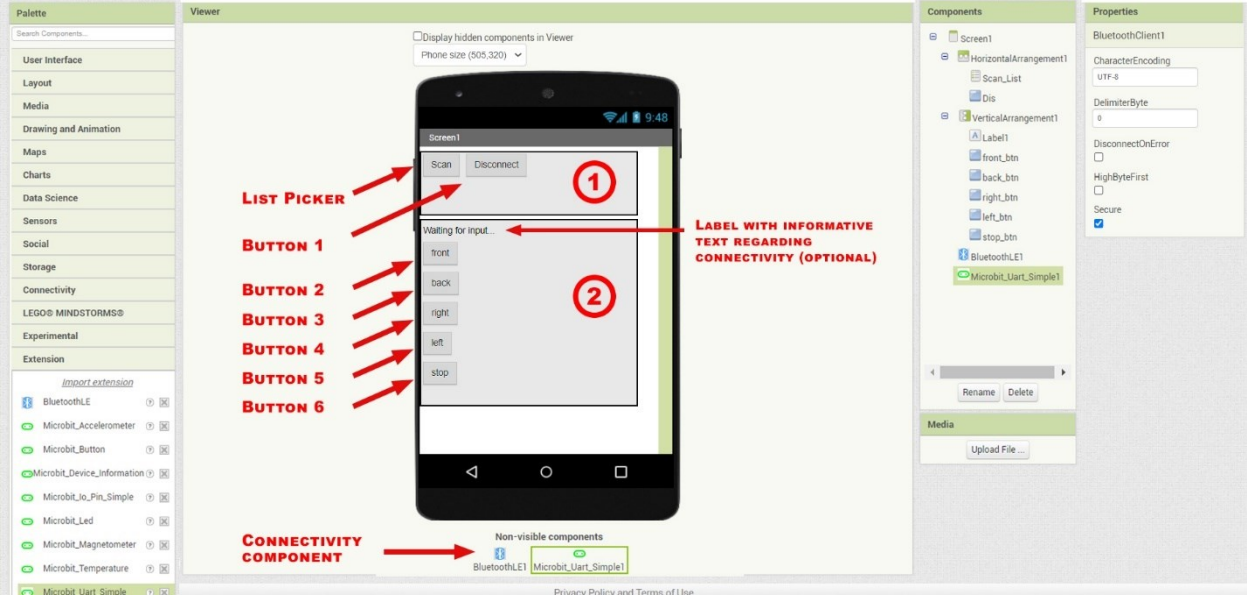


Figure 1: A preview of the interface you are about to design

To better organize all the components on our screen we want to create two layouts. One that will host the Scan/search button and the Disconnect button and will allow us to arrange them in a row (i.e., one next to the other) **(1)**, and one that will host the navigational buttons and will allow us to arrange them in a column (i.e., one under the other) **(2)**.

Let's have a closer look at the functionality of each button:

1) **Scan:** The button "Scan" should open a list of all the available Bluetooth Low Energy devices in the area. From that list, the user should choose Micro:bit's Bluetooth address. Then, the connection will be automatically established. This button differs since it will redirect the user to a list with all the available Bluetooth connections. To enable this feature, we will add a "ListPicker" button, which will be further described later in this guideline.

2) **Disconnect:** When the button "Disconnect" is pressed, the connection between the micro:bit and the user's smart device will be deactivated.

3) **Navigation Buttons (front, back, etc.):** When one of these buttons is pressed, our robotic car will move in the corresponding direction.

Tip: During this stage, you can advise/encourage your students to create a sketch or a diagram of the interface and the included components therein, or/and a list with all the needed items. In this way they will be able to better organize the steps towards the realization of the present task.

Creating the layout for placing/arranging the buttons

The first step towards the creation of our application is to set the two layouts that will host and arrange all the needed buttons and labels. To do that we will add two layout items, namely a Horizontal layout that will host the ListPicker and the Disconnect buttons, and a Vertical one that will host the navigational

Co-funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.

buttons. Therefore, from the "Layout" tab, we will drag the "HorizontalArrangement" **(1)** and the "VerticalArrangement" **(2)** components, and we will drop them on the screen (*Figure 2*).

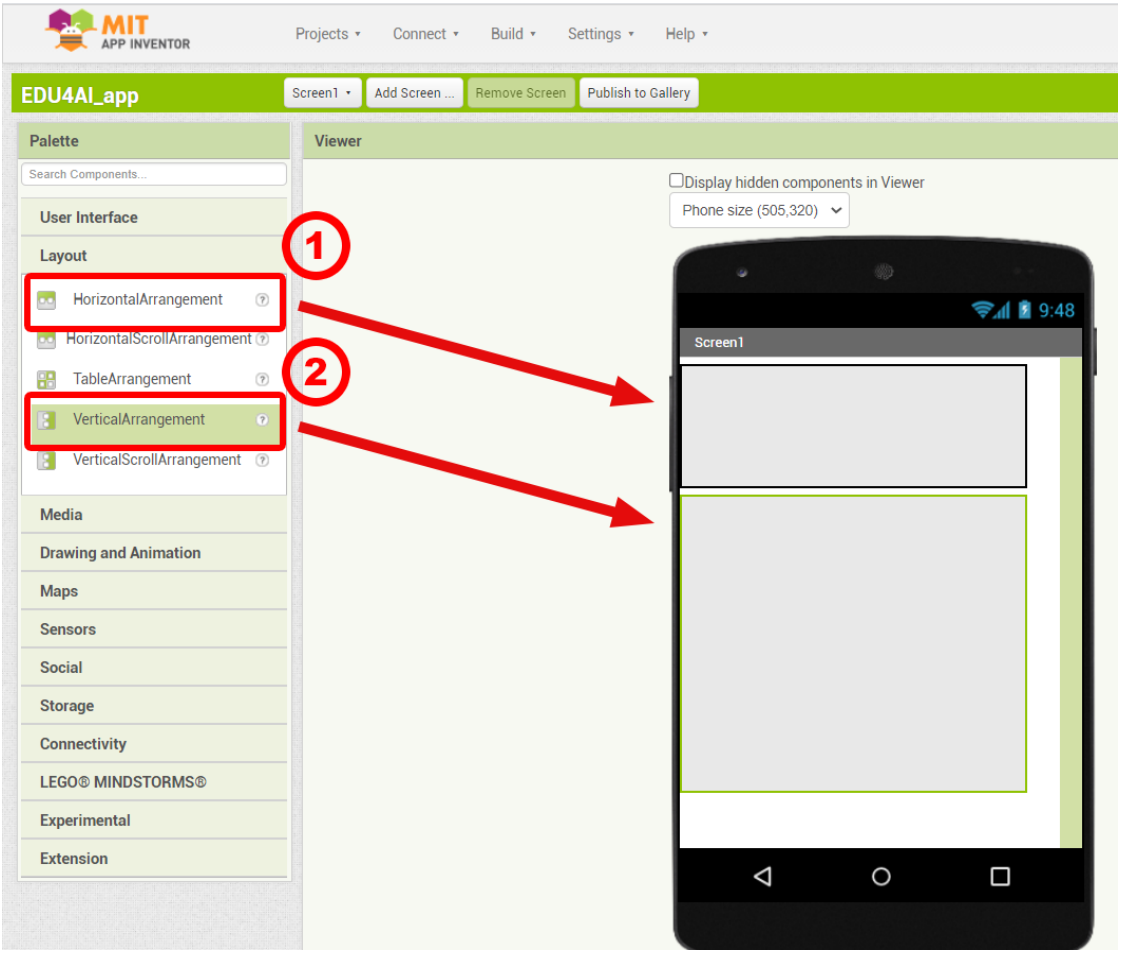


Figure 2: Dragging and dropping the two Layout items on the screen

After placing the two layout components on our screen, we can adjust a number of properties such as their height and their width from the "Properties" section **(4)**. To do that we need to select the corresponding component from the Components **(3)** list. In the example presented in Figure 3, we have set the width of the "HorizontalArrangement" component to 90 percent, and the width and the height of the "VerticalArrangement" component to 60 and 90 percent respectively.

Co-funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.

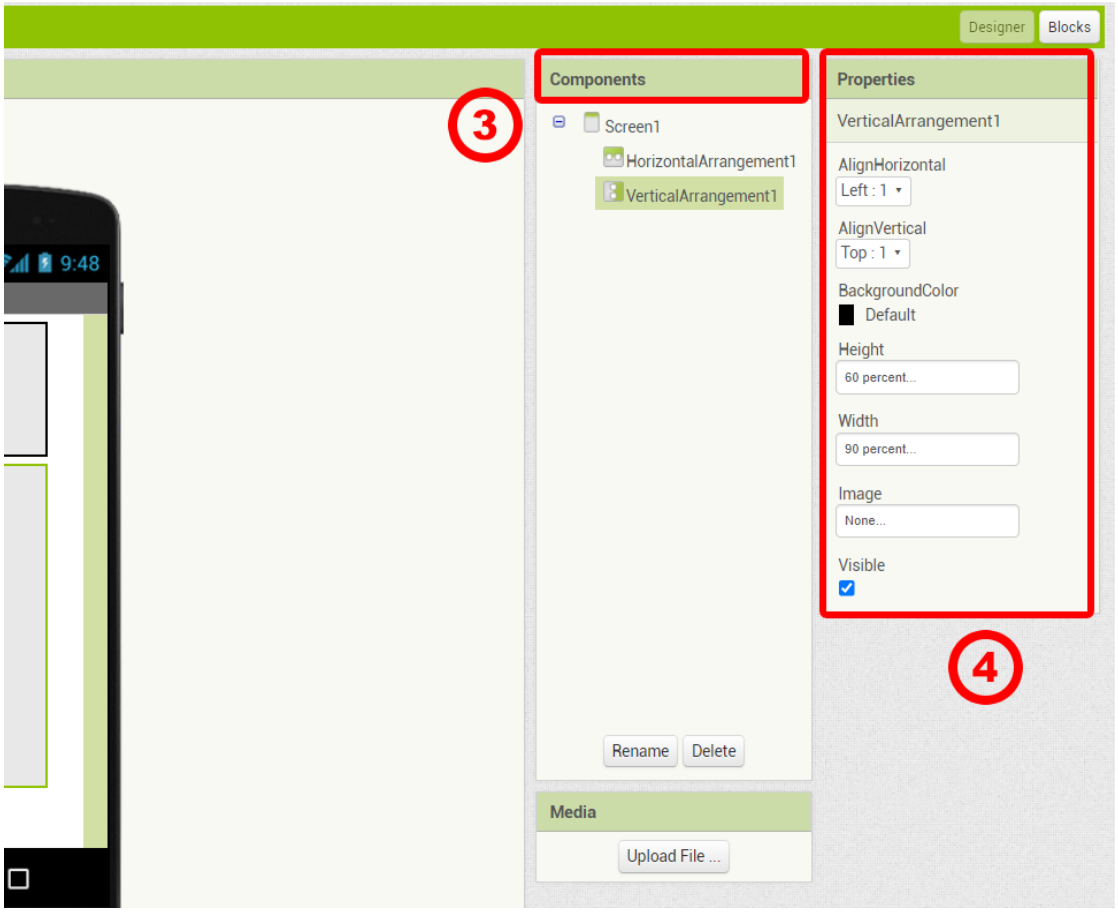


Figure 3: Adjusting the properties of each component

Tip: If you wish, you can revise and modify the aforementioned properties after adding the buttons inside the layout components.

Adding the ListPicker and the buttons

The next step is to add the ListPicker and the buttons on the screen. To do that we go to the "User Interface" sub-menu and from the "Palette" section, we will drag and drop on the screen 7 items, namely a "ListPicker" and 6 buttons. The ListPicker will be used to search/scan and reveal all the available Bluetooth devices.

Specifically, we drag the "ListPicker" **(2)** item and one button **(1)** (i.e., the button for disconnection) and drop them to the "HorizontalArrangement" layout, and after that we will drag 5 more buttons **(1)** (i.e. the navigational buttons) and drop them to the "VerticalArrangement" layout (*Figure 4*).

Co-funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.

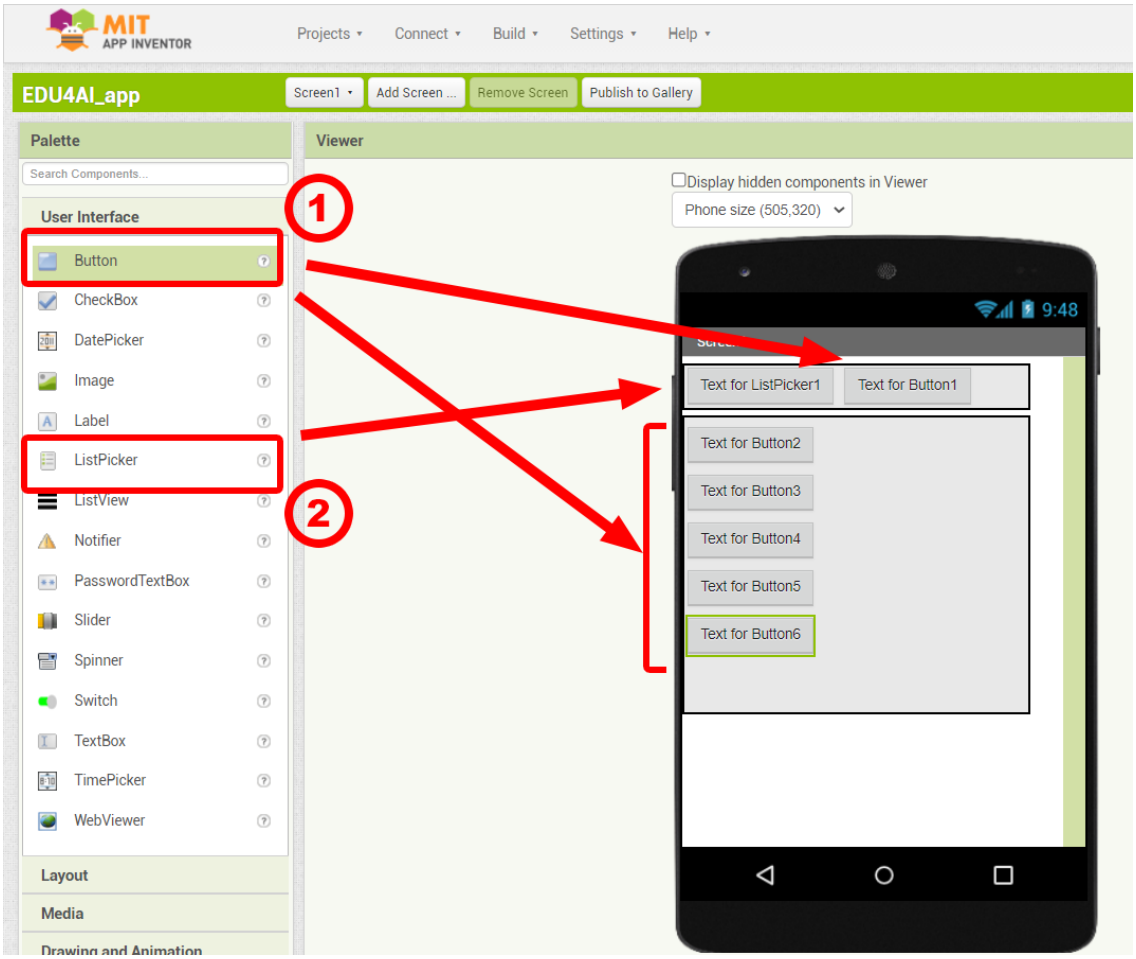


Figure 4: Adding a ListPicker item and 6 buttons on the screen

In order to optimize the look of our interface we can change the name of each button through the "Text" tab **(4)** located at the "Properties" menu. To do that, we need to select the corresponding item from the "Components" list **(3)** (Button 3 in our example) and then change the name manually (Figure 5).

Co-funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.

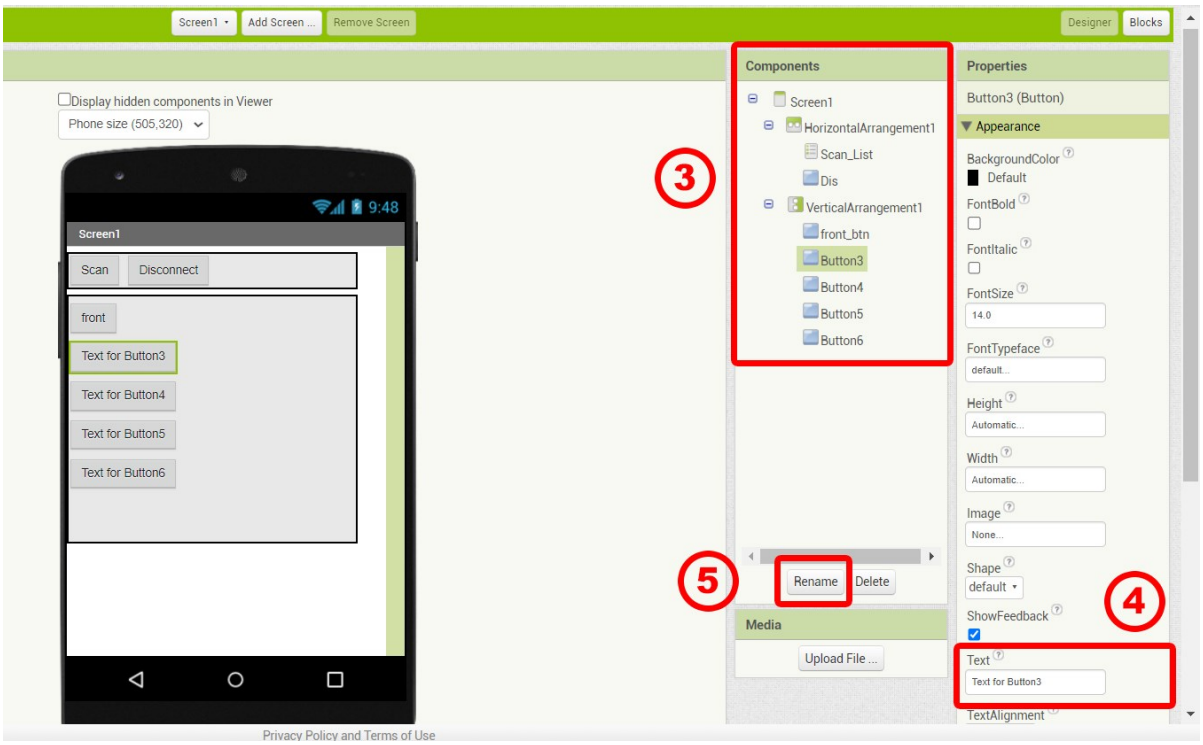


Figure 5: Changing the content of buttons' text

Another good practice is to change the buttons' names. This will also facilitate the coding phase later on. You can do that by clicking on the "Rename" **(5)** button, and by inserting the new name on the "New name" text box located at the pop-up menu (*Figure 5, Figure 6*).

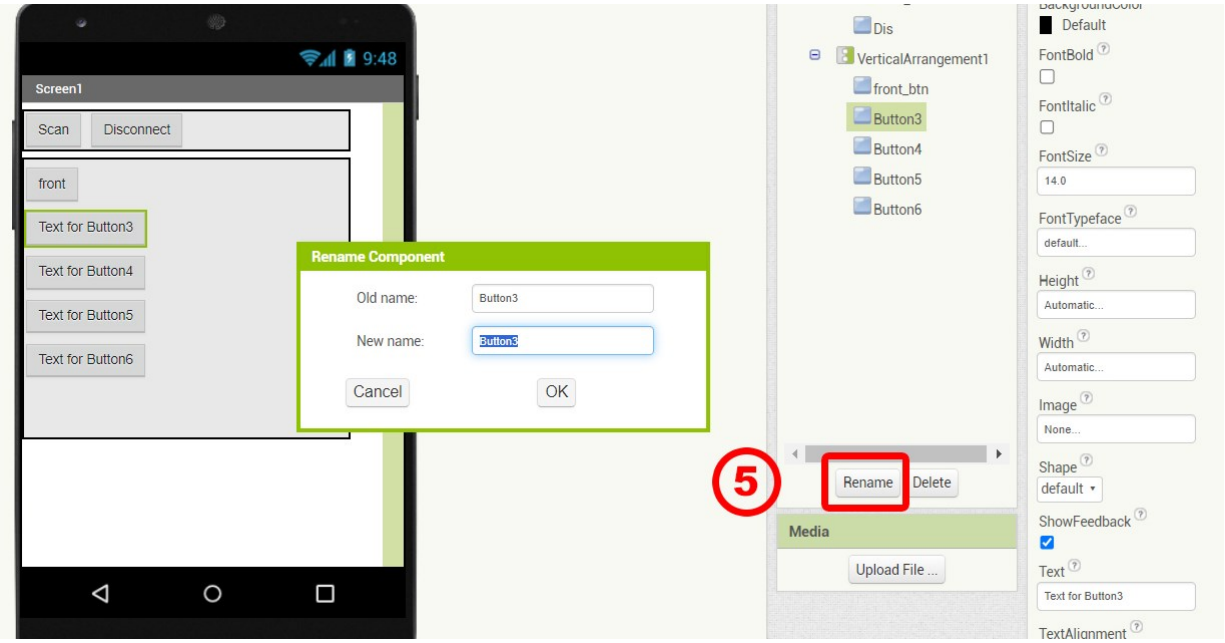


Figure 6: Renaming the Button component

Co-funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.

Tip: The names of the buttons are indicative and do not affect their functionality. However, a good practice is to use names that are meaningful for the entire process (e.g., give the name "front_btn" to the button that will move forward the car, "back_btn" to the one that will move backwards the car etc.). In our example we are renaming the components as following:

Component	**Text Name** (changed from Text tab in properties)	**New Name** (changed from Rename in components list)
ListPicker	Scan	Scan_List
Button1	Disconnect	Dis
Button2	front	front_btn
Button 3	back	back_btn
Button 4	right	right_btn
Button5	left	left_btn
Button6	stop	stop_btn

Important note: Do not use the same word for text name and button name, because this will cause a malfunction to App Inventor, disabling it to build the application.

To summarize, the ListPicker "Scan" will search for available Bluetooth devices, while the button "Disconnect" will stop the connection between our smart device and the Bluetooth module. The remaining buttons will be used to navigate our robotic car.

Tip: Keep in mind that the aforementioned guidelines are indicative. Feel free to experiment with different shapes and colors for each button, or even different arrangements, thus creating a more unique and visually appealing interface.

Adding a text notification

The next step is to add a text label to our screen. This step is not mandatory, but it is really useful, since it will be informing us whether the connection with our smart device is established or not. Find "Label" in "User Interface" sub-menu, and drag and drop it on the screen (*Figure 7*). Change the text content to "Waiting for input" (or to something similar of your preference in the same way that you changed the text content for the rest components.

Co-funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.

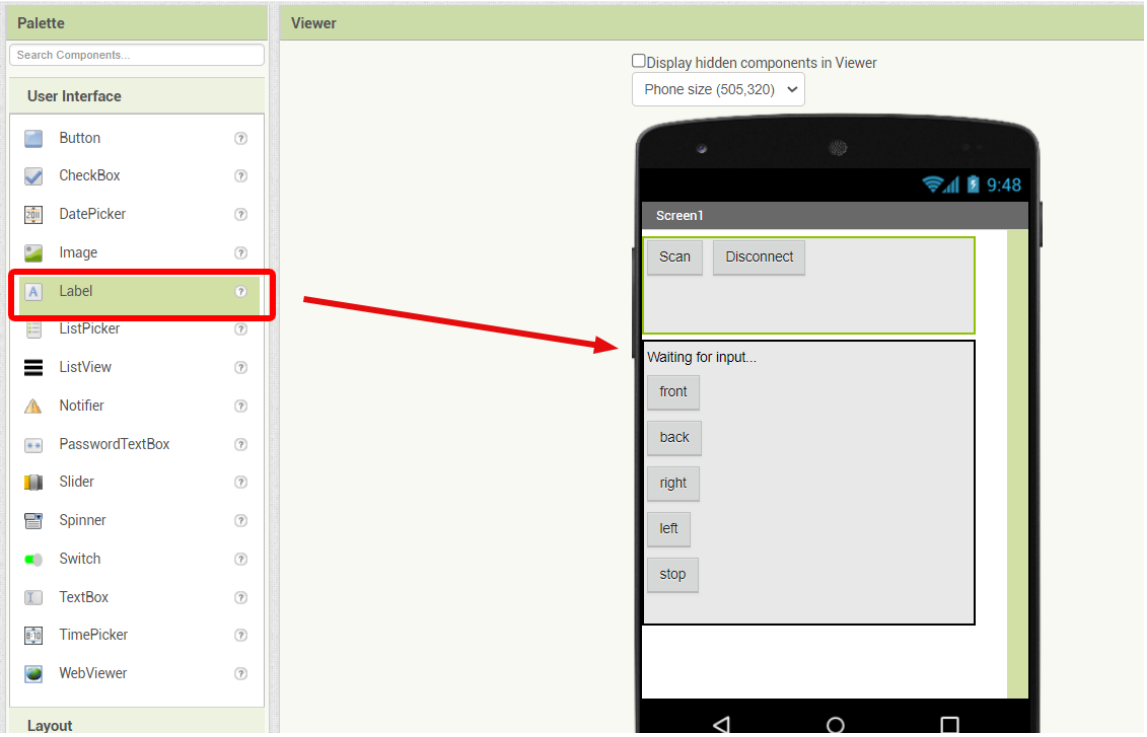


Figure 7: Adding a Label for informative purposes regarding connectivity

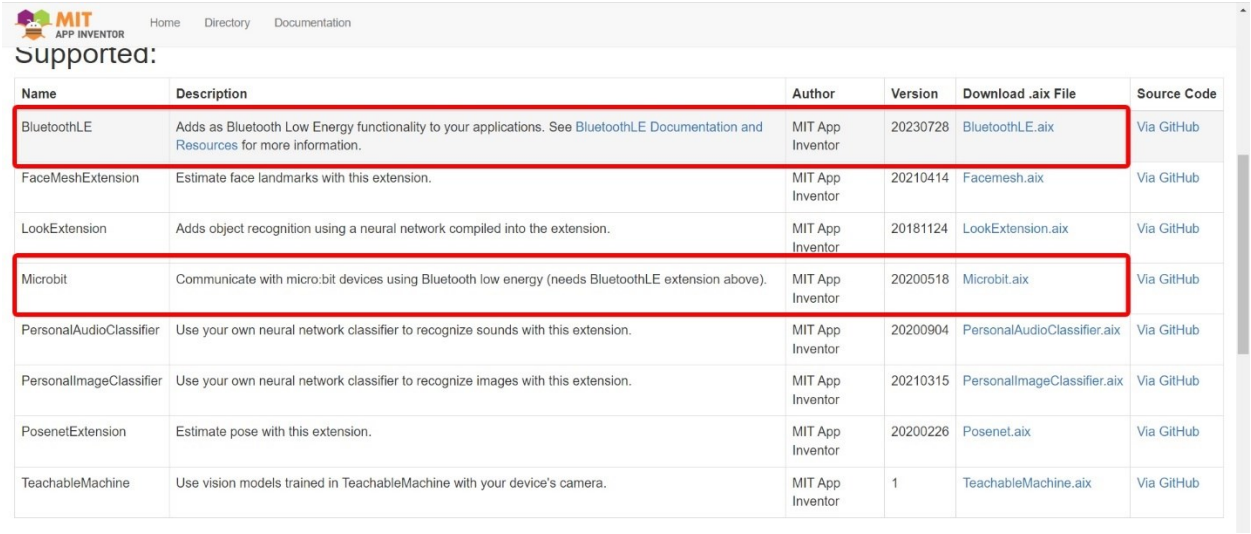
Adding Extensions

The next step is to add some components that will permit the connection between our application and the robotic car. To do that, we must add the following extensions to our application; the BluetoothLE extension and the Microbit_Uart_Simple. The first one is used for establishing the Bluetooth connection between our smart device and the Micro:bit, whereas the second one for sending the appropriate messages after the connection has been established.

To be able to use this extensions, you need to download them locally to your computer. To do that, click here https://mit-cml.github.io/extensions/ and download to your computer the *BluetoothLE.aix* file and the *Microbit.aix* file (*Figure 8*).

Co-funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.



Name	Description	Author	Version	Download .aix File	Source Code
BluetoothLE	Adds as Bluetooth Low Energy functionality to your applications. See BluetoothLE Documentation and Resources for more information.	MIT App Inventor	20230728	BluetoothLE.aix	Via GitHub
FaceMeshExtension	Estimate face landmarks with this extension.	MIT App Inventor	20210414	Facemesh.aix	Via GitHub
LookExtension	Adds object recognition using a neural network compiled into the extension.	MIT App Inventor	20181124	LookExtension.aix	Via GitHub
Microbit	Communicate with micro:bit devices using Bluetooth low energy (needs BluetoothLE extension above).	MIT App Inventor	20200518	Microbit.aix	Via GitHub
PersonalAudioClassifier	Use your own neural network classifier to recognize sounds with this extension.	MIT App Inventor	20200904	PersonalAudioClassifier.aix	Via GitHub
PersonalImageClassifier	Use your own neural network classifier to recognize images with this extension.	MIT App Inventor	20210315	PersonalImageClassifier.aix	Via GitHub
PosenetExtension	Estimate pose with this extension.	MIT App Inventor	20200226	Posenet.aix	Via GitHub
TeachableMachine	Use vision models trained in TeachableMachine with your device's camera.	MIT App Inventor	1	TeachableMachine.aix	Via GitHub

Figure 8: Finding the extensions to be downloaded

After downloading the extensions, return to App Inventor. On the Palette section, click on the Extension tab, and then click on the *Import extension* selection (**6**) (*Figure 9*)

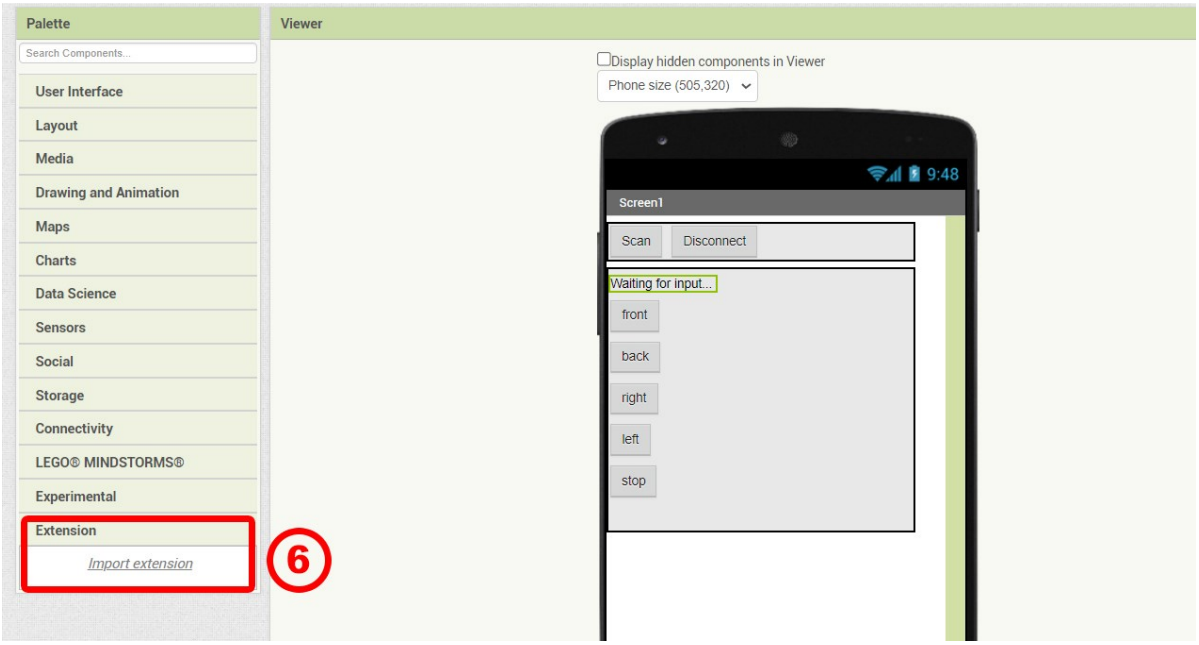


Figure 9: The Import extension selection

On the pop-up menu click on *Choose File* button (7) to search on your local folder and select the downloaded extension (*Figure 10*). Make sure that "*From my computer*", located above the *Choose File* button, is selected.

Co-funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.

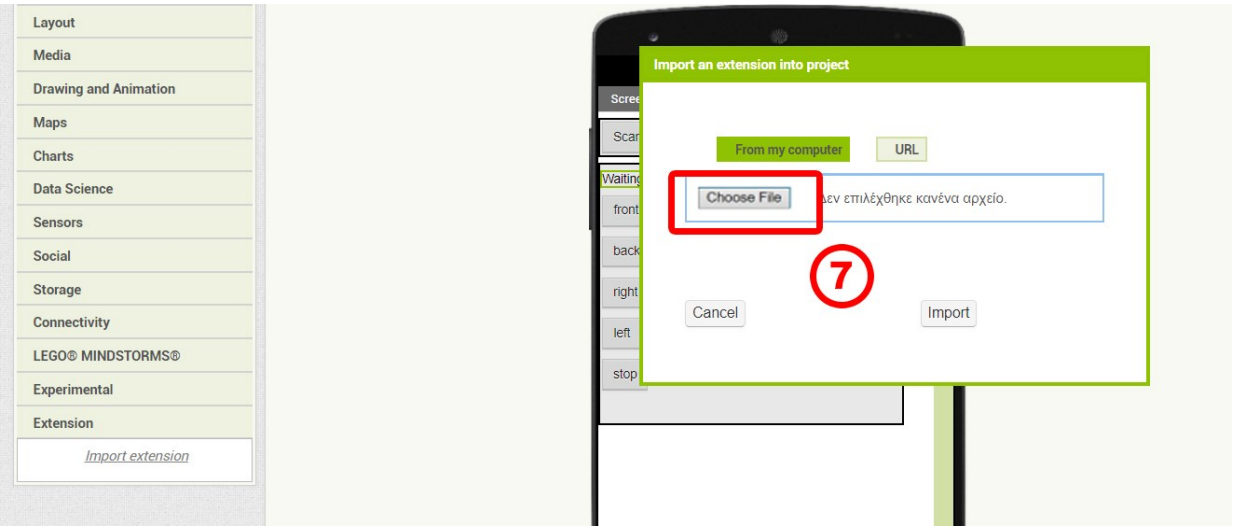


Figure 10: Click on Choose File button to search for the extension file on your local folder

After finding and selecting the extension file, click on the Import button (8) (*Figure 11*).

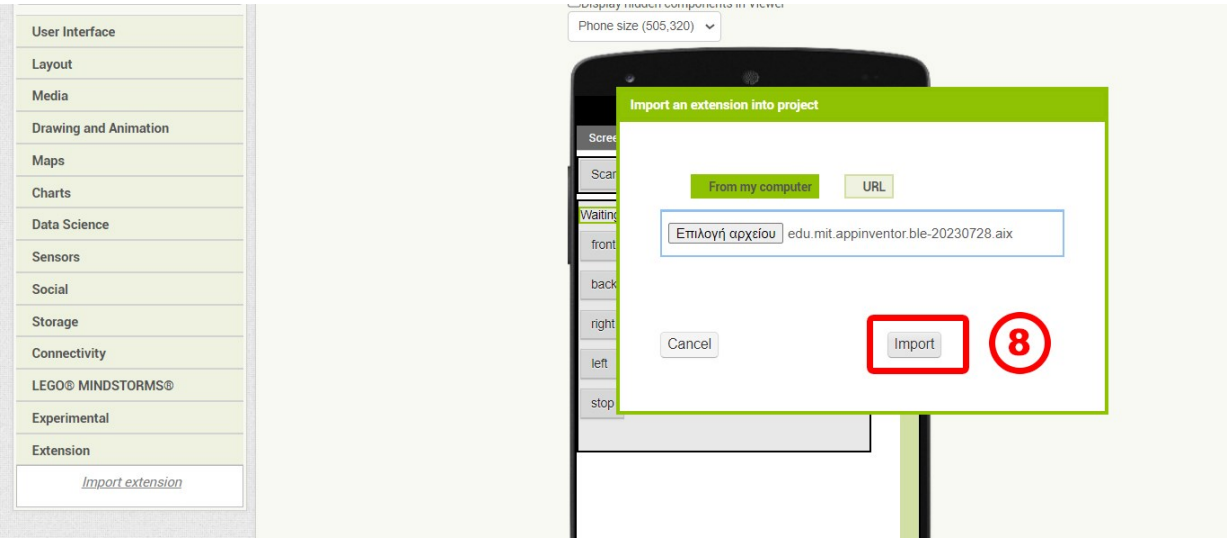


Figure 11: Click on the Import button to import the selected extension

After a while, the new extension will appear under the ***Import extension***, on the Extension tab (for example, in Figure 12, the BluetoothLE extension has been imported)

Co-funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.

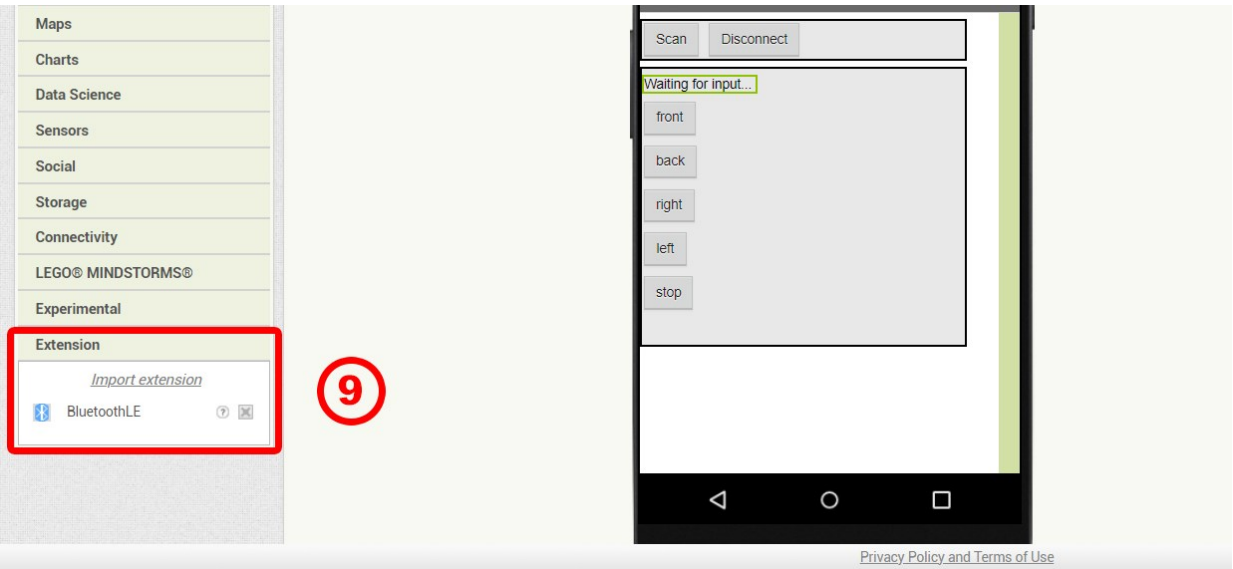


Figure 12: The BluetoothLE extension has been imported

To add this extension to the designed application, drag and drop it to the design area (*Figure 13*). The extensions are normally non-visible components. Therefore, these component appear under the design area, on the "Non-visible components" section.

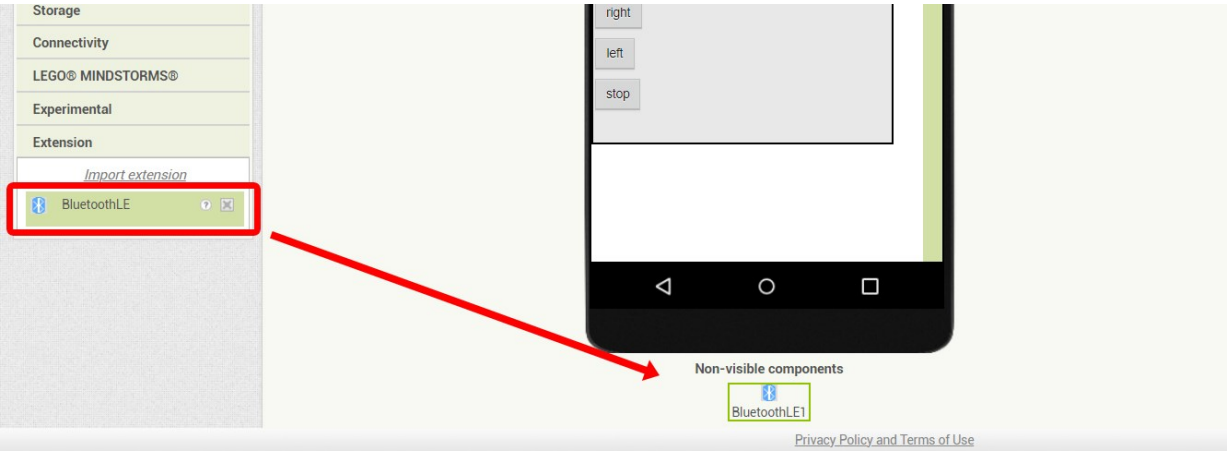


Figure 13: Dragging and dropping the BluetoothLE extension on the design area

Repeat the same process to import the Microbit_Uart_Simple extension.

Important note: after choosing and importing the *Microbit.aix* file, you will notice that several extensions will appear under the Extension tab. For the purposes of this activity, you will only need to drag and drop to the design area the Microbit_Uart_Simple extension (Figure 14).

Co-funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.

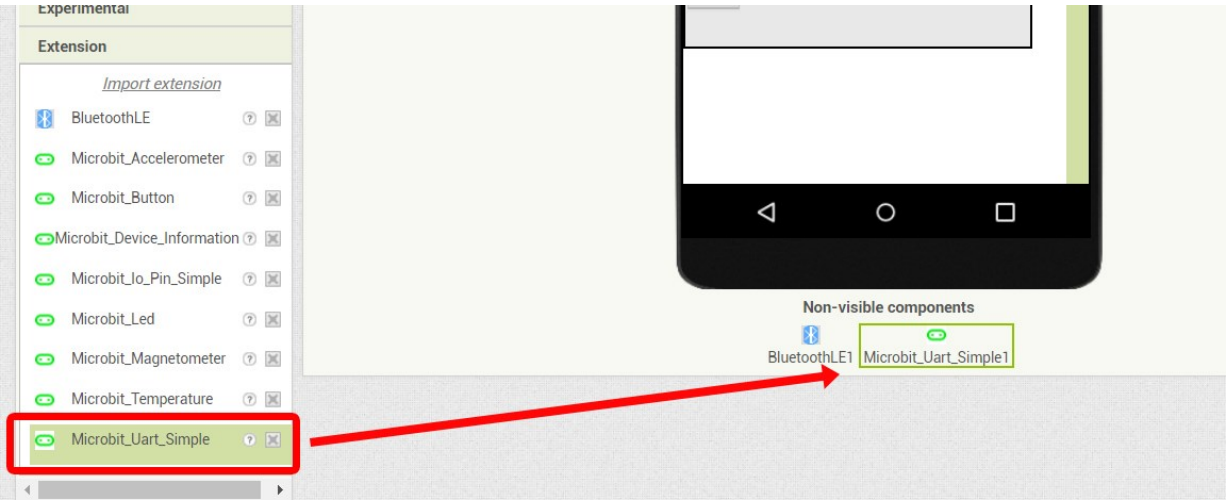
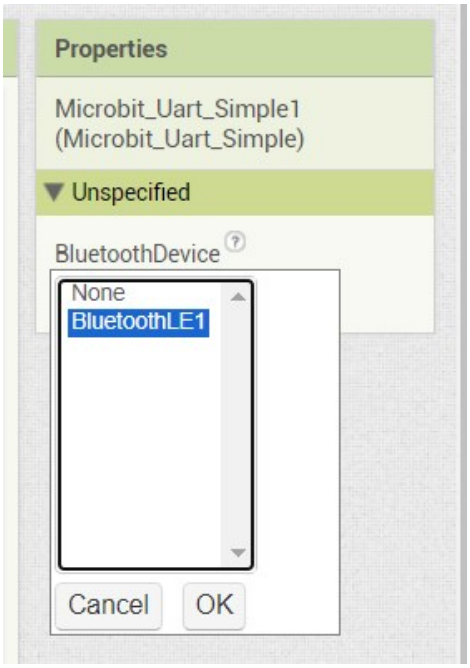


Figure 14: Only the Microbit_Uart_Simple is needed to be drag and dropped

After importing the Microbit_Uart_Simple extension, go to the Properties menu and on the Bluetooth device field, select BluetoothLE1 from the floating menu.



After adding all the needed extension, we are ready to move on to the programming of the application.

Co-funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.

1.4 Programming the Application

In order to make our application work, we need to determine how the added components will function by specifying what actions they will trigger when they are pressed (on the screen of our application). To do that we need to go to the Blocks menu and create our code on the viewer space (*Figure 21*) by dragging and assembling the suitable blocks of commands.

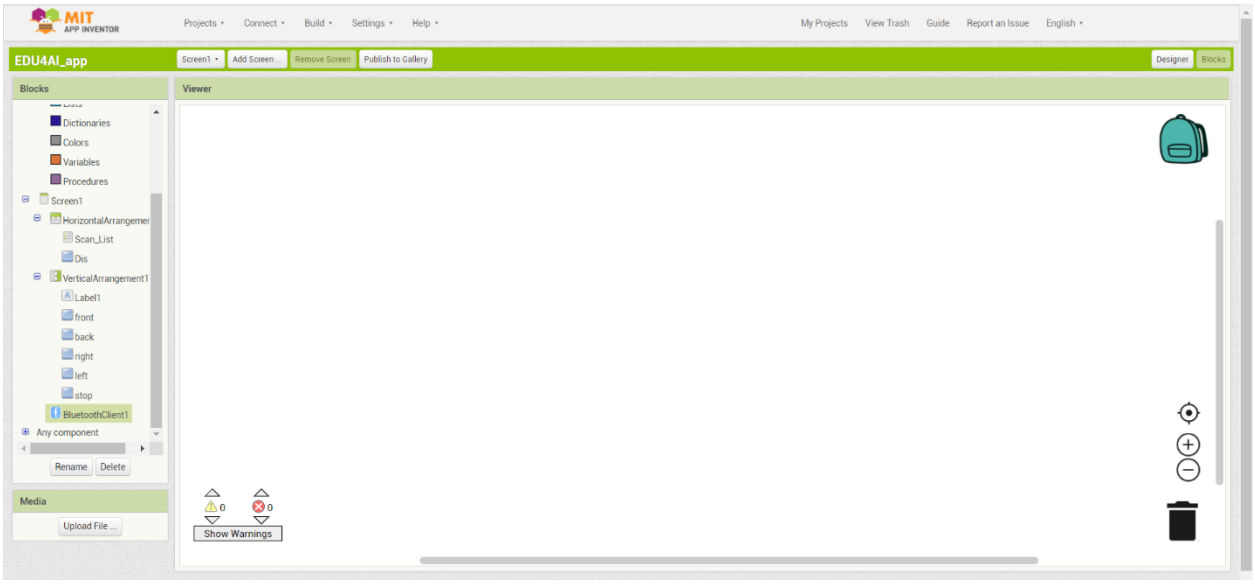


Figure 15: App Inventor's Block menu

Coding the ListPicker (i.e., Scan_List) component

We will start by coding the ListPicker component, or the "Scan_List" component, as named in our example. The following diagram depicts the operations/actions that we want to be performed when the ListPicker component is pressed, and the corresponding **event** command that should be applied for this purpose. So, let's see how these codes will be structured.

BeforePicking command

Co-funded by the European Union

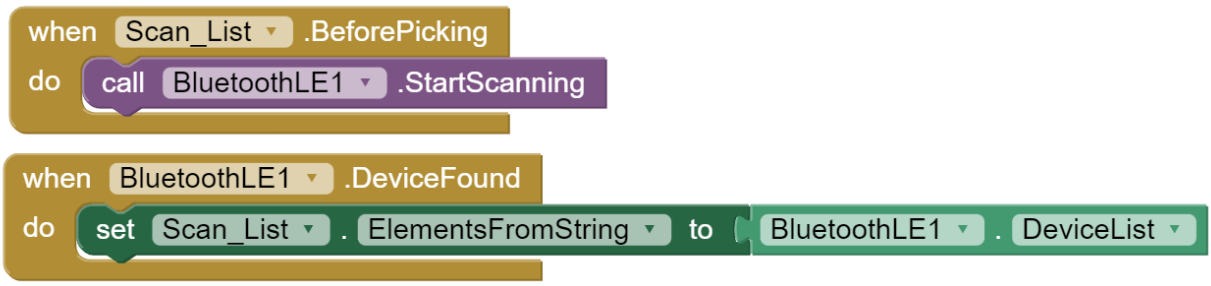
Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.

Click on the "Scan_List" item **(1)** and from the floating menu, select the event command **"When Scan_List BeforePicking" (2)** (*Figure 16*).

When Scan_List is pressed, a list of the available Bluetooth devices should come up. "BeforePicking" means that we have not yet picked any of the shown Bluetooth devices.

Figure 16: Clicking on the Scan List item and selecting the needed command from the floating menu

In order to be able to pick elements from our list, the list should be initially filled with all the available Bluetooth Devices. (The list is empty in the beginning, so before we pick something, it has to be filled first). To do that, we need to start scanning for BLE devices. Then, the list is filled with every available device that is found:



Note: The blocks of commands used on the script above can be similarly found, by clicking on the corresponding item (Scan_List or BluetoothLE1) and by finding the relevant blocks from the corresponding floating menu (some examples are presented in *Figure 17*).

Co-funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.

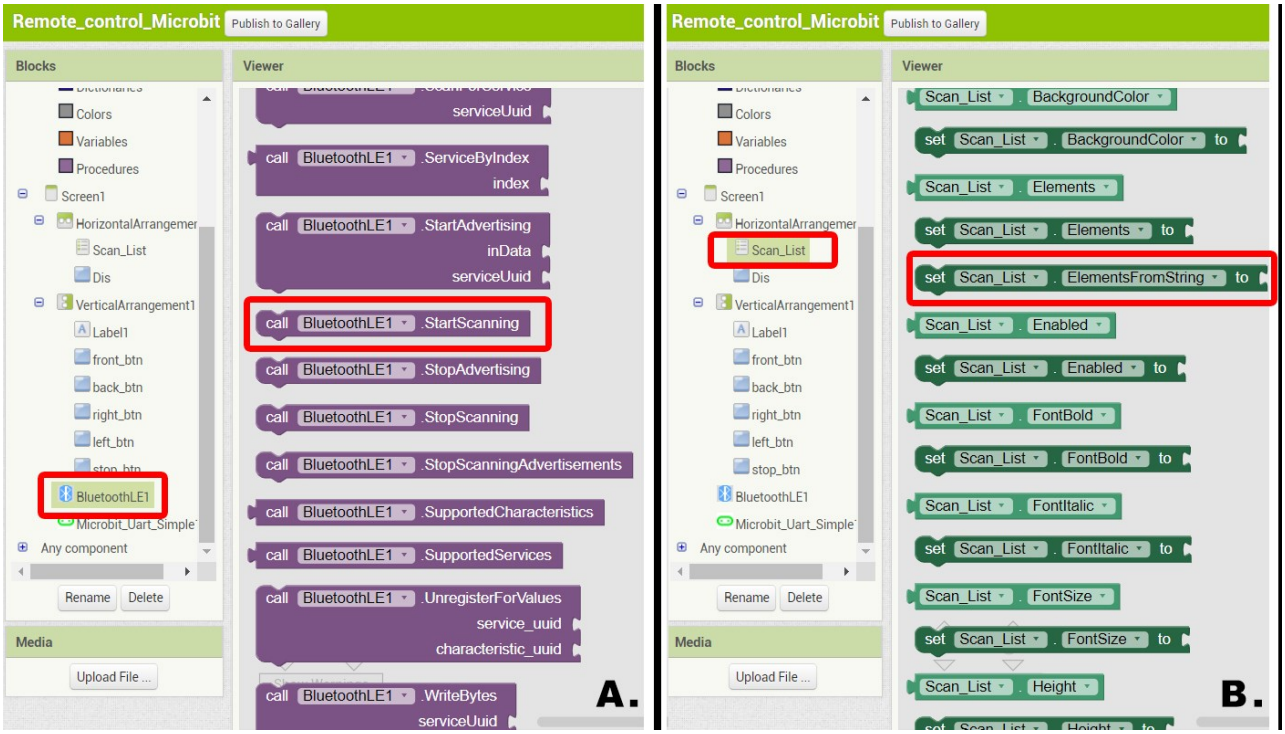


Figure 17: Finding some of the block commands

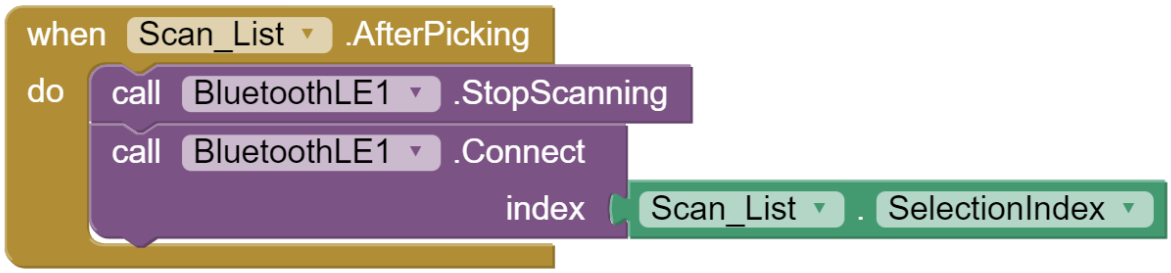
AfterPicking command

Now, we should determine what happens when the user selects the micro:bit's Bluetooth address in order to establish connection between the application and the robotic car. To do that, we will need the event command **"When Scan_List AfterPicking".**

Inside the **event command** we will place some commands that will determine the action that will be performed after we have picked a Bluetooth Address from the list. Therefore, we will use the **"call BluetoothLE1.StopScanning"** command, so that our device stops scanning for other Bluetooth LE devices. Then we will add the **"call BluetoothLE1.Connect"** command, and in its right side we will snap the **"Scan_List.SelectionIndex"** command, to enable the application to be connected to the selected Bluetooth device.

Co-funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.



Coding the Label (i.e. Label1)

To make sure that the Bluetooth connection has been established, we will program the application to send a relevant notification. This will be done by programming the Label item, and by using the following blocks of code:

In particular, we use the **"set Label1.Text to"** command and on its right side we snap a **text ""** block, in which we type "Connection Established".

If the connection is successful, the notification "Connection Established" will appear on the screen, replacing the "Waiting for input…" message (Figure 7). If the connection is not successfully established, then the "Waiting for input…" message, will not change.

So, after the selection of the Bluetooth device is made (namely the Bluetooth of the micro:bit), the BluetoothLE component will attempt to connect to the preferred address (i.e., the micro:bit that the robotic car is using). If the attempt is successful, the Label's text "Waiting for input…" will change to "Connection established!".

Note: text input block can be found at the floating menu of "Text" tab (*Figure 18*).

Co-funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.

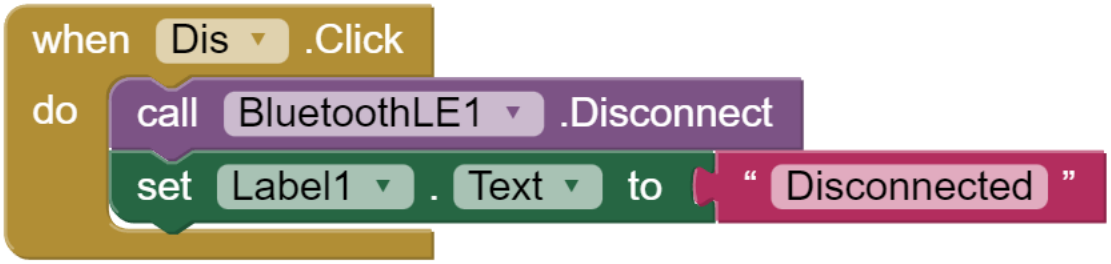
Figure 18: Finding the text input block

Coding the Disconnect (i.e., Dis) button

In this step we will program the BluetoothLE component to terminate the connection, when the "Disconnect" button is pressed. The Label's text will also change to "Connection timed out".

For this step of coding we will need the event command **"When Dis .Click…do"**.

So, through the following script, **when** the button "Disconnect" is pressed, the application is instructed to **call the BluetoothLE** and **disconnect** it from the pre-selected device. Also, we **set the Label's text to "Disconnected"**, in order to inform the user that the connection has been terminated.



Note: The script for activating the "Scan" and "Disconnect" buttons may be difficult for your students. Therefore, depending on their level, you can provide them with these parts of the script, and teach them the following part of the script thoroughly.

Coding the buttons for navigating the robotic car

The last step is to program the navigation buttons. As mentioned earlier in this document, when a navigational button is pressed, our smart device will transmit (via Bluetooth) a specific message to our robotic car's micro:bit board. When this message is received the robotic car will behave accordingly, by performing a specific movement (i.e,, moving forward, backward etc.). The following table presents which message is transmitted when a specific button is pressed:

Co-funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.

Button that is pressed	*Word that is sent*
“front”	“#forward#”
“back”	“#backwards#”
“right”	“#right#”
“left”	“#left#”
“stop”	“#stop#”

Important note: Every message that we want to transmit must start and end with the symbol **“#”**, so that the micro:bit can distinguish the message’s boundaries.

For example, when we transmit the word “#forward#”, the micro:bit will spin the DC Motors forward.

For this part of the script, and for each one of the buttons, we will need the event command **“When .Click…do”** and an **“if..then”** condition.

Note: “if-then” block condition can be found under the “Control” tab (*Figure 19*).

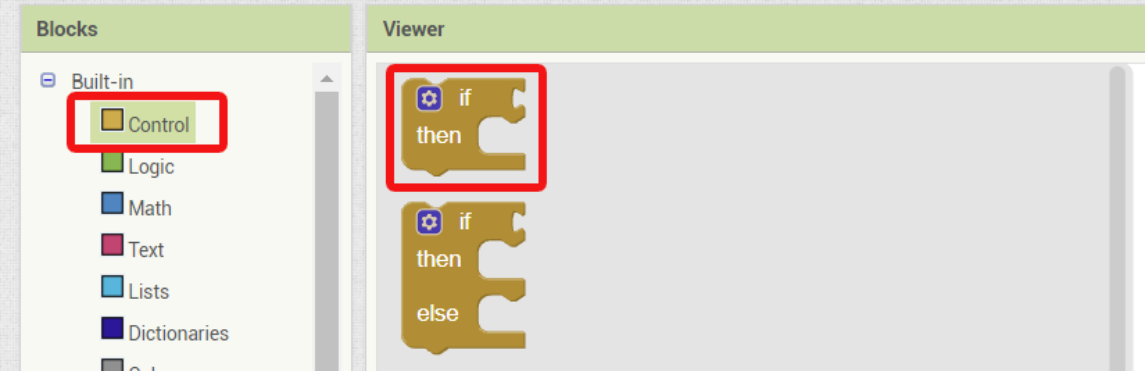
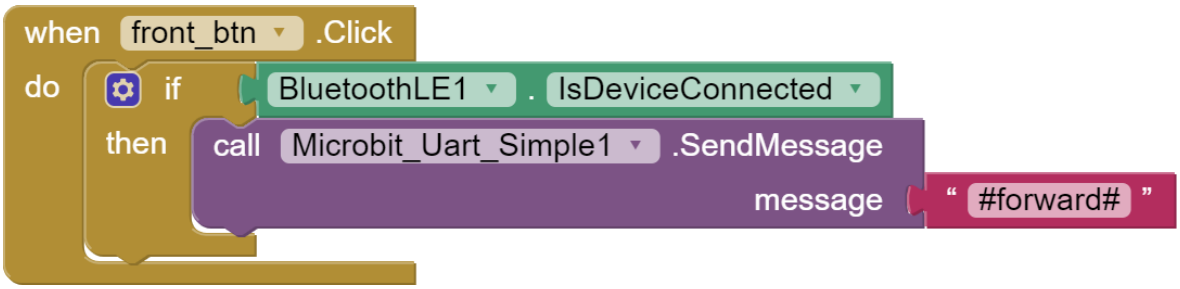


Figure 19: Finding the “if...then” block condition

So, through the following block of commands we instruct our application to check whether the **BluetoothLE** component has **connected to** the desired **device**, and **if** it does, **then** the corresponding **message** (**“#forward#”** in this example) is **sent** by **calling** the **Microbit_Uart_Simple** component.

Co-funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.



Note: the "forward" value, written inside the text, is a value declared on the script created on the MakeCode programming environment, and assigned to the "moving forward" movement.

Repeat the same process for the other four buttons. The result should be similar to the one depicted in *Figure 20*.

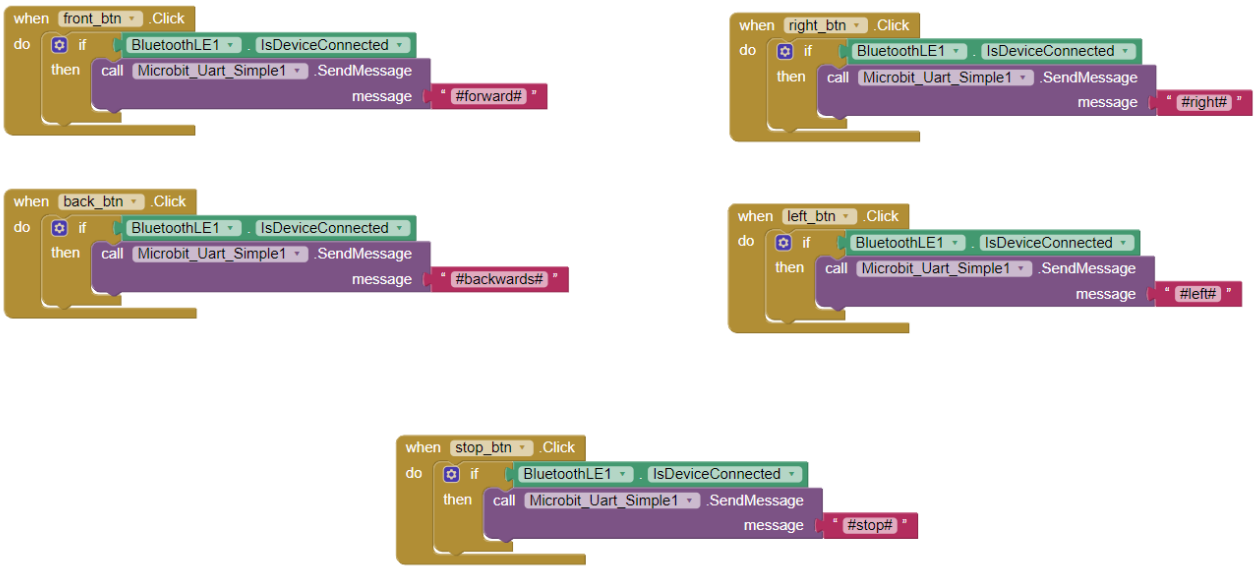


Figure 20: Creating the scripts for all the buttons of the application

When you finish all the aforementioned steps the entire script should look like the one depicted in *Figure 21*.

Co-funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.

Figure 21: The entire script for this warm up activity

Now that the application is ready, you can build the application and uploaded to your smart device.

1.5 Building the application

When you finish all the aforementioned steps (and the entire code is similar to the one depicted in *Figure 21*), the application is ready to be uploaded and installed to your smart device.

Thus, go to the *Build menu* ***(1)*** (*Figure 22a*) and select "*Android App (.apk)*" from the drop-down menu to begin the process of the .apk file production. This might take a couple of minutes.

When the building process has been finalized, a new window will pop-up **(2)** (*Figure 22a*). You can either choose to download the produced .apk file, or you can scan with your smart device the embedded QR Code, through the **MIT AI2 Companion app** (*Figure 22b*), that you should previously have download and install to your smart device.

Note 1: MIT AI2 Companion is an application/service that is available (for free) and you can find it to the "Play Store" service of your smart device. This application acts as a mediator, facilitating the successful installation of the produced .apk file to your smart device. This application is only available for Android devices.

Note 2: During the installation of the .apk file, a number of notifications regarding the safety/security of this file might appear. Ignore all of them and ask your device to proceed with the installation process.

Co-funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.

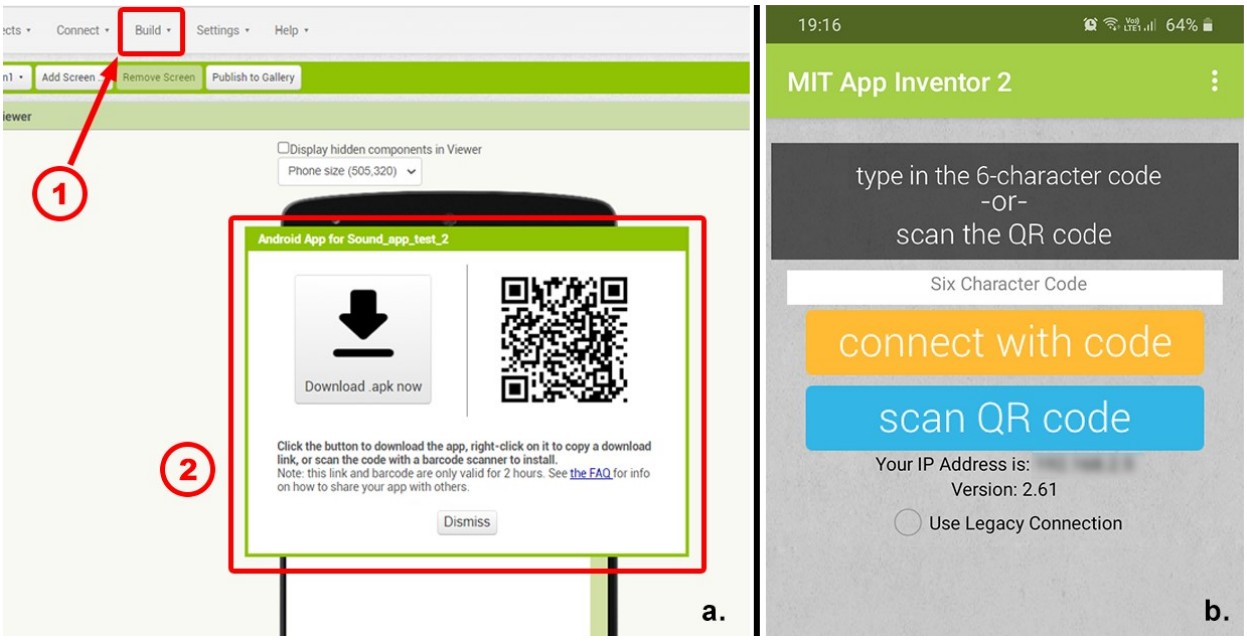


Figure 22: a. QR code produced after completing the building process; b. Screenshot of the MIT AI2 Companion interface

After successfully installing the application to your smart device, you can test if it works properly. **Remember** that you need to have downloaded to your robotic car the script produced in the Makecode programming environment.

1.6 Pairing the application with the robotic car

As mentioned above, to connect the application to the robotic car, you need to press the “Scan” button and select the micro:bit’s Bluetooth address from the list. However, it is likely that the micro:bit will not be included in the list of available Bluetooth devices. To solve this problem, go back to the main menu of the application and **activate** the “airplane” mode on your smart device for a few seconds. Then **deactivate** it and press the Scan button again. Micro:bit’s Bluetooth address will now be available.

Note: Make sure “Location” is also activated to your smart device

Co-funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.