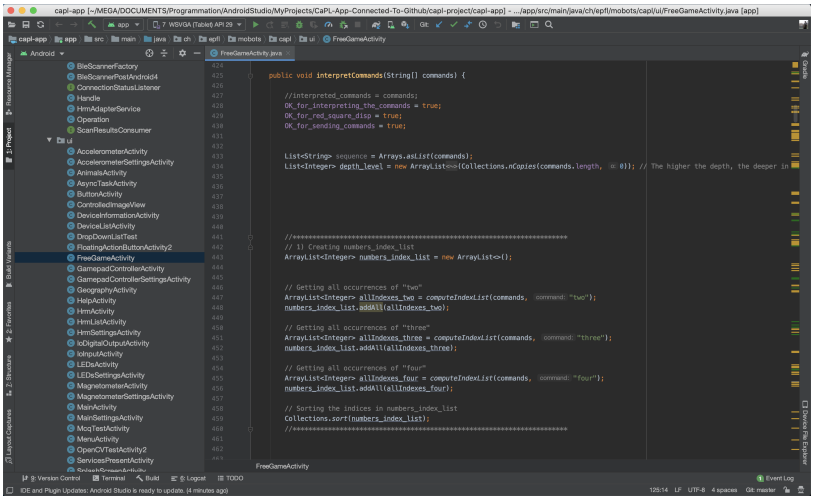
Semester project – Spring 2020

CaPL app framework - documentation



Professor:
Francesco Mondada

Student:
Anthony Guinchard

Assistants:
Christian Giang
Laila El Hamamsy

July 10, 2020

Abstract

This document summarizes the general operation of the capl-app [1] and the necessary adaptations to be made in the capl-framework-app to use a robot other than a micro:bit-based robot like the Cardbot. It aims at providing sufficient information about the Bluetooth communication between the tablet and the micro:bit to be able to reproduce this scheme with a robot working with another microcontroller.

Contents

List of Figures

1 Working principle of capl-app

The capl-app is made up of two main parts. The first part covers everything related to computer vision. It uses the `ComputerVision.java` class to identify and then interpret a sequence of commands from a photo. This is the core working of the app: identifying (analysing the number of tiles and their shapes) and interpreting the tangible tiles (in case 'for loops' are present). Part A of figure 1 represents the computer vision phase. The second part is the transmission of the commands to be executed to the robot. In the figure 1, this is part B. In the case of the original version of the app, the robot used is based on micro:bit. The Bluetooth connection and communication part is therefore made to recognize and work with a micro:bit. However, the same principle can be used to connect with any device including a Bluetooth module.

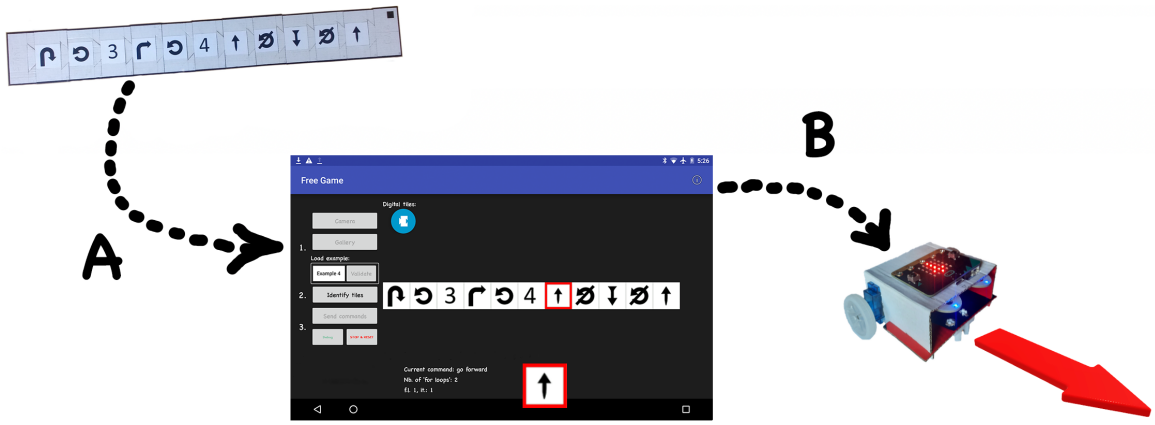


Figure 1: Basic operation of capl-app

Parts A and B are completely decoupled and work independently. The application first analyzes and interprets the entire command sequence from the photo and only then sends each command to the robot one by one. The java class designed for the type of device used with the new robot can be adapted if necessary. In the case of the original app, the `MicroBit.java` and `MicroBitEvent.java` classes are used to set up the transmission of Bluetooth events from the tablet to the micro:bit-based robot. More details about the files and code parts concerned will be given in chapter 3.

As a reminder of how Bluetooth communication works in the CaPL system, the developer should be aware that the tablet and the micro:bit communicate via a client/server architecture using ATT protocol (Attribute Protocol) as depicted in figure 2. First, the tablet subscribes to the micro:bit that is nearby and selected by the user. This is done with the `DeviceListActivity.java` situated in the folder "app > src > main > java > ch > epfl > mobots > capl > ui". Then the two devices can exchange information. But in the case of the capl-app, the information only goes in one direction: from the tablet to the micro:bit. The tablet sends Bluetooth events corresponding to a certain numerical value (see `Settings.java`, line 33 and following) to the micro:bit.

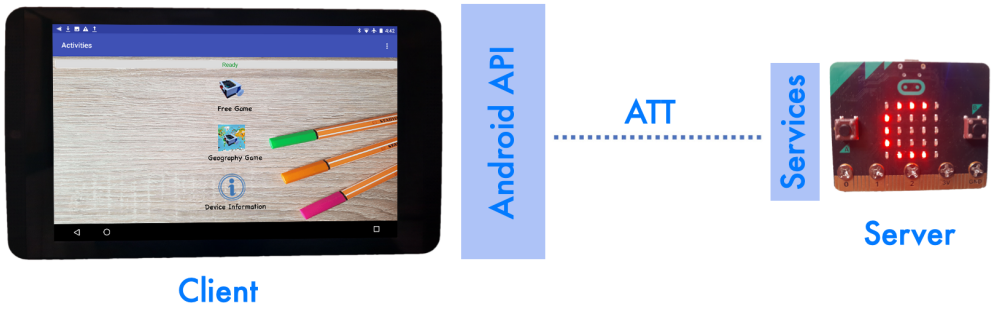


Figure 2: Client/server architecture between an Android device and a micro:bit

Depending on the Bluetooth events received and the value they carry, the micro:bit triggers certain processes that have an immediate effect on the robot's behavior. Let's take the example of the "go_forward" command which has been identified by the computer vision algorithm and which is currently sent to the micro:bit-based robot in figure 1. In the `FreeGameActivity.java` activity, by means of a "switch statement", the currently interpreted command (which is this "go_forward") is matched with the string "go_forward". To the variable `event_value` is then assigned a particular numerical value that corresponds to make the robot move forward once the signal is received by the micro:bit. This process simply unlocks the part of code used to make the robot move forward a certain distance. The block-based code used to make the robot perform this action is depicted in figure 3.

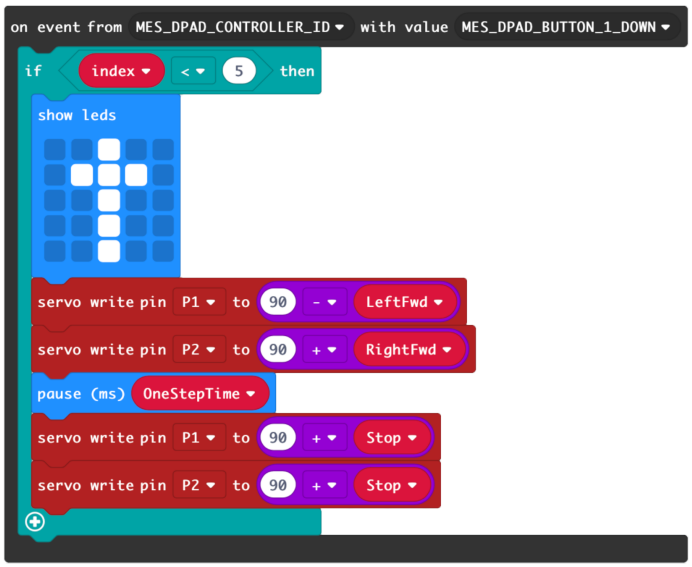


Figure 3: Block-based code to make the robot go forward

Thus, depending on the event received (or more precisely, in the case of the capl-app, depending on the numerical value carried by the `MES_DPAD_CONTROLLER_ID` event), the micro:bit triggers one or the other of the desired actions. These actions are visible in figure 4 where we can see that the robot will move forward if it receives the Bluetooth event `MES_DPAD_CONTROLLER_ID` with the value `MES_DPAD_BUTTON_1_DOWN`. Similarly, other robot movements can thus be triggered.

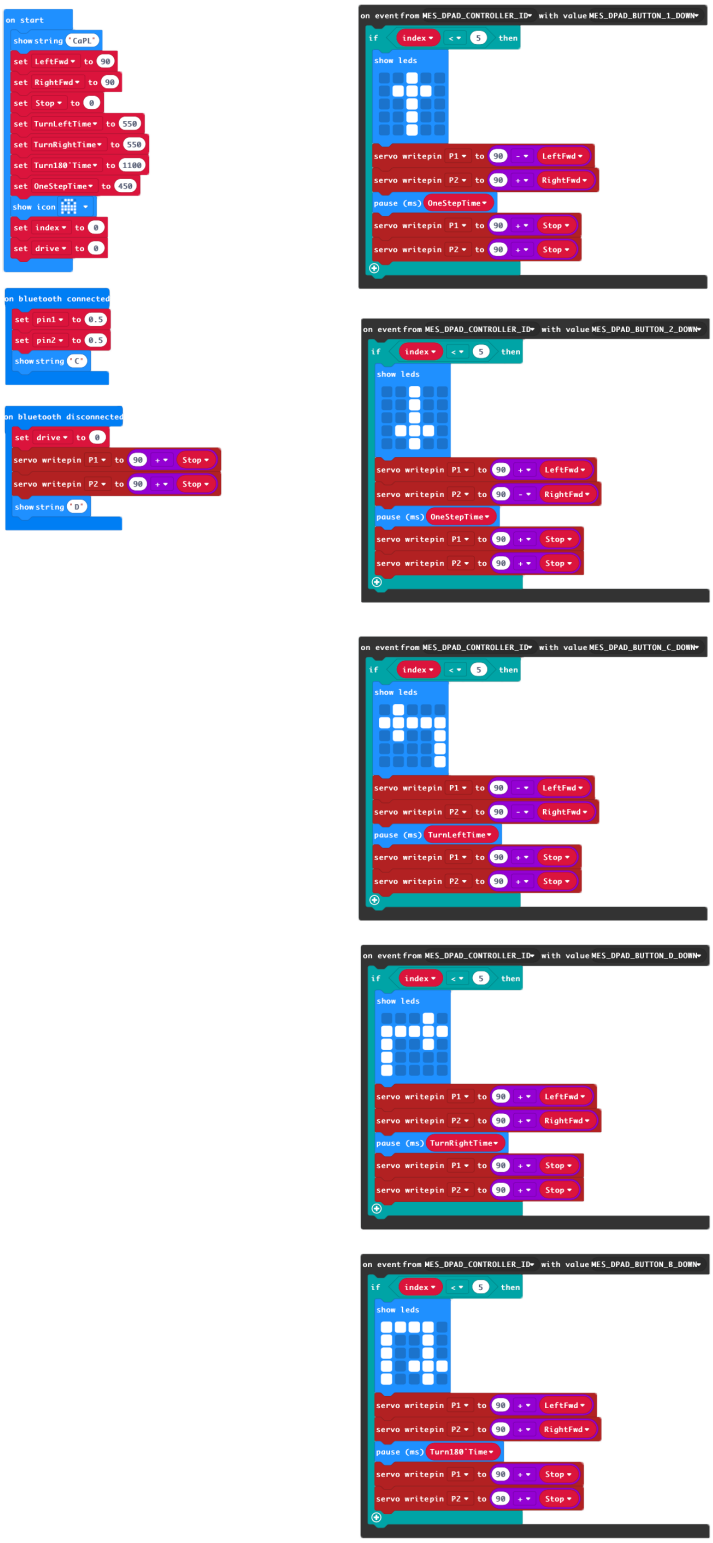


Figure 4: Block-based code flashed on the micro:bit of the micro:bit-based robot

2 App architecture

2.1 Original capl-app architecture

The original app is composed of several activities including two important games: the “Free Game“ and the “Geography Game“. The main activities of the capl-app can be selected from the `MenuActivity.java` activity. This can be seen in figure 5.

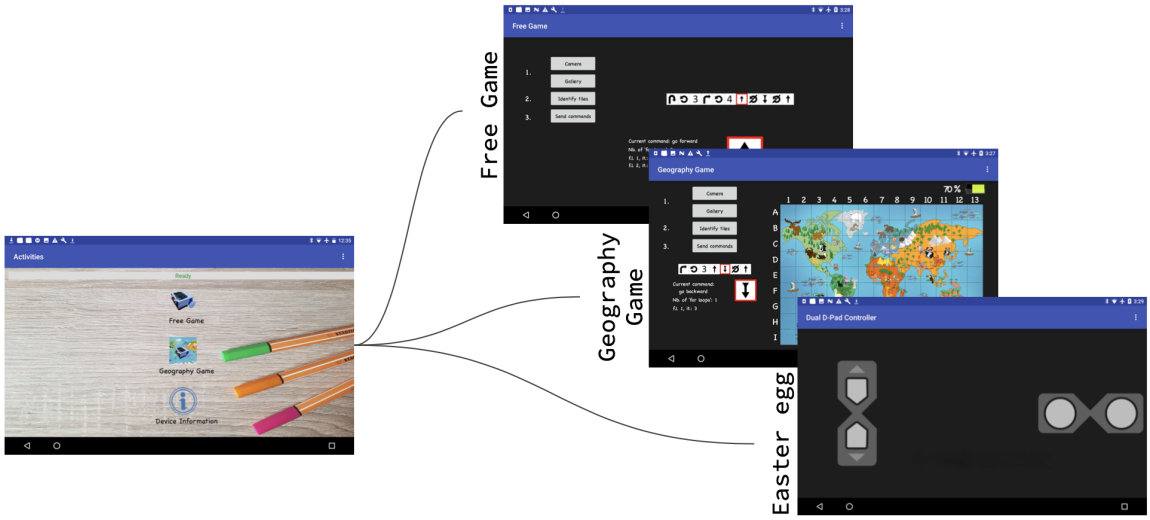


Figure 5: capl-app workflow

This app is based on the Android *micro:bit Blue* app [2]. It is therefore made to work with micro:bit-based robots. The possibilities of integrating other types of robots are developed in chapter 3. The two important folders that the developer should consider carefully are the one containing the java classes and activities (“capl“) and the one containing the layouts of these activities (“layout“). These two folders are located as depicted in figure 6 in Android Studio.

- app > src > main > java > ch > epfl > mobots > capl
- app > src > main > res > layout

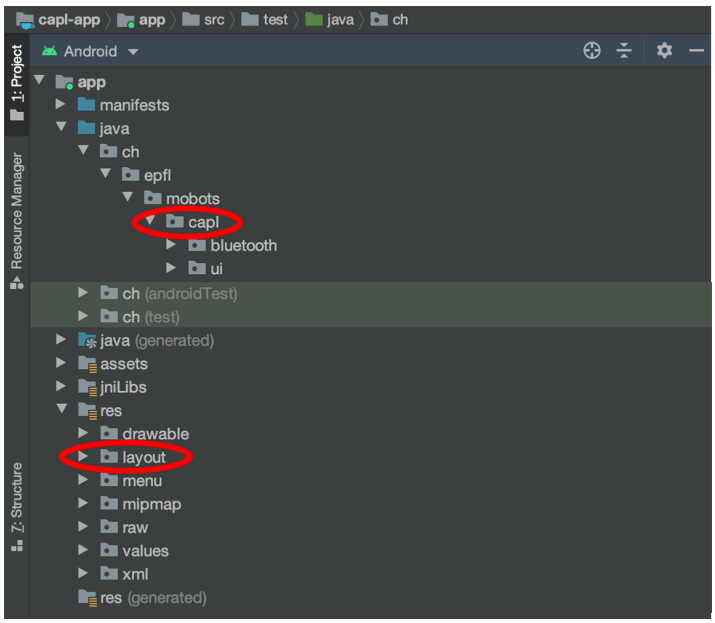


Figure 6: Location of the "capl" and "layout" folders in the "capl-app"

The computer vision functions (to recognize the tangible tiles and identify the number of tokens in the "Geography Game") are defined in the `ComputerVision.java` class that is situated in the "capl" folder whose path is shown above. This class can be customized to enhance robustness and execution time of the computer vision algorithms. The developer can also add other micro:bit related activities by customizing the basic activities of the *micro:bit Blue* app. Indeed, As it can be seen in the file `activity_menu.xml` (i.e. the layout of the `MenuActivity.java` activity), the default activities of the *micro:bit Blue* app are still present, but are just made invisible. This is to facilitate the integration of these activities in case a derivative version of the capl-app requires e.g. an interface to the accelerometer, the temperature sensor or the LED matrix of the micro:bit. These extra hidden activities are situated in the folder "app > src > main > java > ch > epfl > mobots > capl > ui" and are listed below:

- `AccelerometerActivity.java`
- `AccelerometerSettingsActivity.java`
- `AnimalsActivity.java`
- `ButtonActivity.java`
- `GamepadControllerSettingsActivitiy.java`
- `HrmActivity.java`
- `HrmListActivity.java`
- `HrmSettingsActivity.java`
- `IoDigitalOutputActivity.java`

- `IoInputActivity.java`
- `LEDsActivity.java`
- `LEDsSettingsActivity.java`
- `MagnetometerActivity.java`
- `MagnetometerSettingsActivity.java`
- `MainSettingsActivity.java`
- `TemperatureActivity.java`
- `TemperatureAlarmSettingsActivity.java`
- `TrivaScoreBoardControllerActivity.java`
- `UartAvmActivity.java`

Most of the other java classes situated outside the `ui` folder are useful for the working of the application and debug examples accessible through the debug mode which can be triggered by clicking on the menu "Debug mode" of the `MainActivity.java`. These `.java` files include among others the `ComputerVision.java` and `Microbit.java` classes respectively used to identify the tangible tiles and set the Bluetooth connection between the Android device and the micro:bit.

Briefly, regarding the "Geography Game", the developer can customize the displacement costs of the robot and the MCQ questions related to each box. The displacement costs can be adjusted from line 171 of the `GeographyGameActivity.java` file and the MCQ questions are situated in the class `QuestionLibraryCaPL.java` (situated in the "capl" folder). This file contains each questions, choices and answers to the MCQs. The comment at the end of each line indicates which box of the geography world map is concerned.

2.2 capl-framework-app architecture

The capl-framework-app is a very simplified version of the capl-app. It basically contains only the `DeviceListActivity.java` and the debug version of the `FreeGameActivity.java` activities as it can be seen in figure 7. Once the developer launches the app, he directly lands on the `DeviceListActivity.java` activity. He can then either select the Bluetooth device he wants to connect to or he can click on the shortcut button leading to the `FreeGameActivity.java` activity without a Bluetooth connection. Both alternatives open directly the `FreeGameActivity.java` activity without passing through the `MenuActivity.java` activity as in the original capl-app.

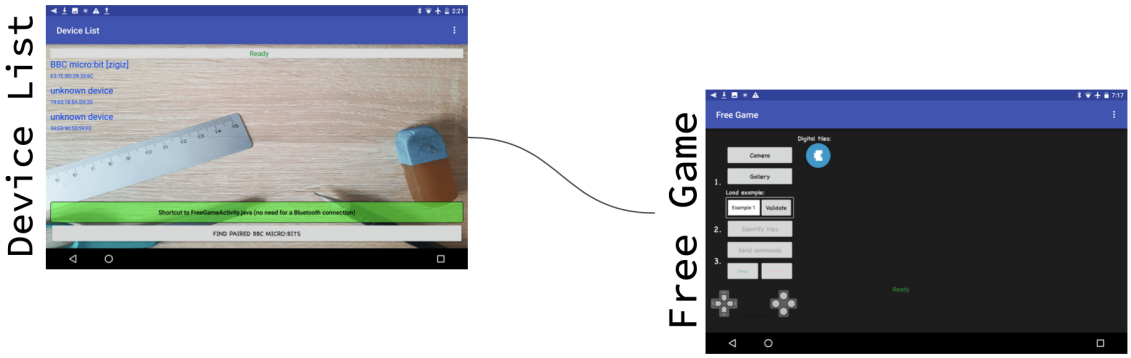


Figure 7: capl-framework-app workflow

In addition, all unnecessary classes, activities, menus, layouts and images have been removed on purpose to keep the architecture as simple as possible and with as little detail as possible. The only classes and activities needed for the capl-framework-app to work are visible in the figure 8.

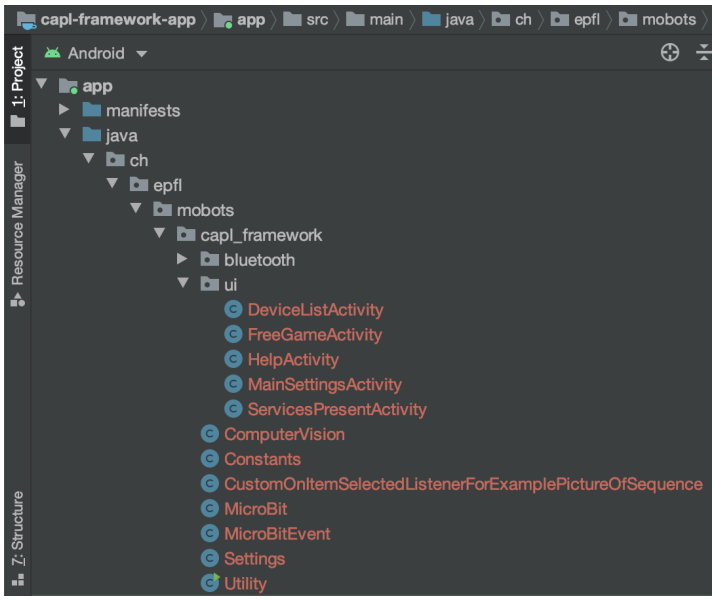


Figure 8: `capl_framework` folder of the capl-framework-app

This simplified version the capl-app provides a framework for testing the Bluetooth communication and the tiles identification algorithm. One can test the tiles recognition algorithm in the same way as in the original capl-app by either taking a picture, loading a picture from the gallery or selecting one of the five built-in examples. The simplest way to test the Bluetooth communication is to compose a simple program using the floating action buttons (e.g. by selecting only a "go_forward" command) and send it to the robot. Because no matter the BLE (Bluetooth Low Energy) device used by the robot, as long as it has Bluetooth enabled, the application can connect to it. One can even connect the Android device to another mobile phone or a sound system device. However, if the device cannot interpret the events sent from the tablet, this server device of course won't react to the Bluetooth signals. This is why it is important to program the robot so that it can receive and interpret Bluetooth events sent from the Android application. More details about robot coding and custom communication are given in the chapter 3.

3 Necessary adaptations to interface other robots

As briefly explained previously, the initial connection can be made with any device emitting Bluetooth. It is even possible to connect a phone to the tablet-based app. The only thing is that, in the original capl-app, the expected device (i.e. the brain of the robot) has to be a micro:bit or at least a device that can receive and interpret the Bluetooth events generated by the tablet. If the new robot is programmed in such a way that it is able to receive the Bluetooth event `mes_dpad_controller`, the present implementation of the capl-framework-app should work without any adaptation.

The following lines describe in detail the process of creating and emitting Bluetooth events. In the `FreeGameActivity.java` there are two functions that send commands to the micro:bit. The first one is `sendCommandsToMicroBit`. It is used to send a batch of commands to the robot. The second one is called `sendCommandsToMicroBitOneByOne` and, as the name suggests, it is used to send commands to the robot one after the other in a debug fashion by clicking the "Debug" button of the `FreeGameActivity.java`. Based on the preliminary work, that is the tiles identification, these two functions take care of sending Bluetooth events corresponding to the tiles to the robot. For the current identified tile, a specific numerical value, corresponding to the movement to be executed by the robot, is set to the Bluetooth event called `event_value`. This is done by means of a "switch statement" to differentiate between the different orders to send. For the `sendCommandsToMicroBit` function, this part of code of interest begins on line 1077 of the `FreeGameActivity.java`. For the `sendCommandsToMicroBitOneByOne` it begins on line 1692. Three lines of code are essential for generating Bluetooth events and sending them to the receiving device. Here is a detailed description of these three steps:

1. First, a micro:bit Bluetooth event is created. It is based on an event type (in our case it is `mes_dpad_controller`) and a numerical event value (e.g. the number "9" for the command "go_forward" as it can be seen on line 42 of the `Settings.java` class). This step uses the `MicroBitEvent` function of the class `MicroBitEvent.java` and looks as follows:

   ```
   mb_event = new MicroBitEvent(settings.getMes_dpad_controller(), event_value);
   ```

2. Then the micro:bit event is converted into a byte event. This is to be able to send digital information over Bluetooth. A byte is the smallest addressable unit of memory in many computer architectures. This step uses the `getEventBytesForBle` function of the `MicroBitEvent.java` class and is written as follows:

   ```
   event_byte = mb_event.getEventBytesForBle();
   ```

3. Finally, the Bluetooth event in byte format is sent to the micro:bit. This step uses the `writeCharacteristic` function of the `BleAdapterService.java` class. It takes as argument the server (i.e. the micro:bit) and the client (i.e. the tablet) and sends the byte event from the tablet to the micro:bit. The client and server are identified by their UUID that can be found e.g. on line 140 of the `BleAdapterService.java` class. A UUID is a universally unique identifier that is a 128-bit number used to identify information in computer systems. The line of code

corresponding to this step is written as follows:

```
bluetooth_le_adapter.writeCharacteristic(Utility.normaliseUUUID
(BleAdapterService.EVENTSERVICE_SERVICE_UUID), Utility.normaliseUUID(BleAdapter-
Service.CLIENTEVENT_CHARACTERISTIC_UUID), event_bytes);
```

In conclusion, the principle of Bluetooth communication is always the same and any robot integrating a BLE module (in the same vein as the micro:bit-based robot used in the original CaPL project) should be usable. In fact, nothing in the implementation of java classes handling Bluetooth communication seems to be exclusively related to micro:bit devices. Even if the `MicroBit.java` and `MicroBitEvent.java` classes might be adapted in certain cases to fit the specifications of another microcontroller. As a proof for this, communication can be initiated with any device with Bluetooth functionality (e.g. a phone, a sound system, etc.). The developer has however to make sure to code the robot so that it can recognize the received events from the tablet. Moreover, these events have to be linked to specific actions to perform and that is it! This code situated on the robot could be written in any language. In the original CaPL project, it is JavaScript (or equally Block-based code). Below is the JavaScript code that is triggered to make the micro:bit-based robot move forward a certain distance when the requested event with a specific value is received. This JavaScript code is equivalent to the block-based code presented in figure 3.

```
1  control.onEvent(EventBusSource.MES_DPAD_CONTROLLER_ID, ...
       EventBusValue.MES_DPAD_BUTTON_1_DOWN, function () {
2      if (index < 5) {
3          basic.showLeds(`
4              . . # . .
5              . # # # .
6              . . # . .
7              . . # . .
8              . . # . .
9              `)
10         pins.servoWritePin(AnalogPin.P1, 90 - LeftFwd)
11         pins.servoWritePin(AnalogPin.P2, 90 + RightFwd)
12         basic.pause(OneStepTime)
13         pins.servoWritePin(AnalogPin.P1, 90 + Stop)
14         pins.servoWritePin(AnalogPin.P2, 90 + Stop)
15     }
16 })
```

With Thymio the language used could be Python or Aseba and with other robots C or C++. As long as the events of the tablet and the numerical values they carry can be received by the BLE module of the robot, the robot can be asked to behave in any predefined way.

References

[1] Anthony Guinchard. *CaPL* Android application code. https://github.com/Antho1426/capl-app, 2020. CaPL project.

[2] Martin Woolley. *micro:bit Blue* Android application code. https://github.com/microbit-foundation/microbit-blue, 2017. [Consulted on the 17th of May 2020].