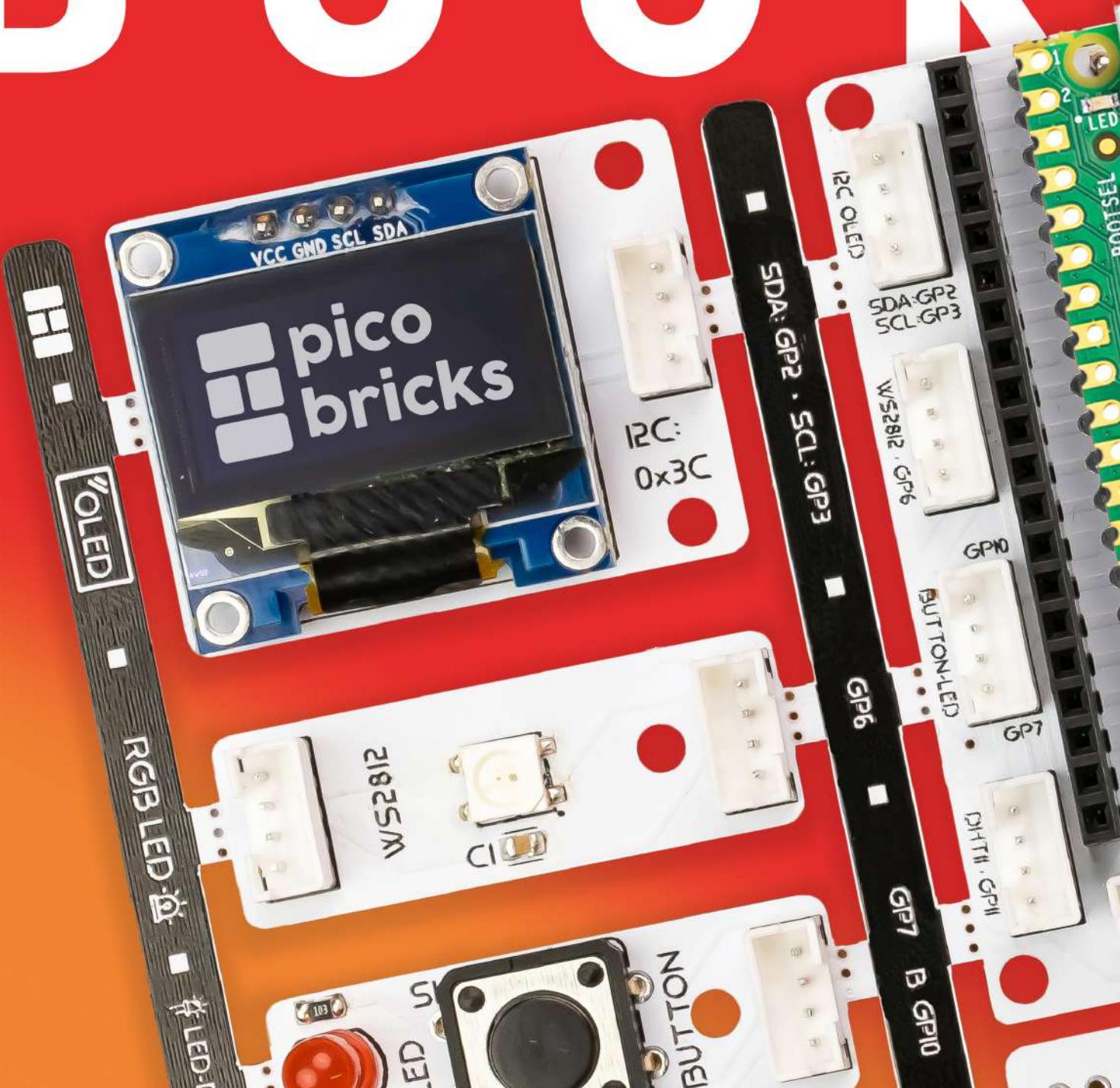


Project Book





Copyright © 2022 Robotistan

All rights reserved. It is strictly forbidden to copy, reproduce, use, publish and distribute the text, photographs and other content in this book, in whole or in part, without permission, except for individual use.

Contents: Mustafa Kemal Avcı, Abdullah Kaya

Translation: Naze Gizem Özer

Design: Ahmet Gürsu

Pico Bricks Developer Team

Yasir Çiçek - Project Manager

Yusuf Gündoğdu - Software Developer

Mehmet Suat Morkan - Chief Developer

Mehmet Ali Dağ - Hardware Developer



Powered by



John Maloney · Turgut Güneysu · Kathy Giori · Bernat Romagosa

Update: 12 Dec. 2022 Ver. 1.15



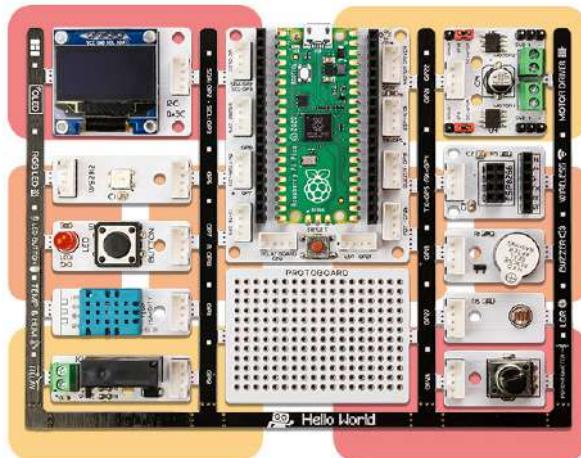
CONTENTS

What Is Pico Bricks?	4
1. Development Environments	6
1.1. MicroBlocks Block Programming Language	6
1.1.1. Interface Introduction	7
1.1.2. MicroBlocks-Picobricks Connection and Operation	9
1.2. Thonny (MicroPython) IDE for Beginners	13
1.2.1 Thonny IDE Setup	13
1.2.2. Thonny IDE Interface	13
1.2.3. Upload MicroPython Firmware to Raspberry Pi Pico	14
1.2.4. Installing and Running Code on Raspberry Pi Pico	15
1.3. Arduino IDE	17
1.3.1. Writing and Running Code with Arduino IDE	18
1.3.2. How to Add Arduino Library?	20
2. PROJECTS	22
2.1. Blink	23
2.2. Action - Reaction	27
2.3. Autonomous Lighting	31
2.4. Thermometer	38
2.5. Graphic Monitor	44
2.6. Dominate the Rhythm	49
2.7 Show Your Reaction	60
2.8. My Timer	68
2.9. Alarm Clock	78
2.10. Know Your Color	85
2.11. Magic Lamp	97
2.12. Smart Cooler	101
2.13. Buzz Wire Game	106
2.14. Dinosaur Game	115
2.15. Night and Day	120
2.16. Voice Controlled Robot Car	132
2.17. Two Axis Robot Arm	140
2.18. Smart House	154
2.19. Glutton Piggy Bank	161
2.20. Confirming Door	168
2.21. Automatic Trash Bin	182
2.22. Digital Ruler	188
2.23. Air Piano	196
2.24. Maze Solver Robot	205
2.25. Smart Greenhouse	213
3. Bibliography	233



What Is Pico Bricks?

Pico Bricks is an electronic development board + software which is designed for use in maker projects. With ten detachable modules included, Pico Bricks can be used to create a wide variety of projects. It also includes a protoboard that you can use to add your own modules!



Pico Bricks is for everyone interested in electronics and coding. Beginners with no prior experience will find it easy to get started thanks to the modular hardware design, Scratch-like block coding environment, and simulator. Those with experience can dig more deeply into electronics or explore coding in Python. And even the most expert makers will appreciate how quickly they can explore ideas and create prototypes with Pico Bricks.

Unlike other boards, Pico Bricks has an incredible amount of flexibility for every level of makers! Bricks IDE has example codes for different scenarios.

Learn coding from zero to hero with MicroBlocks or the Pico Bricks's drag-n-drop, block coding builder. MicroBlocks is the easiest coding experience ever created and widely known in the maker industry.



KIDS



STUDENTS



HOBBYISTS



EXPERT

Have a question? You can find more information [here](#)



POWERED BY
MicroBlocks

DEVELOPMENT ENVIRONMENT



1. Development Environments

In software, web and mobile application development, the development environment is a workspace with a set of processes and programming tools used to develop the source code for an application or software product. Development environments enable developers to create and innovate without breaking something in a live environment. You can code Picobricks using both text-based and block-based editors. MicroBlocks is a powerful editor with which you can code Picobricks with blocks.

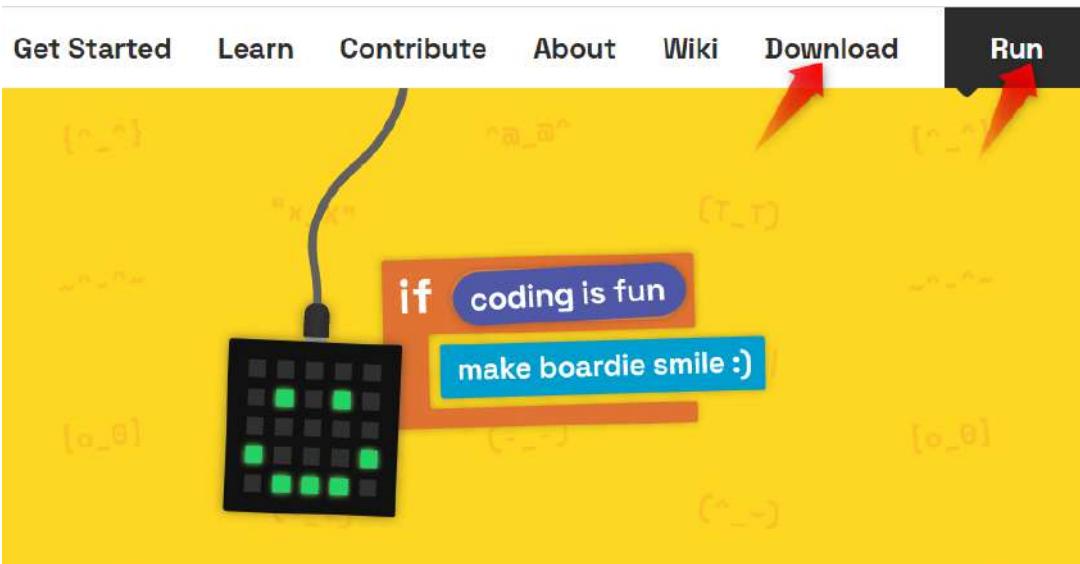
Thonny editor is a free, dedicated IDE(integrated development environment) for Python designed for beginners and one of the best choices for those who are just starting to learn MicroPython. Whether you just code MicroPython or code an electronic circuit board, Thonny provides you with a lot of support.

Arduino IDE is an open-source software, designed by Arduino.cc and mainly used for writing, compiling & uploading code to almost all Arduino Modules. It is an official Arduino software, making code compilation too easy that even a common person with no prior technical knowledge can get their feet wet with the learning process. If you are proficient in Arduino C language or want to learn C language, Picobricks and Arduino IDE will be very good choices for you.

1.1. MicroBlocks Block Programming Language

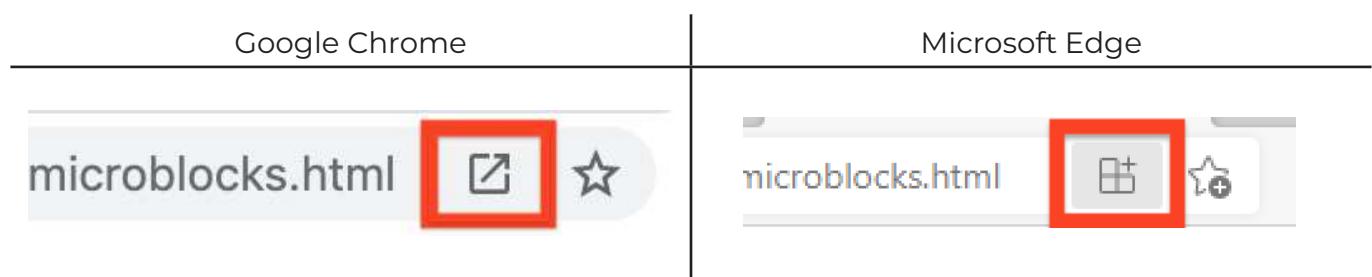
MicroBlocks is a free, Scratch-like blocks programming language for learning physical computing with educational microcontroller boards such as the micro:bit, Adafruit Circuit Playground Express, and many others. MicroBlocks is a live environment. Click on a block and it runs immediately, right on the board. Try out commands. See and graph sensor values in real time. No more waiting for code to compile and download. Want to display an animation while controlling a motor? No problem! MicroBlocks lets you write separate scripts for each task and run them at the same time. Your code is simpler to write and easier to understand. MicroBlocks runs on many different boards, but your scripts are portable. Buttons, sensors, and display blocks behave the same on all boards with the relevant hardware. Once you run the codes in MicroBlocks, you can disconnect the USB and feed the Picobricks with a different power source. The codes on the card will work automatically. You can use Thonny IDE and Arduino IDE to code Picobricks in text-based.

To code Picobricks with MicroBlocks, let's open <https://microblocks.fun/> in the browser (Google Chrome and Edge browsers are recommended).



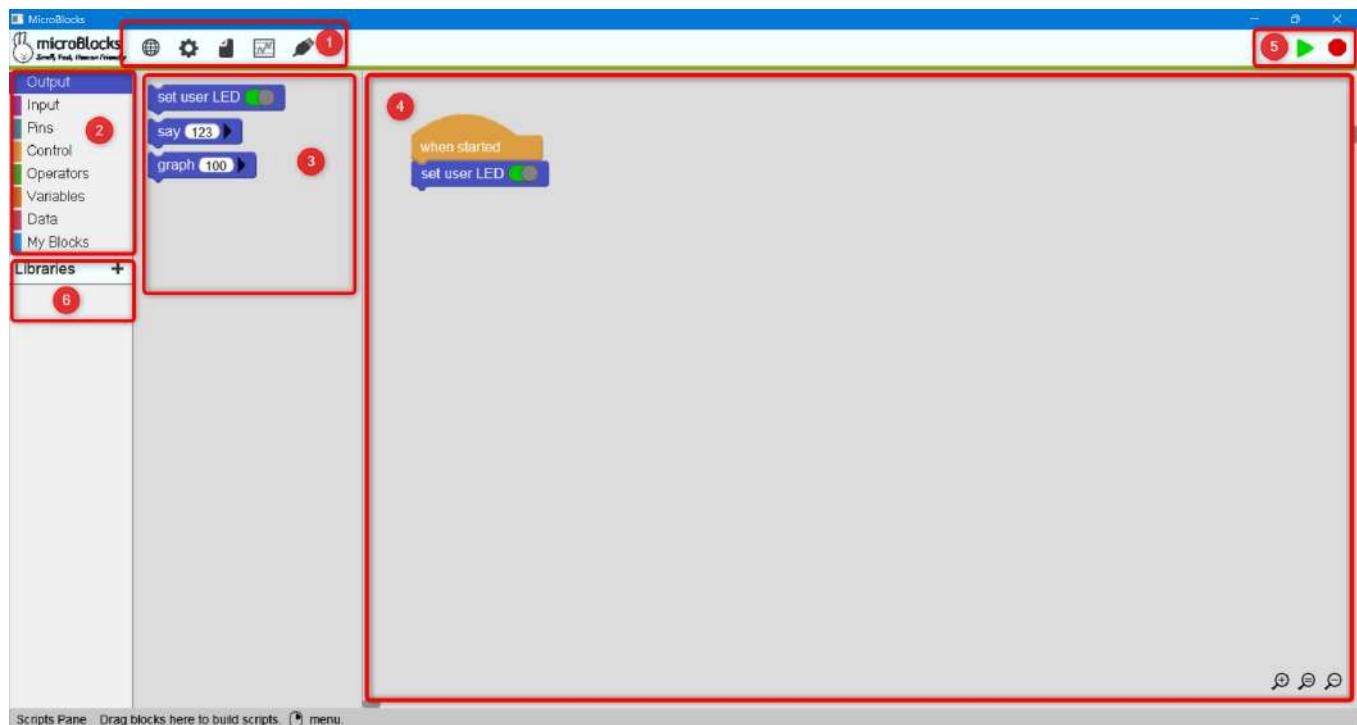
You don't need to install anything to run MicroBlocks in a Chrome or Edge browser; you can **run** the online editor by clicking the Run button in the menu at the top right of the screen. By clicking the **Download** button, you can download the version suitable for your operating system and install it on your computer.

You can save MicroBlocks Web editor in your browser and use it without internet access. Run MicroBlocks in your browser to register the MicroBlocks Web app, then click the **install** button in the upper-right corner of your browser's URL bar.



1.1.1. Interface Introduction

When you open the MicroBlocks program, an interface like the image will greet you. You can review the detailed explanation of the program interface below.



1. Menu Bar (➊ ➋ ⌂ ⌃ ⌄): In this section, the first button from left to right allows us to change the language option of the program. The second button is the menu where we can see the advanced MicroBlocks code options and the firmware update is done while the third button offers the save options. The fourth button opens a graph window used by the graph block to plot the data, while the fifth rightmost button is used to connect to Picobricks.

2. Block Categories: This field contains the categories of blocks used for programming in MicroBlocks. Categories are grouped using different colors for each category. As the categories are selected, the relevant blocks in that category will be listed in the Block 3 field in the Palette.

3. Block Palette: As selections are made in the Block categories field, blocks with specific functions will be listed in this field. Codes are written by dragging and dropping the blocks in this area to the Scripting area number 4.

4. Scripting Area: This is the area where all coding activities take place. User drag and drop blocks into this area to create scripts and custom blocks (functions).

5. Start/Stop Buttons (⏪ ⏹): This area contains two icons, Start and Stop, which are used to control the MicroBlocks programs.

6. Library List (Libraries +): In this area, there are libraries that are loaded depending on the requirements of user scripts and micro-devices.

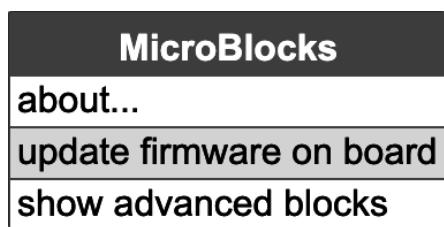


1.1.2. MicroBlocks-Picobricks Connection and Operation

Connect the board to your computer while holding down the white BOOTSEL button.



From the MicroBlocks menu (gear icon), select update firmware on board.

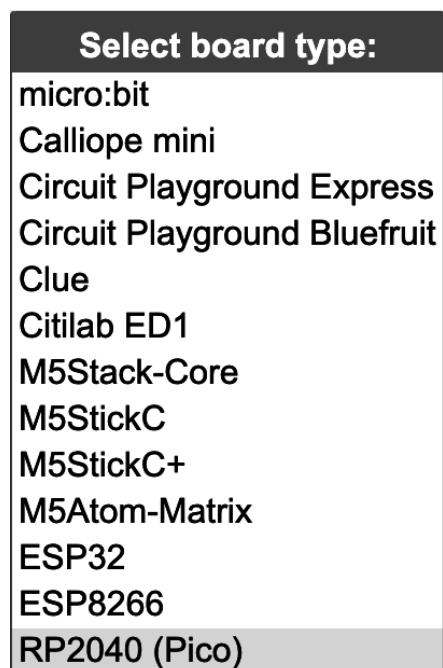


Firmware installation takes just a few seconds. If you are running the MicroBlocks app, MicroBlocks will connect to the board automatically when it is done.

Extra Steps in Browser

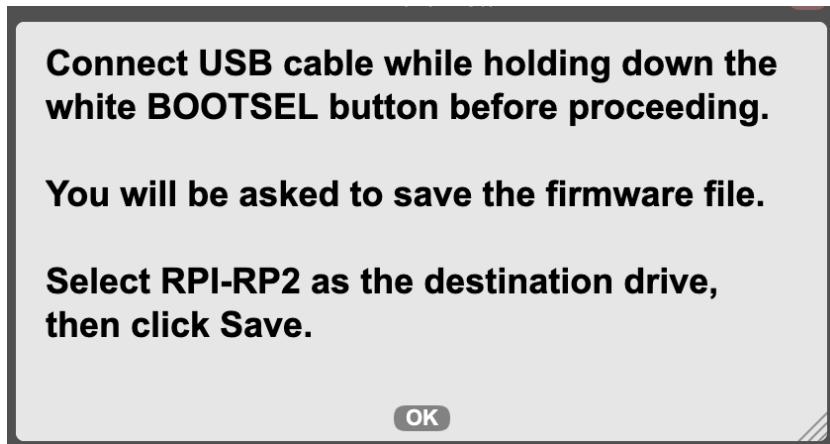
If you are running MicroBlocks in the browser or as a web app, you need to help the browser. For security reasons, the browser cannot access the board's USB drive without asking the user.

First, select your board type from the menu.





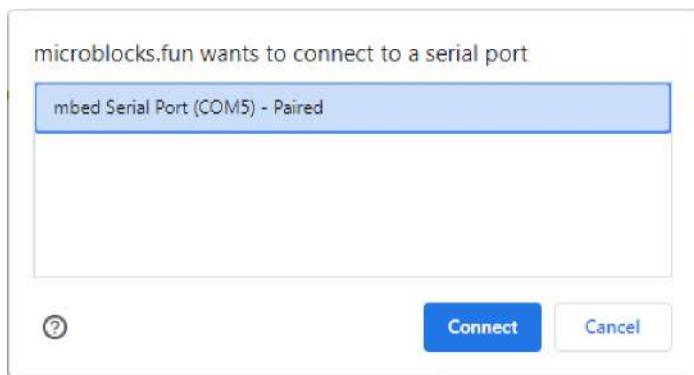
You'll be asked to select the USB drive for the board in the browser's file save dialog.



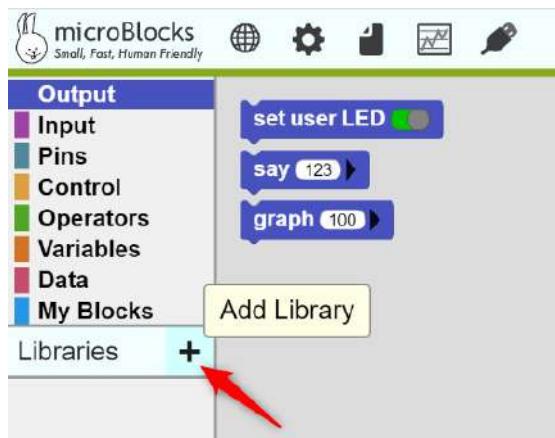
Follow the instructions to save the firmware file on your board. When the file is saved (just a few seconds), click to USB icon to connect to it.



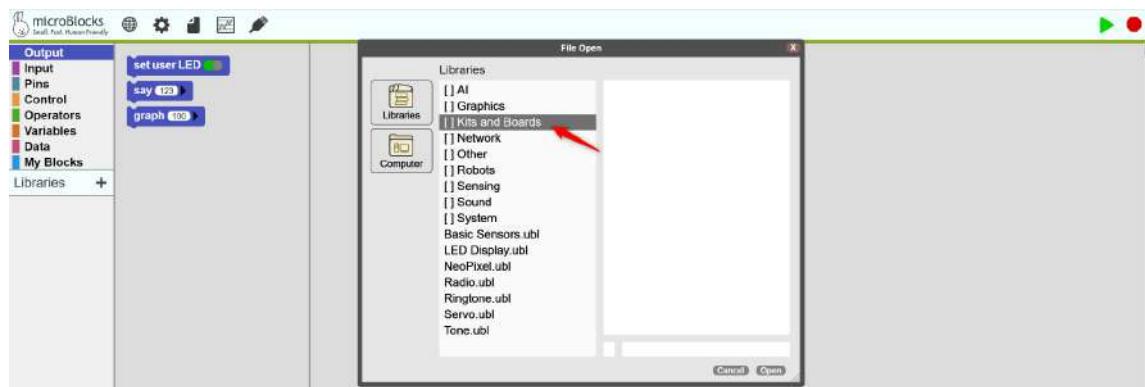
🔌 ➡️ Clicking the **Connect** button will display the system USB ports where the micro devices are plugged in. In this window, you can connect Picobricks to MicroBlocks by first selecting the Pico device and then clicking the Connect buttons. When the connection is successful, a green circle will appear behind the USB icon.



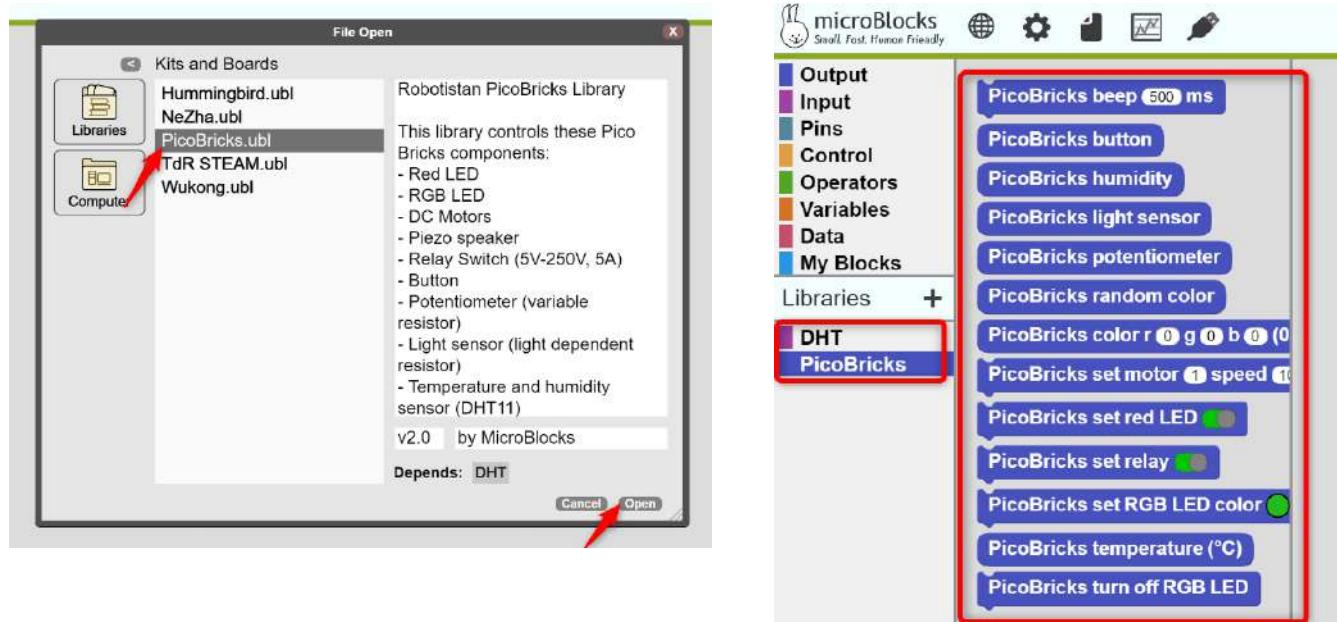
MicroBlocks is a real-time coding editor. There is no process of compiling and uploading the codes to the card after they are written. When you click on the code blocks, the codes will run. First, you need to import Picosbricks' library into the Microblocks editor. You have to click the **Add Library** button for this.



In the **File Open** window that opens, click the Kits and Boards button to open the list of devices that you can code with Microblocks.



Click **PicoBricks.ubl** from the drop-down list, and then click the **Open** button.



If all went well, the PicoBricks library and code blocks will be displayed in the Code blocks panel.



Now let's run our first code. First, drag and drop the when started block in the control menu to the code writing area. Then drag the Picobricks set red LED block from the Picobricks category and add it below the when started block. When you press the start button, you will see the red led on the Picobricks light up.

 After editing your codes in MicroBlocks, when you click the Start button, your codes will be installed into Picobricks and run.

 The Stop button stops the codes from running. But the codes uploaded to Picobricks are not deleted. You can disconnect USB, run Picobricks with external power supply.

If you have previously uploaded the necessary firmware file to encode Picobricks with MicroBlocks to Pico, you can connect by clicking the USB icon. If you are going to connect MicroBlocks Picobricks for the first time, you can follow the steps in heading 1.1.2.

For detailed information on using the Microblocks editor, visit:

<https://wiki.microblocks.fun/ide>



1.2. Thonny (MicroPython) IDE for Beginners



[Download version 3.3.13 for Windows](#) • [Mac](#) • [Linux](#)

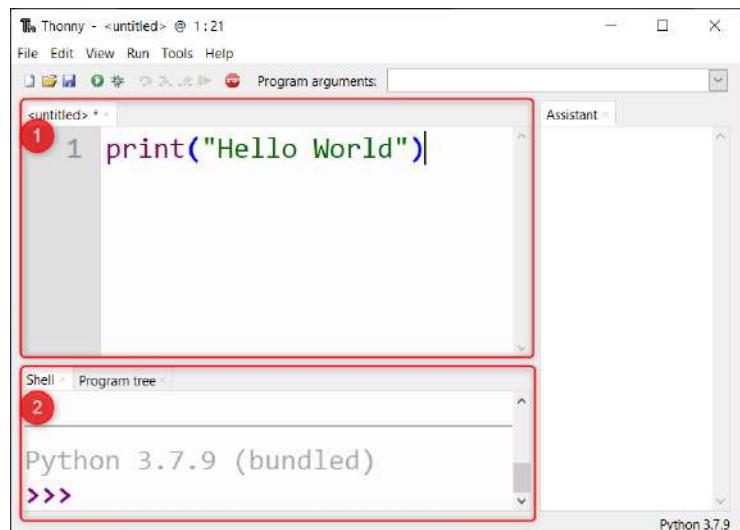
For the curious: [4.0.obj](#)

At the heart of Picobricks is the Raspberry Pi Pico. The Thonny Raspberry Pi is a great choice for coding Pico and therefore Picobricks.

1.2.1 Thonny IDE Setup

Visit <https://thonny.org/> Select the version suitable for your system and download it to your computer. Then perform the installation. You can also install the Thonny IDE using the command “ \$ pip install thonny ”

1.2.2. Thonny IDE Interface



When you start Thonny, you will see a window like the one below. We will write our codes in part 1. In part 2, we will see the outputs of our codes.



A: Opens an empty script file.

B: Allows you to open an existing code file.

C: Allows you to save changes to the code file you are working on.

D: Runs the code you wrote in the interpreter environment you specify.

E: Allows you to check for errors in your code.

F: Allows you to run lines of code in order to debug.



G: Lets you navigate through the commands in the line of code while debugging.

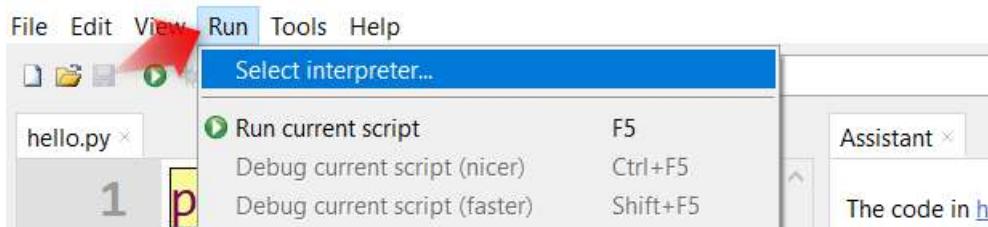
H: Lets you exit debug.

I: Allows you to switch from debug mode to run mode.

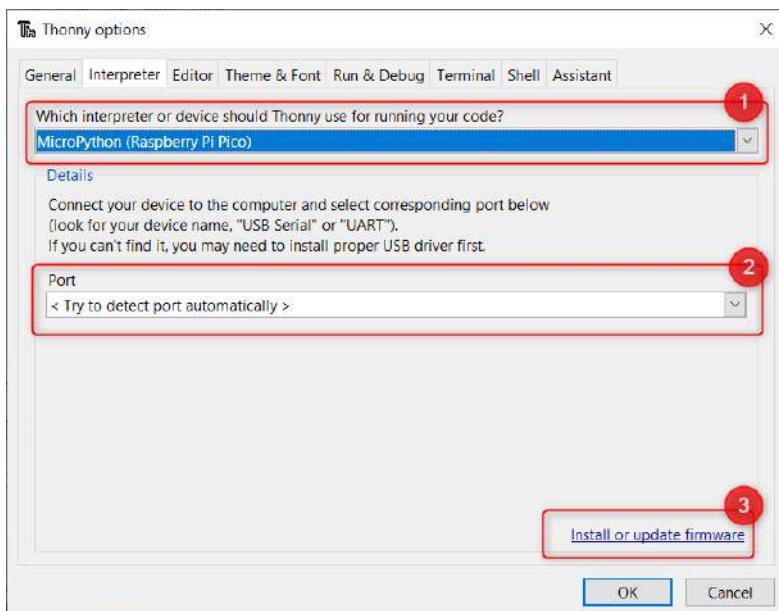
J: Makes the code stop executing.

1.2.3. Upload MicroPython Firmware to Raspberry Pi Pico

In order for Raspberry Pi Pico to understand the MicroPython codes we will write, we must install a special operating system for it. We call this firmware. Open the Thonny editor and click **Select interpreter** from the **Run** menu.



Select the **Raspberry Pi Pico** from the drop-down list shown in area 1. Leave the 2nd area as in the image, click on the 3rd area.

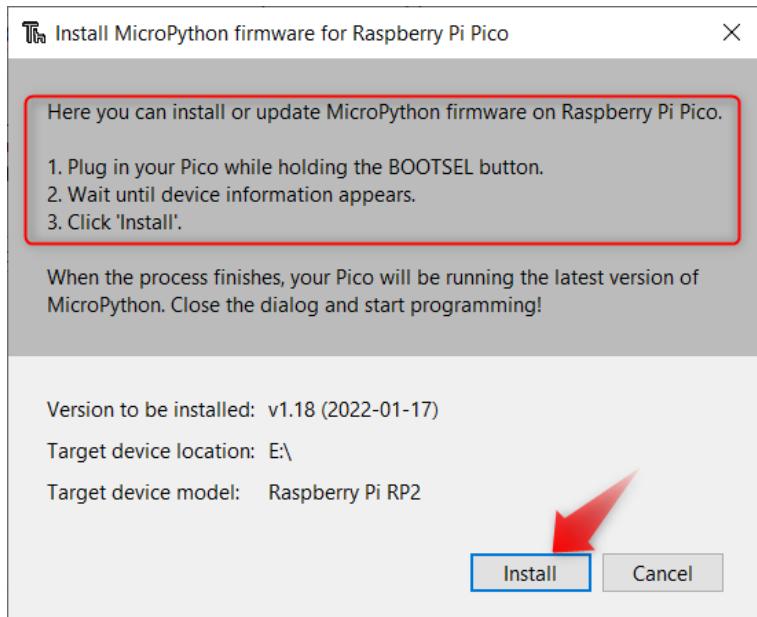


Connect Pico to your computer's USB port with a cable while holding down the white **bootsel** button on it.





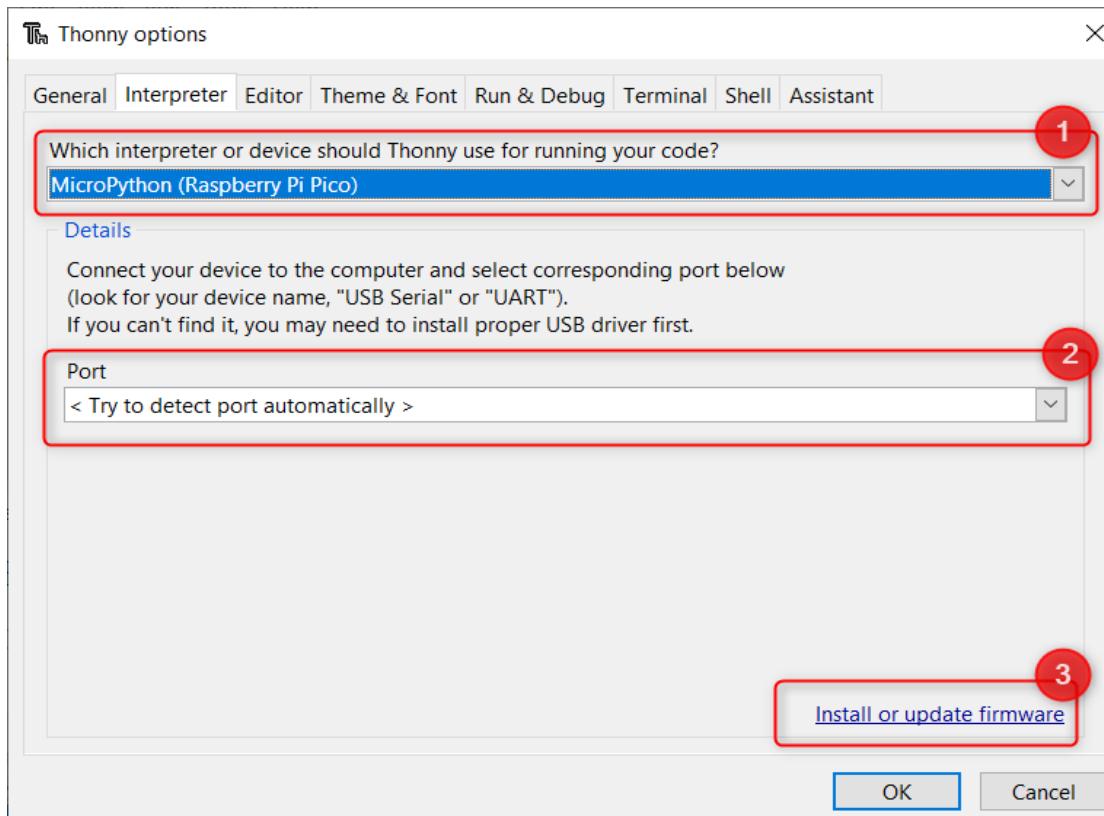
After the Install button is activated, you can release the button. Press the Install button and wait for the firmware to load.

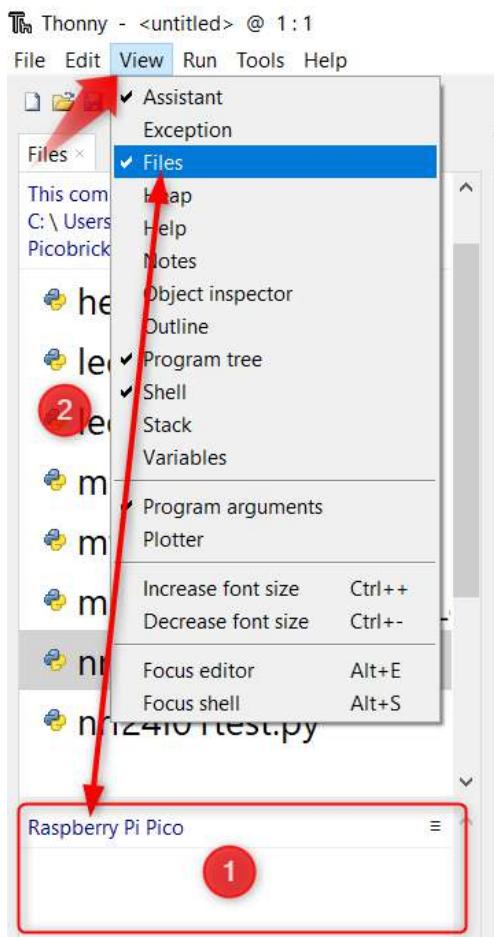


After the installation is complete, click the Close button to complete the installation.

1.2.4. Installing and Running Code on Raspberry Pi Pico

Plug the Pico's cable directly into the computer's USB port. You **don't need** to hold down the Bootsel button. Select the **Select interpreter** option from the **Run** menu in Thonny. Make sure Raspberry Pi Pico is selected in section 1. Click the **OK** button to close the window.

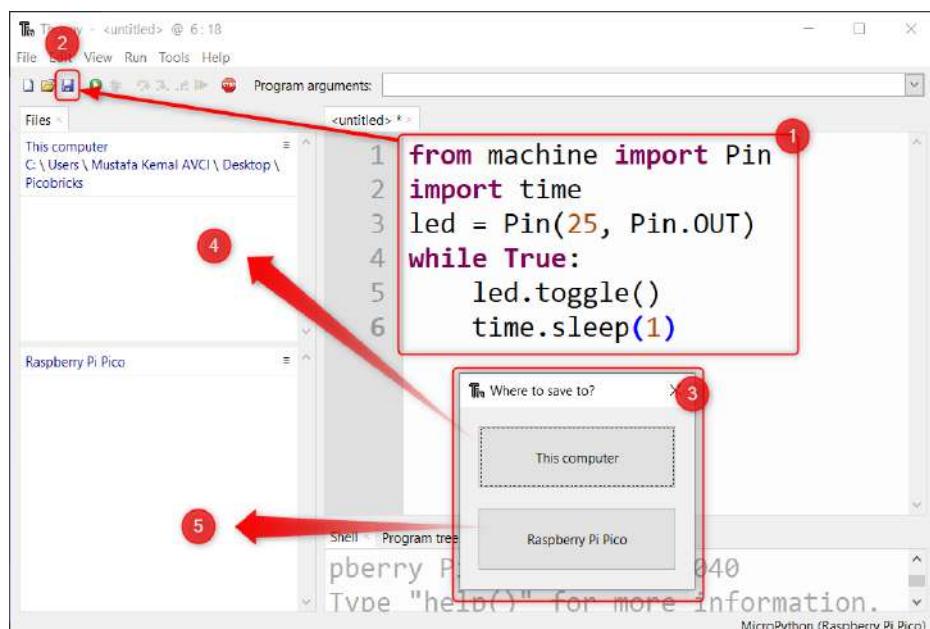




Activate the **Files** option from the **View** menu. A long file explorer tab will be placed on the left side of the screen. If you see **Raspberry Pi Pico** in **section 1**, it means that it is connected to Thonny Pico without any problems, you are ready to write, save and run your code. The part number 2 behind the menu is the file explorer area that shows the working directory **on your computer**.

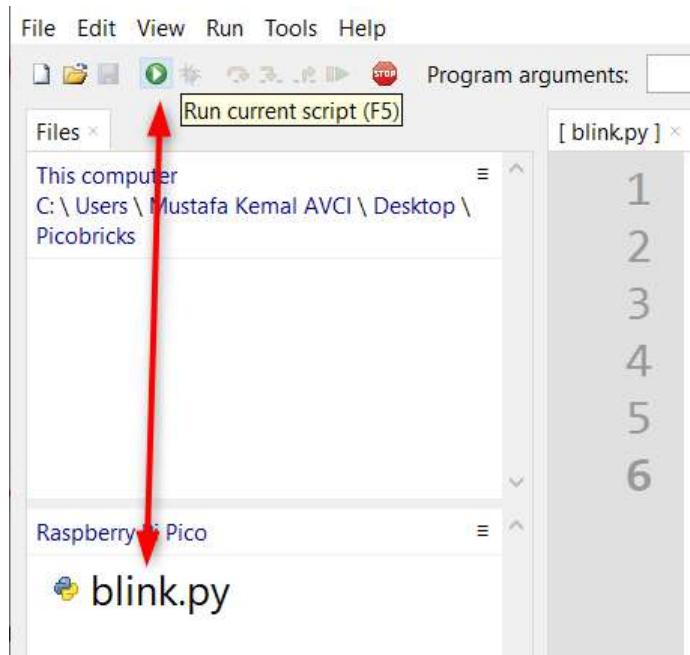
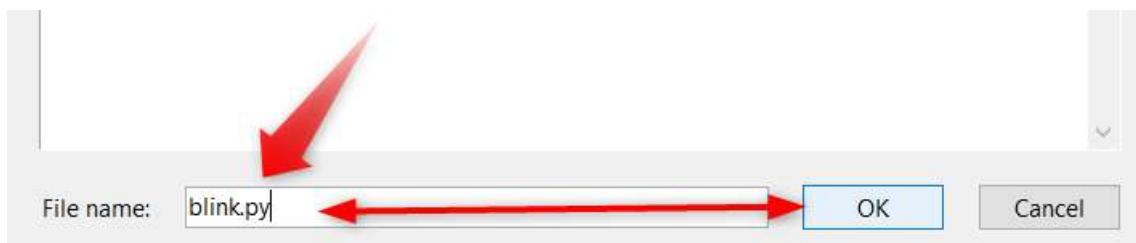
The MicroPython codes you wrote in Thonny consist of libraries arranged for Raspberry Pi Pico and similar micro control cards and are called MicroPython. The syntax and almost all libraries work the same as MicroPython.

The “**hello world**” application of the software world is the “**blink**” application to physical programming. Write down the code shown in field 1. Click the save button in area 2. Thonny will ask you in the window in area 3 whether you want to save your code in the working directory on your computer or in Pico’s onboard memory. If you choose your computer, the resulting file will appear in field 4, and if you choose Pico, the resulting file will appear in field 5.





Select Raspberry Pi Pico from the Save in window, type “blink.py” in the **File Name** field and click the OK button.



After seeing the “blink.py” file in Pico’s file explorer, click the F5 key on the keyboard or the green Run button on the toolbar, and the code file will be run by Pico. If you see the internal LED on the Pico blinking at 1 second intervals, you have successfully written and run your first code. Congratulations :)

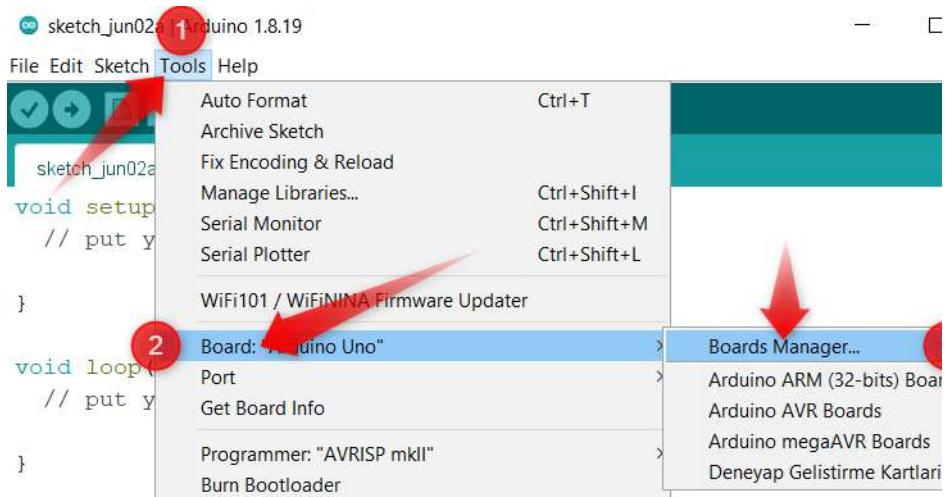
An important note: If you want the code you have written to run as soon as Pico is opened without giving a run command, you should save your code in Pico’s main directory with the name “main.py”.

1.3. Arduino IDE

Picobricks offers us the opportunity to code with Arduino C. Getting started coding the Raspberry Pi Pico at the heart of Picobricks with the widely used Arduino IDE is pretty easy.

Download the Arduino IDE 1.8.x setup file from <https://www.arduino.cc/en/software> to your computer and install it.

First you need to add Raspberry Pi Pico to Arduino IDE. Start the Arduino IDE. Then go to **Tools>Board>Boards Manager**.



Write “Raspberry Pi Pico” in field 1. After waiting for a while, click on the **Arduino Mbed OS RP2040 Boards** option and click the **install** button in field 2



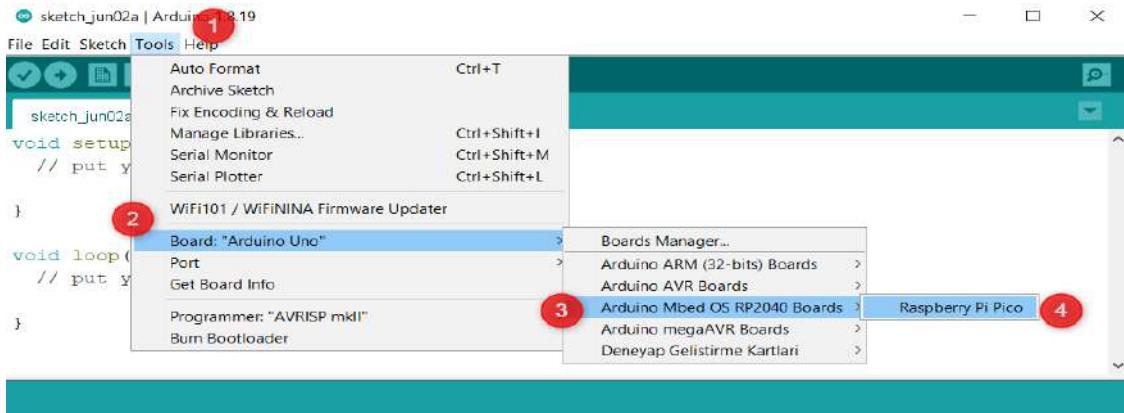
During all these installations, you must accept the approvals it will ask you for. When the installation is complete and click the close button, you will have added Pico to the Arduino IDE.

1.3.1. Writing and Running Code with Arduino IDE

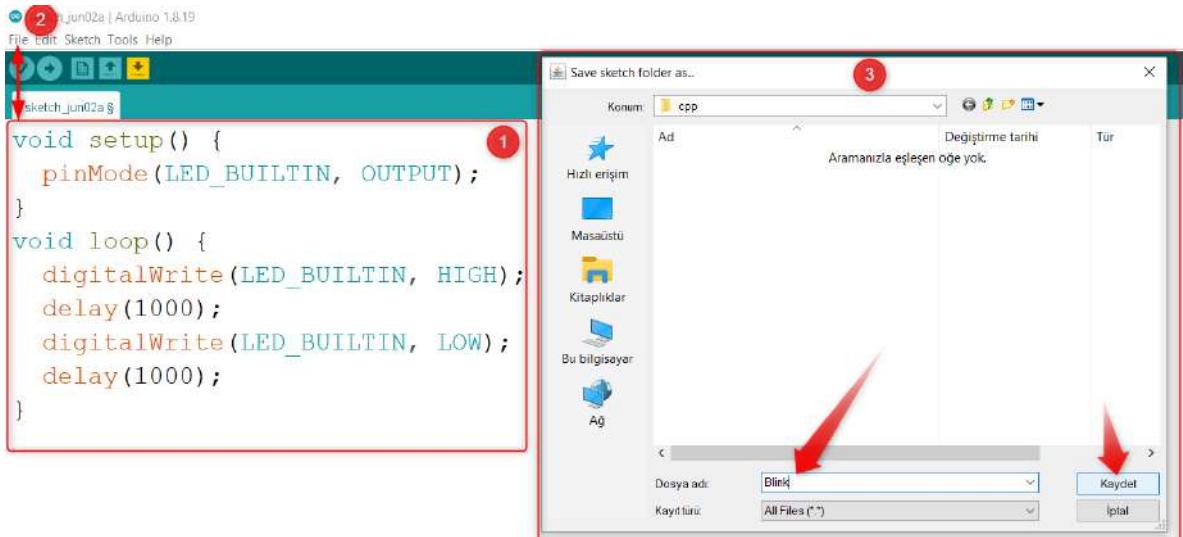
When you want to code Pico with Arduino IDE, you just have to connect it to your computer by holding the BOOTSEL button for the first time.



In this way, Pico will be connected in bootloader mode and recognized by your computer as external memory. Connect Pico to your computer by holding down the Bootsel button. After seeing Pico as the computer's flash memory, activate your card by going to **Tools>Board>Arduino Mbed OS RP2040 boards> Raspberry Pi Pico**.



Write the code in the field number 1 below and follow the **File>Save** path and save it anywhere on your computer with the name “**Blink**”.



```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
}

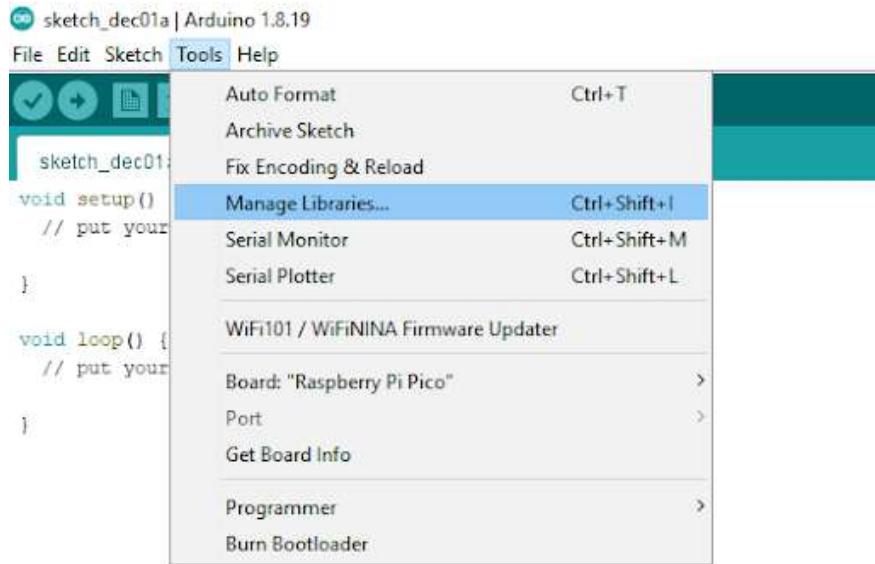
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000);
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000);
}
```

After the saving process, we must click the “Upload” button in the 1st field to compile the code and save it in Pico. When we see Done uploading at the bottom, our code will run in Pico and the built-in LED will blink at 1-second intervals. Important Note: While coding Picobricks with Arduino IDE, connect it to your computer by pressing the BOOTSEL button at the first pass from Micropython or Microblocks firmware. You do not need to press BOOTSEL for subsequent code uploads. Enjoyable projects :)



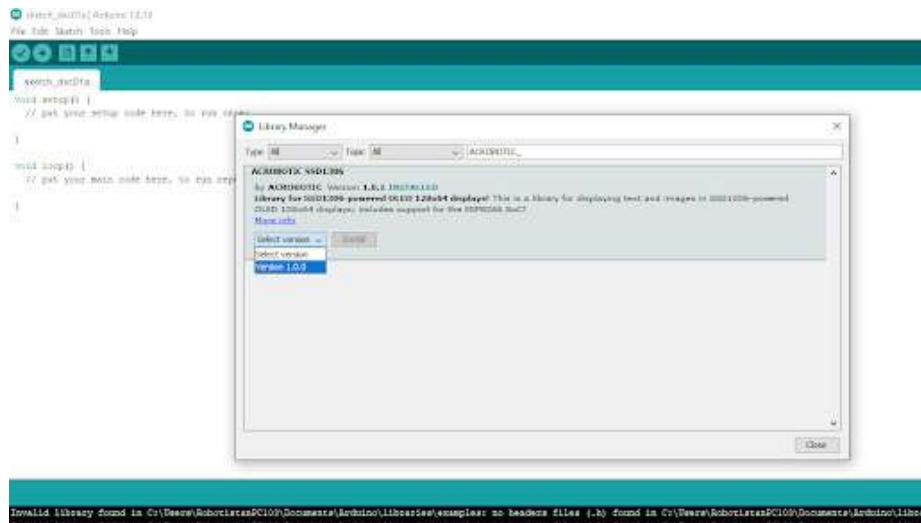
1.3.2. How to Add Arduino Library?

To install a new library into your Arduino IDE you can use the Library Manager. Open the IDE and click to the “Tools” menu and then Tools > Manage Libraries.



Then the Library Manager will open and you will find a list of libraries that are already installed or ready for installation.

Search for the library you want to install by typing its name, then select the version of the library. Finally, click the “install” button and wait for it to install.



Installing the library depends on your connection speed. When the installation is complete, you will start to see “INSTALLED” next to the library. In this way, you can easily install the libraries you need according to the codes you have written or the project you have made.

We define our library as shown below.

 sketch_dec01a | Arduino 1.8.19

File Edit Sketch Tools Help



sketch_dec01a §

```
#include <Wire.h>
#include "ACROBOTIC_SSD1306.h"
|
#define TRIGGER_PIN 15
#define ECHO_PIN     14
#define MAX_DISTANCE 400
```

PROJECTS



2.1. Blink

In real life, the employee, who has just started to learn the job, first undertakes the most basic task. The cleaner first learns to use the broom, the cook learns to use the kitchen utensils, the waiter to carry a tray. We can increase these examples. The first code written by newcomers to software development is known as "Hello World". Printing "Hello World" as soon as the program starts on the screen or console window in the language they use is the first step in programming. Like a baby starting to crawl... The first step to robotic coding, also known as physical programming, is the Blink application. It means winking at robotic coding. By simply connecting an LED to the circuit board, the coding is made to keep the LED blinking continuously. Ask people who have developed themselves in the field of robotic coding how they got to this level. The answer they will give you starts like this; it all started with a flashing LED!

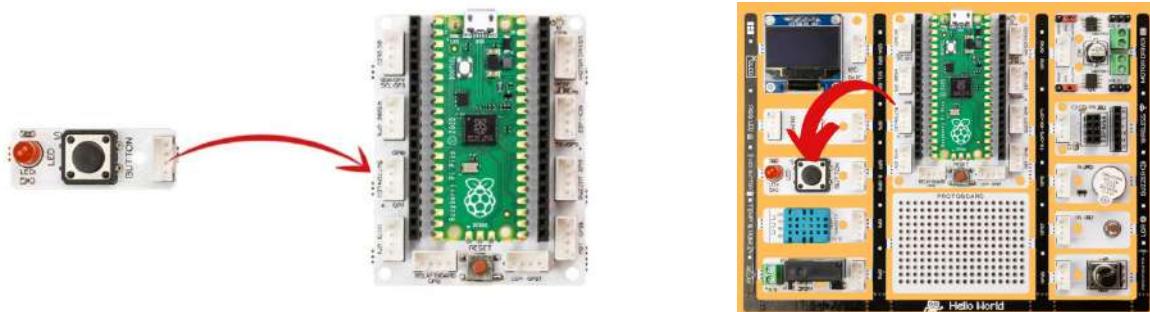
LEDs are the language of electronic devices. Thanks to the LEDs, the programmer tells the users at which stage of the task the device is, what the problem is, if any, and which options are active. In this project, you will learn the types of LEDs on it with Picosbricks and learn how to flash them.

2.1.1. Project Details and Algorithm

There are 1 x 5mm red LED and 1 x WS2812B RGB LED on Picosbricks. While normal LEDs can light up in one color, RGB colors can light up in different colors, both primary and secondary colors. In this project we will use the red LED on Picosbricks.

In the project, we will write the necessary codes to turn on the red LED on Picosbricks, turn it off after a certain time, turn it on again after a certain time, and repeat these processes continuously.

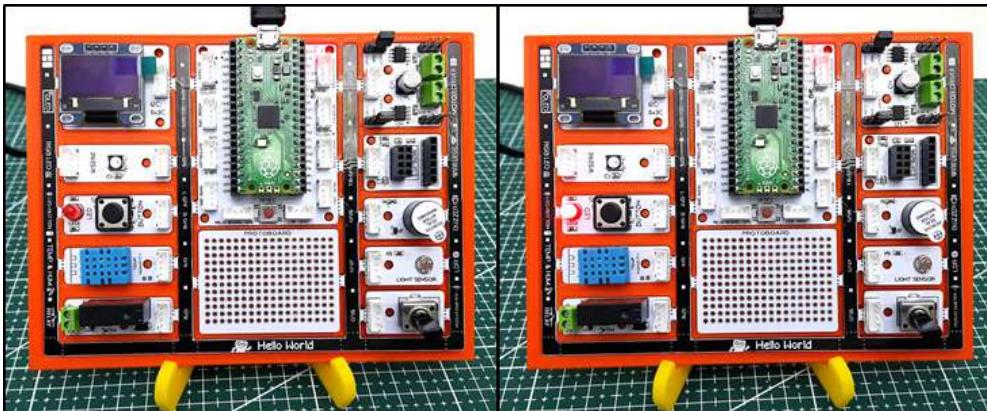
2.1.2. Wiring Diagram



You can code and run Picosbricks' modules without wiring. If you are going to use the modules by separating them from the board, you should make the module connections with grove cables.



2.1.3. Project Image



2.1.4. Project Proposal

Can we light the LED with different time intervals? For example; flashing of the LED several times per second, several times every half second.

2.1.5. Coding the Project with MicroBlocks

If you have done MicroBlocks-Picobricks connection and library installation, the steps you need to follow for the first project are detailed in the table below.

1	Drag and drop the when started block to the code writing area in the Control menu so that the code you wrote when Picobricks starts to run first.	
2	Then, drag the forever block from the Control menu and add it under the when started block so that the codes you write will run continuously as long as Picobricks is running.	
3	Drag the PicoBricks set red LED block among the code blocks in the Picobricks library and drop it into the forever block for the red LED to light up. Test if the red LED is lit by pressing the start button.	



4	Now, to turn off the red LED, click once on the checkbox in the Picobricks set red LED block to set the checkbox to red, that is, off, and test whether the LED goes out by pressing the Start button again.	<pre> when started forever Picobricks set red LED [checkbox off] end </pre>
5	After flashing the red LED with the code block, we will write the necessary codes for the LED to flash itself at certain time intervals. Drag the wait 500 millisecs block from the Control category and add it below the Picobricks set red LED block.	<pre> when started forever Picobricks set red LED [checkbox off] wait 500 millisecs end </pre>
6	Now add the Picobricks set red LED block again under the wait 500 millisecs block and turn the checkbox off. Then add the wait 500 millisecs block to the bottom again. When you press the start button, you will see that the red led on the Picobricks blinks at 500 millisecond intervals. The number 500 in the wait 500 millisecs block represents milliseconds. You can change this number as you wish. When it reaches 1000, the red LED will flash in 1000 milliseconds, i.e. 1 second intervals.	<pre> when started forever Picobricks set red LED [checkbox off] wait 500 millisecs Picobricks set red LED [checkbox off] wait 500 millisecs end </pre>

[Click](#) to access the project's MicroBlocks codes.

2.1.6. MicroPython Codes of the Project

```

from machine import Pin #to access the hardware
on the pico
import utime #time library

led = Pin(7, Pin.OUT) #initialize digital pin 7 as an output for LED

while True: #while loop

    led.toggle() #LED on&off status
    utime.sleep(0.5) #wait for a half second

```



2.1.7. Arduino C Codes of the Project

```
void setup() {  
    // put your setup code here, to run once:  
    pinMode(7, OUTPUT); // initialize digital pin 7 as an output  
}  
  
void loop() {  
    // put your main code here, to run repeatedly:  
    digitalWrite(7, HIGH); //turn the LED on by making the voltage HIGH  
    delay(500); //wait for a half second  
    digitalWrite(7, LOW); //turn the LED on by making the voltage LOW  
    delay(500); //wait for a half second  
}
```

GitHub Blink Project Page



<http://rbt.ist/link>



2.2. Action - Reaction

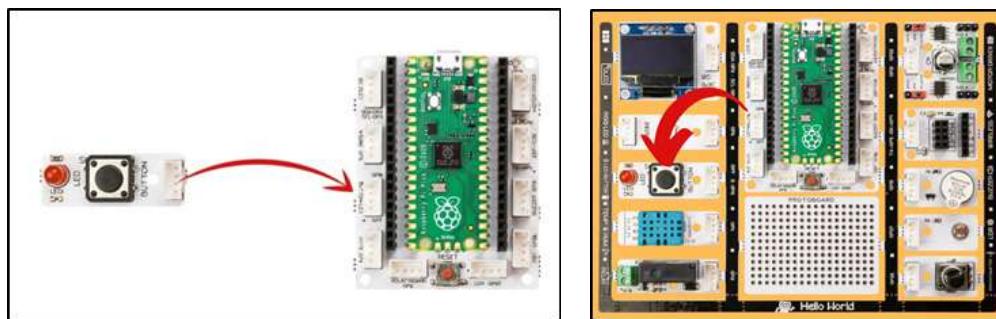
As Newton explained in his laws of motion, a reaction occurs against every action. Electronic systems receive commands from users and perform their tasks. Usually a keypad, touch screen or a button is used for this job. Electronic devices respond verbally, in writing or visually to inform the user that their task is over and what is going on during the task. In addition to informing the user of these reactions, it can help to understand where the fault may be in a possible malfunction. In this project, you will learn how to receive and react to a command from the user in your projects by coding the button-LED module of Picobricks..

2.2.1. Project Details and Algorithm

Different types of buttons are used in electronic systems. Locked buttons, push buttons, switched buttons... There is 1 push button on Picobricks. They work like a switch, they conduct current when pressed and do not conduct current when released. In the project, we will understand the pressing status by checking whether the button conducts current or not. If it is pressed, it will light the LED, if it is not pressed, we will turn off the LED.

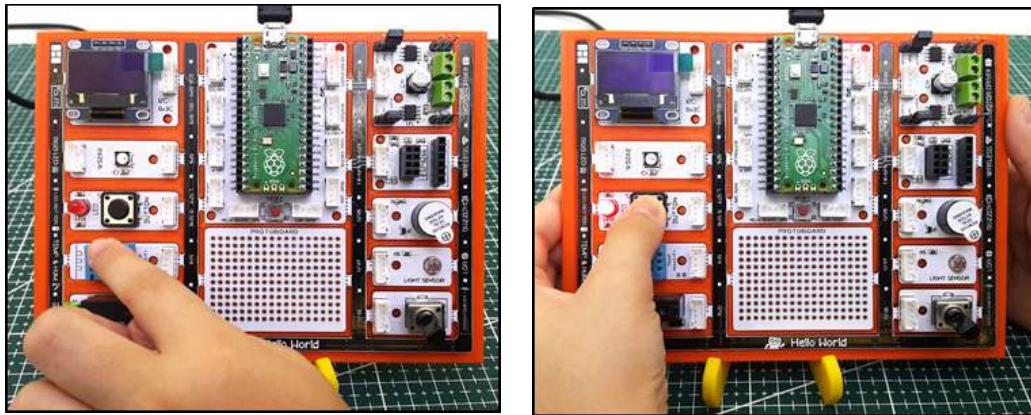
2.2.2. Wiring Diagram

You can code and run Picobricks' modules without wiring. If you are going to use the modules by separating them from the board, you should make the module connections with grove cables.





2.2.3. Project Image



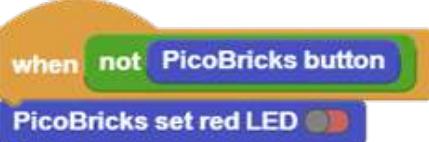
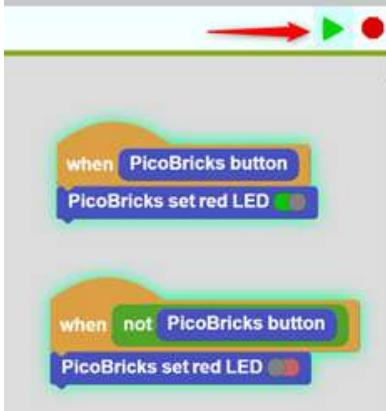
2.2.4. Project Proposal

In this project, the LED turns on when the button is pressed, and the LED turns off when the button is released. You can write the necessary codes for the LED to turn on when the button is pressed once and to turn the LED off when it is pressed again.

2.2.5. Coding the Project with MicroBlocks

1	Let's drag the when block in the Control category because it is a command that we need to run in case of button press. This block constantly checks the condition we have set, and if the condition is true, it runs the command below.	
2	Place the PicoBricks button block in the Picobricks category in the when block, as our condition is that the Picobricks button is pressed.	
3	We want the red LED in Picobricks to light up when the condition is met. So place the PicoBricks set red LED block in the Picobricks category below the when block.	



4	<p>We want the red LED to stay off when the button is not pressed. So drag the second when block from the Control category. In the Condition field, place the block in the operators category to create the expression when the button is not pressed.</p>	
5	<p>Place the PicoBricks button block from the Picobricks category in the not block to create the expression when the Picobricks button is not pressed.</p>	
6	<p>We want the red LED to remain off whenever the button is not pressed. So place the PicoBricks set red LED block from the Picobricks category under the when block and turn the switch off.</p>	
7	<p>When you press the Start button of MicroBlocks, the codes will run in real time. When you press the button on Picobricks, the red LED will turn on, and when you release it, it will turn off.</p>	

[Click](#) to access the project's MicroBlocks codes.

2.2.6. MicroPython Codes of the Project

```
from machine import Pin #to acces the hardware picobricks
led = Pin(7, Pin.OUT) #initialize digital pin as an output for led
push_button = Pin(10, Pin.IN,Pin.PULL_DOWN) #initialize digital
pin 10 as an input
```

```
while True: #while loop
```

```
    logic_state = push_button.value() #button on&off status
```



```
if logic_state == True: #check the button and if it is on
    led.value(1) #turn on the led
else:
    led.value(0) #turn off the led
```

2.2.7. Arduino C Codes of the Project

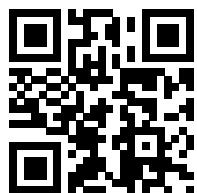
```
void setup() {
// put your setup code here, to run once:
pinMode(7, OUTPUT); //initialize digital pin 7 as an output
pinMode(10,INPUT); //initialize digital pin 10 as an input

}

void loop() {
// put your main code here, to run repeatedly:
if (digitalRead(10) == 1){ //check the button and if it is on

    digitalWrite(7, HIGH); //turn the LED on by making the voltage HIGH
}
else{
    digitalWrite(7, LOW); //turn the LED on by making the voltage LOW
}
delay(10); //wait for half second
}
```

GitHub Action - Reaction Project Page



<http://rbt.ist/actionreaction>



2.3. Autonomous Lighting

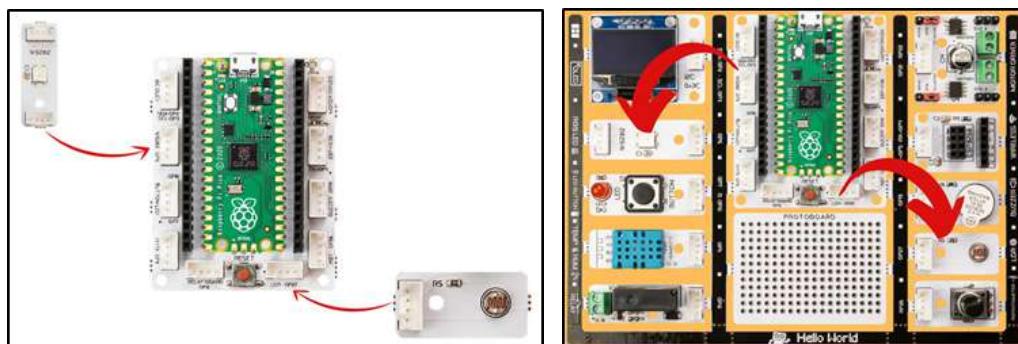
It is called the state of being autonomous when electronic systems make a decision based on the data they collect and perform the given task automatically. The components that enable electronic systems to collect data from their environment are called sensors. Many data such as the level of light in the environment, how many degrees the air temperature is, how many lt/min water flow rate, how loud the sound is, are collected by the sensors and transmitted to PicoBricks as electrical signals, that is data. There are many sensors in Picobricks. Knowing how to get data from sensors and how to interpret and use that data will improve project ideas like reading a book improves vocabulary. In this project, with PicoBricks, we will enable the LED to turn on when the amount of light decreases in order to understand the working systems of the systems where the lighting is turned on automatically when it gets dark.

2.3.1. Project Details and Algorithm

Sensors are electronic components that detect data in external environments and send data to microcontrollers. The LDR sensor also detects the amount of light in the environment and sends analog values. In our project, we will first check the incoming data when the environment is light and dark by reading the LDR sensor values, then we will set a limit according to these data, and if the amount of light is below this limit, we will turn off the RGB LED of Picobricks, if not, we will turn off the LED.

2.3.2. Wiring Diagram

You can code and run Picobricks' modules without wiring. If you are going to use the modules by separating them from the board, you should make the module connections with grove cables.





2.3.3. Project Image



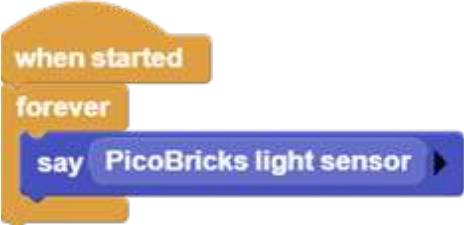
2.3.4. Project Proposal

In this project, we turned on the LED on the LDR sensor data on Picobricks if the environment was dark, and turned off the LED if it was bright. By processing the LDR sensor data, you can code a nightlight or table lamp in your home to turn on automatically in the dark. You can use the relay on Picobricks for this.

2.3.5. Coding the Project with MicroBlocks

1	Firstly, drag the when started block from the Control menu and drop it to the code writing area so that the code you wrote when Picobricks starts to run.	
2	Then, drag the forever block from the Control menu and add it under the when started block so that the code you wrote will run continuously as long as Picobricks is running.	
3	In order to provide autonomous lighting, we first need to see the values coming from the LDR sensor when the environment is bright and dark, and we need to act according to these values. For this, drag and drop the say123 block in the Output category into the forever block.	

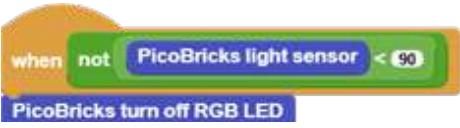
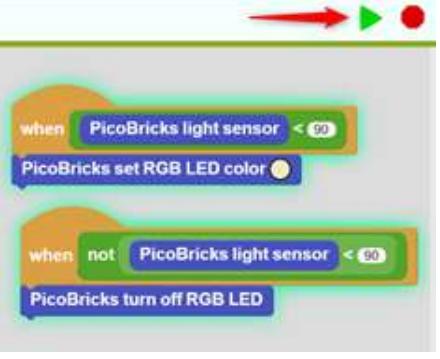


<p>4 Then drag and drop the PicoBricks light sensor block in the Picobricks category to the circle that 123 in the say block. See the values from the sensor by pressing the start button. The Light Sensor block gives the ambient light level as a percentage (%). You should see a value of 100 in full light, 90 and below when you close it with your hand, and values close to 0 in complete darkness. You can delete the code not you see the values.</p>	
<p>5 Now, when the environment is dark, drag the when block in the Control category to the code writing area so that the RGB LED lights up. Unlike the when started block, this block constantly checks the condition we set, and if the condition is true, it executes the command below. The when started block runs the blocks below it from the moment you press the Start button.</p>	
<p>6 Then, to define the condition in the when block, drag and drop the 3<4 block in the operators category to the round part in the when block. The 3<4 block performs size-smallness control in condition operations. You can place the blocks you want to control in the fields written 3 and 4 in the block. In programming, different operators such as equal, not equal, greater than, less than are often used. In this project, we will check whether the value from the sensor is less than 90.</p>	
<p>7 Now drag the PicoBricks light sensor block from the Picobricks category and drop it into the circle that says 3 in the 3<4 operator in the when block.</p>	



8	<p>To complete the condition in the when block, delete the number 4 and type 90 from the keyboard. In this way, the program will check whether the Picobricks sensor values are less than 90% and run the codes under the when block when it is less than 90%.</p>	
9	<p>Drag and drop the PicoBricks set RGB LED block in the Picobricks category under the when block so that the LED can light up when the environment is dark, that is, when the LDR sensor value is less than 90. You can choose the color you want from the color palette that opens by clicking the green circle.</p>	
10	<p>Until this stage, we wrote the necessary codes for the LED to light up when the environment is dark. Now, we need to add the operations to the program when the environment is bright, that is, when the condition is not met. For this, take the when block from the Control category and leave it to the code writing area.</p>	
11	<p>In the when block, place the operators category $3 < 4$ block in the condition field. In this way, we can create the condition if the LDR sensor value is not less than 90.</p>	
12	<p>Again, drag the $3 < 4$ operator from the operators category to the not block.</p>	
13	<p>Drag the Picobricks light sensor block from the Picobricks category and place it in the circle that says 3 in the $3 < 4$ operator and delete the number 4 and write 90.</p>	



14	Drag and place the PicoBricks turn off RGB LED block in the Picobricks category under the when block so that the RGB LED turns off when the LDR sensor value is not less than 90.	
15	Test your codes by pressing the start button. If everything went well, when you close the Picobricks LDR sensor with your hand, the RGB LED will light up in the color you specified, and when you raise your hand, it will turn off.	

[Click](#) to access the project's MicroBlocks codes.

2.3.6. MicroPhyton Codes of the Project

```

import time
from machine import Pin, ADC
from picobricks import WS2812
#define the library

ldr = ADC(Pin(27))
ws = WS2812(6, brightness=0.4)
#define the input and output pins

#define colors
RED = (255, 0, 0)
GREEN = (0, 255, 0)
BLUE = (0, 0, 255)

COLORS = (RED, GREEN, BLUE)
#RGB color Code

while True:#while loop
    print(ldr.read_u16()) #print the value of the LDR sensor to the screen.

    if(ldr.read_u16()>10000):#let's check the ldr sensor
        for color in COLORS:

```



```
#turn on the LDR
ws.pixels_fill(color)
ws.pixels_show()

else:
    ws.pixels_fill((0,0,0)) #turn off the RGB
    ws.pixels_show()
```

2.3.7. Arduino C Codes of the Project

```
#include <Adafruit_NeoPixel.h>
#define PIN      6
#define NUMLEDS   1
#define LIGHT_SENSOR_PIN 27
Adafruit_NeoPixel leds = Adafruit_NeoPixel(NUMLEDS, PIN, NEO_GRB + NEO_KHZ800);
//define the libraries
int delayval = 250; // delay for half a second

void setup()
{
    leds.begin();
}

void loop() {
    int analogValue = analogRead(LIGHT_SENSOR_PIN);
    for(int i=0;i < NUMLEDS;i++)
    {
        if (analogValue > 200) {
            // pixels.Color takes RGB values, from 0,0,0 up to 255,255,255
            leds.setPixelColor(i, leds.Color(255,255,255));
            leds.show(); // This sends the updated pixel color to the hardware.
            delay(delayval);
        }
        else {
            leds.setPixelColor(i, leds.Color(0,0,0)); // white color code
            leds.show(); // This sends the updated pixel color to the hardware.
        }
    }
    delay(10);
```



}

GitHub Autonomous Lighting Project Page



<http://rbt.ist/autonomouslighting>



2.4. Thermometer

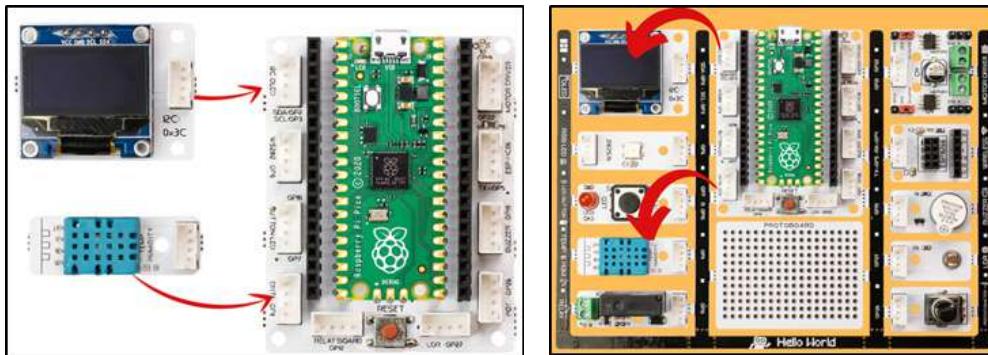
Sensors are the sense organs of electronic systems. We use our skin to feel, our eyes to see, our ears to hear, our tongue to taste, and our nose to smell. There are already many sense organs (sensors) in the picobrix. Also, new ones can be added. You can interact with the environment using humidity, temperature, light and many more sensors. Picobricks can measure the ambient temperature without the need for any other environmental component.

Ambient temperature is used in greenhouses, incubators, in environments used for the transport of drugs, briefly in situations where the temperature change must be constantly monitored. If you are going to do an operation on temperature change in your projects, you should know how to measure the ambient temperature. In this project, you will prepare a thermometer with Picobricks that will display the ambient temperature on the OLED screen.

2.4.1. Project Details and Algorithm

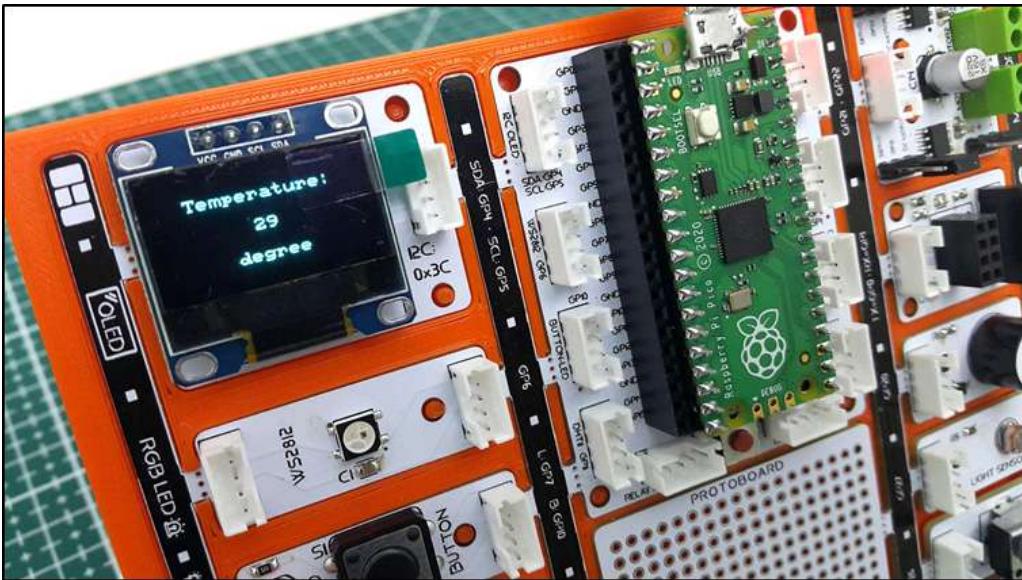
Picobricks has a DHT11 module. This module can sense the temperature and humidity in the environment and send data to the microcontroller. In this project, we will write the necessary codes to print the temperature values measured by the DHT11 temperature and humidity sensor on the OLED screen.

2.4.2. Wiring Diagram





2.4.3. Construction Stages of the Project

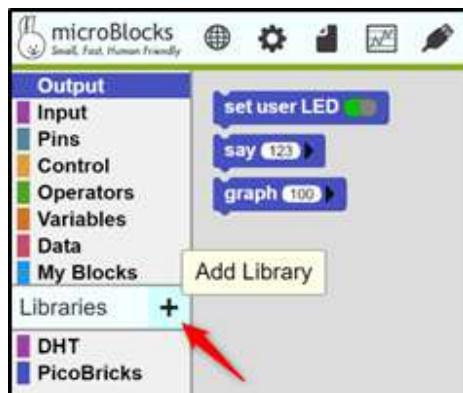


2.4.4. Project Proposal

In order to develop your project, you can make the red LED light up and a warning phrase appear on the screen when the temperature in the environment rises above 30 degrees.

2.4.5. Coding the Project with Microblocks

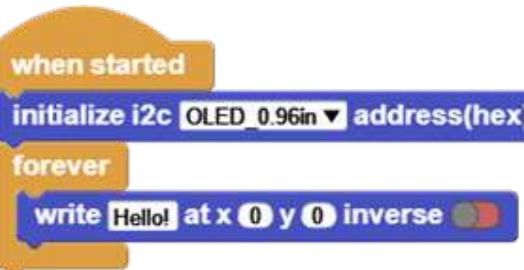
- 1 First, you need to import the OLED library into the Microblocks editor. You have to click the Add Library button for this.



- 2 In the File Open window that opens, click Library and then Graphics.





3	<p>Click the OLED Graphics.ulb library from the graphics libraries and click the open button in the lower right position of the window. You will see the OLED library added to the list of code categories.</p>	
4	<p>After adding the OLED library, drag the when started block in the Control menu and drop it to the code writing area so that the codes you wrote when Picobricks starts to run.</p>	
5	<p>Then you need to add the identification block of the OLED display. For this, drag the initialize i2c OLED block in the OLED category and add it under the when started block.</p>	
6	<p>Then drag the forever block from the Control menu to the bottom of the OLED definition block so that the codes you write will run continuously as long as Picobricks is running.</p>	
7	<p>To print text on the OLED screen, drag and drop the write Hello! at x0 y0 inverse block in the OLED category into the forever block. When you press the start button, Hello! You will see what you wrote. Hello here! You can delete the text and write the text you want so that it is displayed on the screen.</p>	
8	<p>Now in the write block delete the text Hello! and write Temperature: and change the x position to 15 and the y position to 10.</p>	



9	<p>Again take the write block from the OLED Graphics category and drag and drop it into the forever block and set the x position to 55 and the y position to 30. To print numeric values on the OLED screen, you must first convert these numeric values to textual expressions. To do this, drag and drop the block from the Data category, Hello! in the write block Drop it in the round area that says. Then drag the PicoBricks temperature (C) block from the Picobricks category and place it in the micro circle in the join block and delete the blocks text.</p>	<pre> when started initialize i2c OLED 6.96in address(hex) 3C reset pin# 0 flip 0 forever write Temperature at x 55 y 10 inverse write join PicoBricks temperature ("C") at x 65 y 30 inverse </pre>
10	<p>Finally, drag and drop the write block from the OLED Graphics category into the forever block. Hello! Delete the text, write degrees and change the x position to 40 and the y position to 50. By pressing the start button, you can view the ambient temperature measured by Picobricks' internal temperature sensor on the OLED screen.</p>	<pre> when started initialize i2c OLED 6.96in address(hex) 3C reset pin# 0 flip 0 forever write Temperature at x 55 y 10 inverse write join PicoBricks temperature ("C") at x 65 y 30 inverse write degree at x 40 y 50 inverse </pre>

[Click](#) to access the project's Microblocks codes.

2.4.6. Micropython Codes of the Project

```

from machine import Pin,I2C,ADC #to acces the hardware picobricks
from picobricks import SSD1306_I2C, DHT11 #oled library
import utime #time library
#to acces the hardware picobricks
WIDTH=128
HEIGHT=64
#define the weight and height picobricks

sda=machine.Pin(4)
scl=machine.Pin(5)
#we define sda and scl pins for inter-path communication
  
```



```
i2c=machine.I2C(0, sda=sda, scl=scl, freq=2000000)#determine the frequency values
oled=SSD1306_I2C(WIDTH, HEIGHT, i2c)
pico_temp=DHT11(Pin(11))
current_time=utime.time()
while True:
    if(utime.time() - current_time > 2):
        current_time = utime.time()
        pico_temp.measure()
        oled.fill(0)#clear OLED
        oled.show()
        temperature=pico_temp.temperature
        humidity=pico_temp.humidity
        oled.text("Temperature: ",15,10)#print "Temperature: " on the OLED at x=15 y=10
        oled.text(str(int(temperature)),55,25)
        oled.text("Humidity: ", 30,40)
        oled.text(str(int(humidity)),55,55)
        oled.show()#show on OLED
        utime.sleep(0.5)#wait for a half second
```

2.4.7. Arduino C Codes of the Project

```
#include <Wire.h>
#include <DHT.h>
#include "ACROBOTIC_SSD1306.h"

#define DHTPIN 11
#define DHTTYPE DHT11
//define the library

DHT dht(DHTPIN, DHTTYPE);
float temperature;
//define the temperature variable

void setup() {
    //define dht sensor and Oled screen
    Serial.begin(115200);
    dht.begin();
    Wire.begin();
    oled.init();
```



```
oled.clearDisplay();
}

void loop() {
    temperature = dht.readTemperature();
    Serial.print("Temp: ");
    Serial.println(temperature);
    oled.setTextXY(3,1);
    oled.putString("Temperature: ");
    //print "Temperature: " on the OLED at x=3 y=1
    oled.setTextXY(4,3);
    oled.putString(String(temperature));
    //print the value from the temperature sensor to the oled screen at x=4 y=3
    Serial.println(temperature);
    delay(100);
}
```

GitHub Thermometer Project Page



<http://rbt.ist/thermometer>



2.5. Graphic Monitor

When we look at the electronic items around us, you realize that they have many replaceable features and they are designed by engineers to be most useful to the user. Such as lighting systems, cooking systems, sound systems, cleaning systems. The way it works, the amount, the method, etc., by many system users. features can be programmed to change.

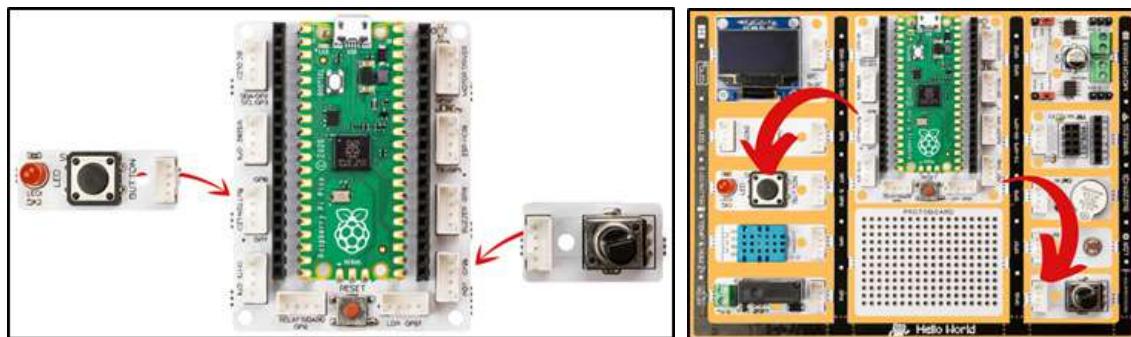
In robotic projects, in the processes of changing the sound level, changing the motor speed, changing the brightness of the light, the electrical voltage is sent in a way that creates a lower or higher effect. By decreasing the frequency of the electrical signal to the component, it can be operated at a lower level, and by increasing the frequency of the outgoing electrical signals, it can be operated at a higher level.

In systems without a screen, real-time graphic monitors are used to monitor some sensors and variables involved in the operation of the system. Graphic monitors make it very easy to detect the fault.

2.5.1. Project Details and Algorithm

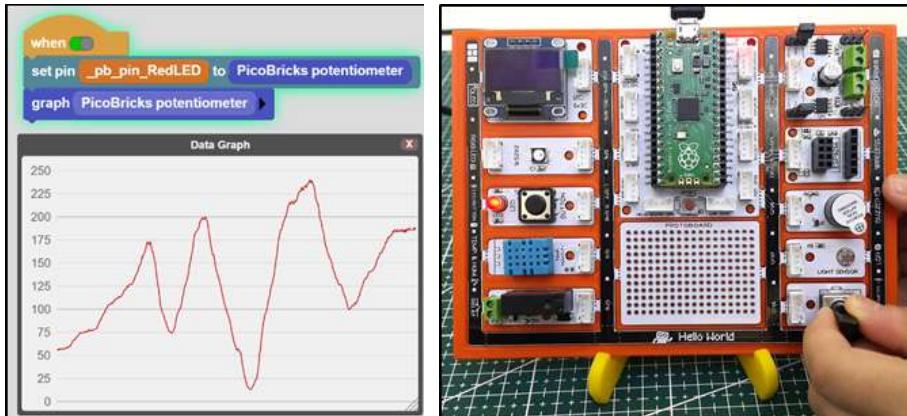
In this project, we will prepare a project in which we increase or decrease the brightness of the red LED with a potentiometer. In addition, we will simultaneously monitor the electrical change occurring during this process on the Microblocks graphic monitor. When the picobricks starts, the potentiometer value will be read continuously and the brightness value of the LED will be adjusted. Applications in which the effect of the electrical signal is reduced by changing the frequency is called PWM. We will send the analog values we read from the potentiometer as PWM signals to the red LED and we will be able to adjust the illumination intensity.

2.5.2. Wiring Diagram





2.5.3. Project Image



2.5.4. Project Proposal

As you turn the potentiometer, you can prepare a project that changes the volume of the sound coming out of the buzzer and displays the flowing data on the graphic monitor.

2.5.5. Coding the Project with Microblocks

1	To prepare the codes that will run as long as Picobricks is on, first drag the when block from the control category.	
2	To adjust the brightness of the LED, we can send it a 3.3V voltage by dividing it into 1024 parts. To do this, drag the set pin 1 to 1023 block from the pins category to the bottom of the when block. Instead of 1, you should write the GPIO number to which the red LED is connected on the Picobricks.	
3	Click the gear icon to add the pin number of the red LED in Picobricks. Select “show advanced blocks” from the drop-down menu.	



4	<p>You will see ready variables in the Variables category. You can find the pin numbers of all Picosbricks modules with the prefix "_pb_pin_". Drag and drop the _pb_pin_RedLED block to the field labeled 1 to represent the pin that the red LED is connected to.</p>	<pre>when green flag is pressed set pin _pb_pin_RedLED to 1023</pre>
5	<p>The area that says 1023 is the area where we set the level of current to be sent to the Red LED. Place the PicoBricks potentiometer block from the Picosbricks category here, as we will get this value in the potentiometer.</p>	<pre>when green flag is pressed set pin _pb_pin_RedLED to PicoBricks potentiometer</pre>
6	<p>Drag the graph 100 block from the output category to see the numerical data sent by the potentiometer on the graph.</p>	<pre>when green flag is pressed set pin _pb_pin_RedLED to PicoBricks potentiometer graph 100</pre>
7	<p>When we place the PicoBricks potentiometer block on the graph block, we write the necessary code for the values to appear on the graphic monitor and complete the project. You can watch the value of the Potentiometer change by running your code and clicking the graphic icon. </p>	<pre>when green flag is pressed set pin _pb_pin_RedLED to PicoBricks potentiometer graph PicoBricks potentiometer</pre>

[Click](#) to access the project's Microblocks codes.

2.5.6. Micropython Codes of the Project

```
from machine import Pin,ADC,PWM
from utime import sleep
#define libraries
led=PWM(Pin(7))
pot=ADC(Pin(26,Pin.IN))
```



```
#define the value we get from the led and pot
led.freq(1000)

while True: #while loop
    led.duty_u16(int((pot.read_u16())))
    print(str(int((pot.read_u16())))) #Turn on the LED according to the value from the
    #potentiometer
    sleep(0.1) #delay
```

2.5.7. Arduino C Codes of the Project

```
void setup() {
    // put your setup code here, to run once:
    pinMode (7,OUTPUT); //initialize digital pin 7 as an output
    pinMode (26,INPUT); //initialize digital pin 26 as an input  Serial.begin(9600); //start
    serial communication

}

void loop() {
    // put your main code here, to run repeatedly:
    int pot_val = analogRead(26);
    int led_val = map(pot_val, 0, 1023, 0, 255);
    digitalWrite(7, led_val);
    Serial.println(led_val);
    //turn on the LED according to the value from the potentiometer

    delay(100); //wait

}
```



GitHub Graphic Monitor Project Page



<http://rbt.ist/monitor>



2.6. Dominate the Rhythm

Many events in our lives have been digitized. One of them is sounds. The tone and intensity of the sound can be processed electrically. So we can extract notes electronically. The smallest unit of sounds that make up music is called a note. Each note has a frequency and intensity. With the codes we will write, we can adjust which note should be played and how long it should last by applying frequency and intensity.

In this project, we will prepare a music system that will play the melody of a song using the buzzer module and adjust the rhythm with the potentiometer module with Picobricks. You will also learn the use of variables, which has an important place in programming terminology, in this project.

2.6.1. Project Details and Algorithm

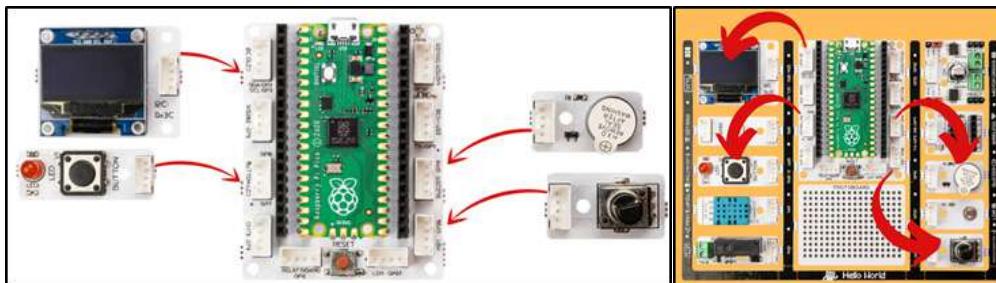
With Picobricks you can play any song whose sheet we know. We will use the button-LED module to start the song, the potentiometer module to adjust the speed of the song, and the buzzer module to play the notes.

Potentiometer is analog input module. It is variable resistance. As the amount of current flowing through it is turned, it increases and decreases like opening and closing a faucet. We will adjust the speed of the song by controlling this amount of current with codes. Buzzers change the sound levels according to the intensity of the current passing over them, and the sound tones according to the voltage frequency. With Microblock's, we can easily code the notes we want from the buzzer module by adjusting their tones and durations.

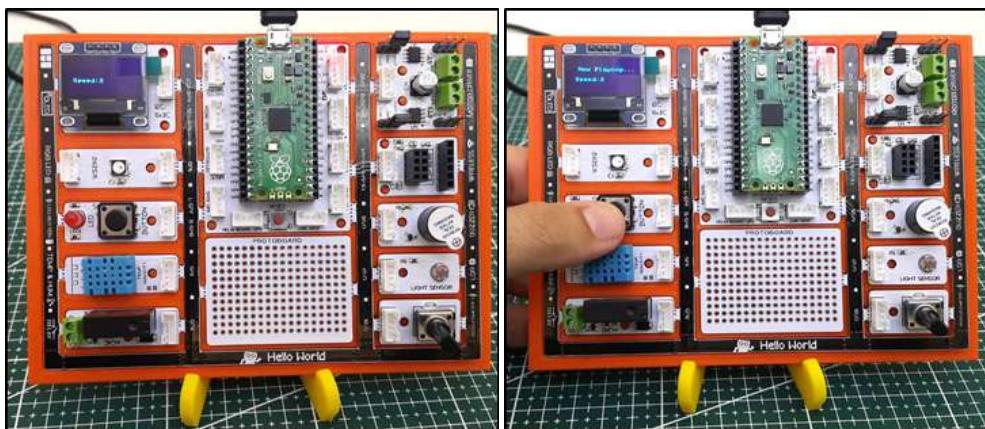
We will check the button press status in the project. We will make the melody start playing when the button is pressed. During the playing of the melody, we will use a variable called rthm to increase or decrease the playing times of the notes at the same rate. After Picobricks starts, we will enable the user to adjust the rthm variable with the potentiometer, either while playing the melody or before playing it. As long as Picobricks is on, we will divide the potentiometer value (0-1023) by 128 and assign it to the rthm variable. Variables are data structures that we use when we want to use values that can be changed by the user or sensors in our codes. When the user presses the button to start the song, we will prepare the note codes that will allow the notes to play for the duration calculated according to the rthm variable.



2.6.2. Wiring Diagram



2.6.3. Project Image

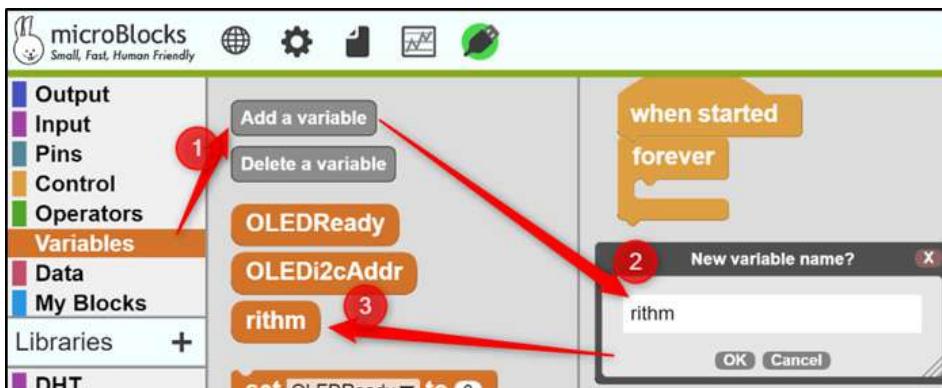


2.6.4. Project Proposal

To make your project more visual, you can light a different color LED according to the played note, show the note names and playing speed on the OLED screen.

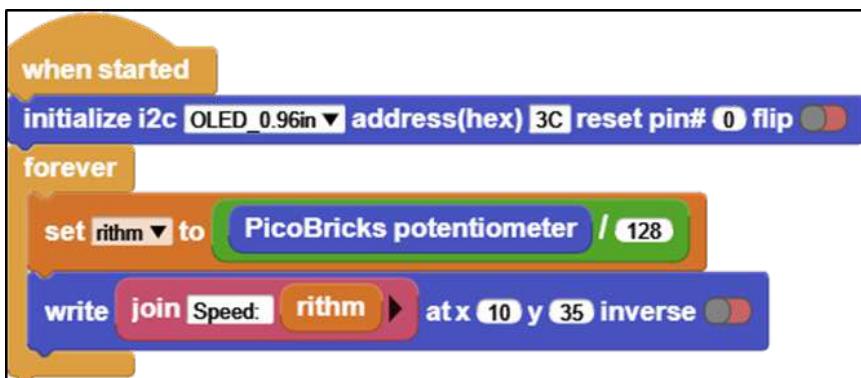
2.6.5. Coding the Project with Microblocks

Drag the “When started” from the Control category and the “forever” block under it to make Picobricks run the code continuously as soon as it starts. Add the OLED display library and place the code that initializes the display before the forever block so that you can display the user’s speed setting on the screen. Click the Add a variable button in the Variables category. When you type and confirm rithm in the dialog that opens, the rithm variable is created.





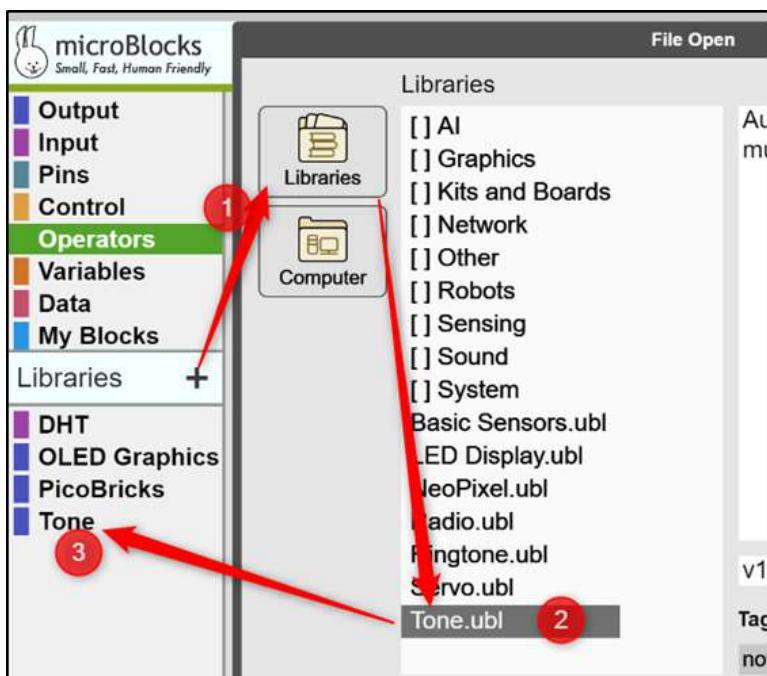
Select the variable name as rithm by clicking the black triangle inside the set OledReady to 0 block in the Variables category. Then put it inside the forever block. We will divide the value to be read from the potentiometer by 128 and use it to calculate the duration of the note. Place the 10/2 block from the Operators category in the variable block and set the denominator to 128. To read the potentiometer, drag the PicoBricks potentiometer block from the PicoBricks category to the 10 field in the division block. Take the write block in the OLED category. In the Hello field, place the Join block from the Data category and place the rithm variable.



Drag the when block from the Control category for continuous monitoring of the button press status on the Picobricks. Place the Picobricks button block inside.



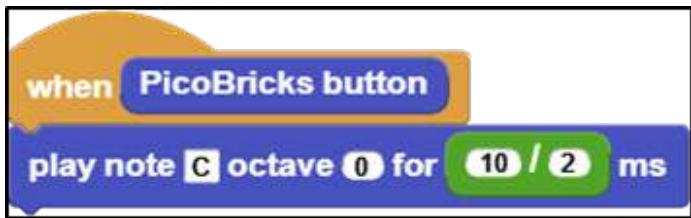
Add tone library to workspace.



Add block

play note C octave 0 for 500 ms

below the when block. C note name, 0 lets you set the note tone, and 500 lets you set the playing time. We will determine the playing times of the notes by dividing them by the rithm variable. Drag the 10/2 block from the operators category to the 500 field of the play note block.



The rithm variable takes the lowest value of 0. we will add one to the rithm value since 0 a quotient of a number will be infinite. Drag the “10+2” sum block from the operators category to the denominator(2) field in the division operation. Put the rithm variable in field 10 and change the number 2 to 1. Duplicate the blocks as many as the number of notes in the song with duplicate and duplicate all options by right clicking on the play note block.



Write the names of the notes in order. Write down the default beat times for each note in field 10 of the division operator. A value of 1000 represents a full stroke, 500 represents a half stroke, and 250 represents a quarter stroke. By adding the write block from the OLED category right after the when block, we show the user Now Playing. When the melody is finished, we will clear the screen.





These notes are the chorus part of the melody. So we have to code it to repeat twice and then clear the screen. Just take the repeat 10 block from the control category and place it inside. Set the number of repetitions to 2. After the loop, add the clear code from the OLED category.

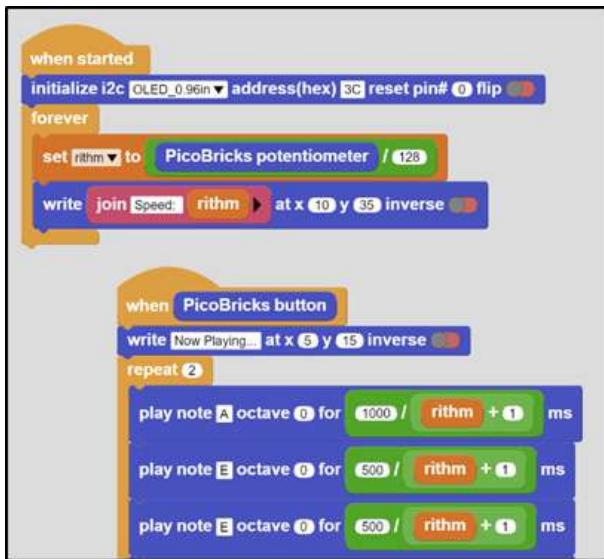
```

when PicoBricks button
write Now Playing... at x 15 y 15 inverse
repeat (2)
  play note A octave 0 for 1000 / rithm + 1 ms
  play note E octave 0 for 500 / rithm + 1 ms
  play note E octave 0 for 500 / rithm + 1 ms
  play note E octave 0 for 500 / rithm + 1 ms
  play note E octave 0 for 500 / rithm + 1 ms
  play note E octave 0 for 500 / rithm + 1 ms
  play note E octave 0 for 500 / rithm + 1 ms
  play note F octave 0 for 500 / rithm + 1 ms
  play note E octave 0 for 500 / rithm + 1 ms
  play note D octave 0 for 500 / rithm + 1 ms
  play note F octave 0 for 500 / rithm + 1 ms
  play note E octave 0 for 1000 / rithm + 1 ms
clear

```



When you run the codes, set the melody speed with the potentiometer on the Picobricks. Control the pace of the melody by turning the potentiometer while or before playing your melody.



[Click](#) to access the project's MicroBlocks codes.

2.6.6. Micropython Codes of the Project

```
from machine import Pin,PWM,ADC,I2C #to acces the hardware picobricks
from utime import sleep #time library
from picobricks import SSD1306_I2C
import utime

WIDTH=128
HEIGHT=64
#define the weight and height picobricks

sda=machine.Pin(4)
scl=machine.Pin(5)
#we define sda and scl pins for inter-path communication
i2c=machine.I2C(0, sda=sda, scl=scl, freq=2000000)#determine the frequency values
oled=SSD1306_I2C(WIDTH, HEIGHT, i2c)

button= Pin(10,Pin.IN,Pin.PULL_DOWN)
pot=ADC(Pin(26))
buzzer= PWM(Pin(20))
#determine our input and output pins
```



```

pressed = False
rithm = 0

tones = {
    "A3": 220,
    "D4": 294,
    "E4": 330,
    "F4": 349
}
#define the tones

mysong = ["A3","E4","E4","E4","E4","E4","F4","E4","D4","F4","E4"]#let's define the
tones required for our song in the correct order into a sequence
noteTime = [1,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,1]#define wait times between tones
into an array

def playtone(frequency):
    buzzer.duty_u16(6000)
    buzzer.freq(frequency)
#define the frequencies of the buzzer
def playsong(pin):
    global pressed
    pressed = True
#define the pins we will trigger
#play the tones with the right cooldowns
#An finally we need to tell the pins when to trigger, and the function to call when
they detect an event:
button.irq(trigger=Pin.IRQ_RISING, handler=playsong)
note_count = 9999
played_time = 0
while True:
    current_time = utime.ticks_ms()
    oled.show()
    oled.text("Press the button",0,0)

    if (note_count < len(mysong)):
        oled.fill(0)
        oled.text("Dominate ",30,10)
        oled.text("the ",45,25)
        oled.text("Rhythm ",35,40)
        rithm=((pot.read_u16()/65535.0)*20) +1
        if (current_time - played_time)/1000.0 >= noteTime[note_count]/rithm:
            playsong(pin)
            played_time = current_time
            note_count += 1

```



```

    played_time = utime.ticks_ms()
    playtone(tones[mysong[note_count]])
    note_count += 1
else:
    buzzer.duty_u16(0)

if pressed:

    note_count = 0
    pressed = False

```

2.6.7. Arduino C Codes of the Project

```

#include <Wire.h>
#include "ACROBOTIC_SSD1306.h"

int buzzer = 20;
int pot = 26;
int button = 10;
//define the buzzer, pot and button

int Re = 294;
int Mi = 330;
int Fa = 349;
int La = 440;
//define the tones

void setup()
{
    Wire.begin();
    oled.init();
    oled.clearDisplay();

    pinMode(buzzer, OUTPUT);
    pinMode(26, INPUT);
    pinMode(button, INPUT);
//determine our input and output pins

```



```
}

void loop()
{
    int rithm = (analogRead(pot))/146;
    String char_rithm = String(rithm);
    oled.setTextXY(3,4);
    oled.putString("Speed: ");
    oled.setTextXY(3,10);
    oled.putString(char_rithm);

    //print "Speed: " and speed value on the OLED at x=3 y=4

    delay(10);

    if (digitalRead(button) == 1){

        oled.clearDisplay();
        oled.setTextXY(3,2);
        oled.putString("Now playing...");

        //print "Speed: " and speed value on the OLED at x=3 y=4
        tone(buzzer, La); delay (1000/(rithm+1));
        tone(buzzer, Mi); delay (500/(rithm+1));
        tone(buzzer, Fa); delay (500/(rithm+1));
        tone(buzzer, Mi); delay (500/(rithm+1));
        tone(buzzer, Re); delay (500/(rithm+1));
        tone(buzzer, Fa); delay (500/(rithm+1));
        tone(buzzer, Mi); delay (1000/(rithm+1));
        //play the notes in the correct order and time when the button is pressed

        oled.clearDisplay(); //clear the screen
    }
    noTone(buzzer); //stop the buzzer
```



GitHub Dominate the Rhythm Project Page



<http://rbt.ist/rhythm>



2.7 Show Your Reaction

Now we will prepare a game that develops attention and reflexes. Moving quickly and being able to provide attention for a long time are important developmental characteristics of children. Preschool and primary school children do activities that increase their attention span and reflexes, as they are liked by their parents and teachers. The electronic system we will prepare will be a game that increases attention and develops reflexes. After finishing the project, you can compete with your friends. :)

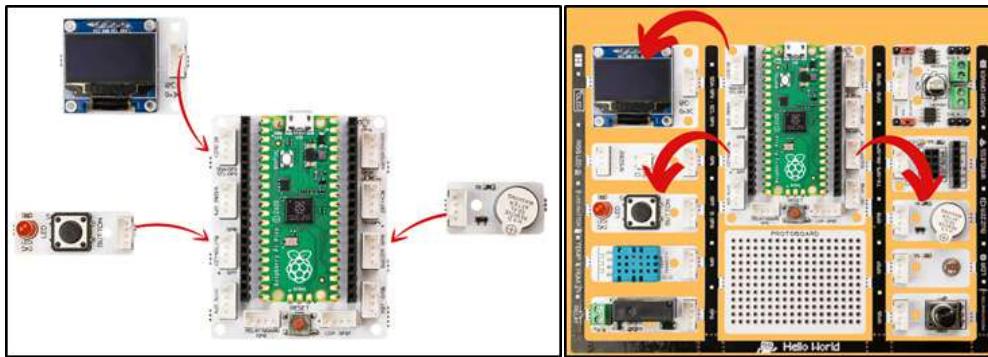
In this project you will learn about the randomness used in every programming language. With Picobricks, we will develop an electronic system using OLED display, Button-LED and Buzzer module.

2.7.1. Project Details and Algorithm

A timer starts running as soon as the Picobricks are turned on. With this timer, we can measure 1 thousandth of a second. One thousandth of a second is called a millisecond. Timers are used in many electronic systems in daily life. Timed lighting, ovens, irons, food processors...

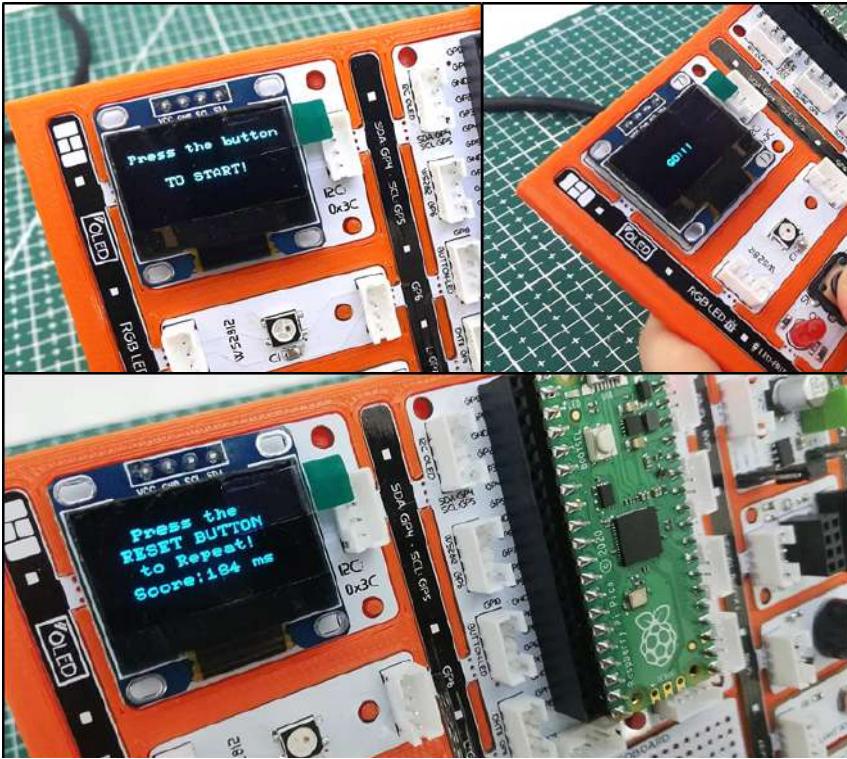
When our project starts working, we will display a welcome message on the OLED screen. Then we will print on the screen what the user has to do to start the game. In order to start the game, we will ask the player to prepare by counting backwards from 3 on the screen after the button is pressed. After the end of the countdown, the red LED will turn on in a random time between 2-10 seconds. We will reset the timer immediately after the red LED lights up. We will measure the timer as soon as the button is pressed again. This value we get will be in milliseconds. We will display this value on the screen as the player's reaction time.

2.7.2. Wiring Diagram





2.7.3. Project Image



2.7.4. Project Proposal

Picobricks needs to be reset to be able to restart the game. You can develop your project by asking the button to be pressed again to start the game again. You can also have the highest score and the last scorer printed on the screen at the end of the game.

2.7.5. Coding the Project with Microblocks

As soon as Picobricks starts after adding the OLED Display library, let's give place to the instruction statements on the screen.

```

when started
  initialize i2c OLED_0.96in address(hex) 3C
  reset pin# 0 flip
  write Press the button at x 0 y 10 inverse
  write TO START! at x 25 y 35 inverse
  
```

When the user presses the button of Picobricks, we take the for i in 10 loop from the Control category and set the number 10 to 3 so that it counts down as 3,2,1. This loop will increase the variable i from 1 to 3 every turn and run the code 3 times. To clear the screen and print 3...,2...,1... every second, you must place the OLED screen blocks and the wait block inside the for block as in the image.



```

when PicoBricks button
for i in 3
clear
write Hello! at x 25 y 35 inverse
wait 1000 millisecs

```

From the Data category, place the A block in the hello field in the write block, delete the micro text, and place the subtraction operator in the blocks field. Do $4 - i$ as subtraction.

join micro blocks

```

when PicoBricks button
for i in 3
clear
write join [ 4 - i ] at x 25 y 35 inverse
wait 1000 millisecs

```

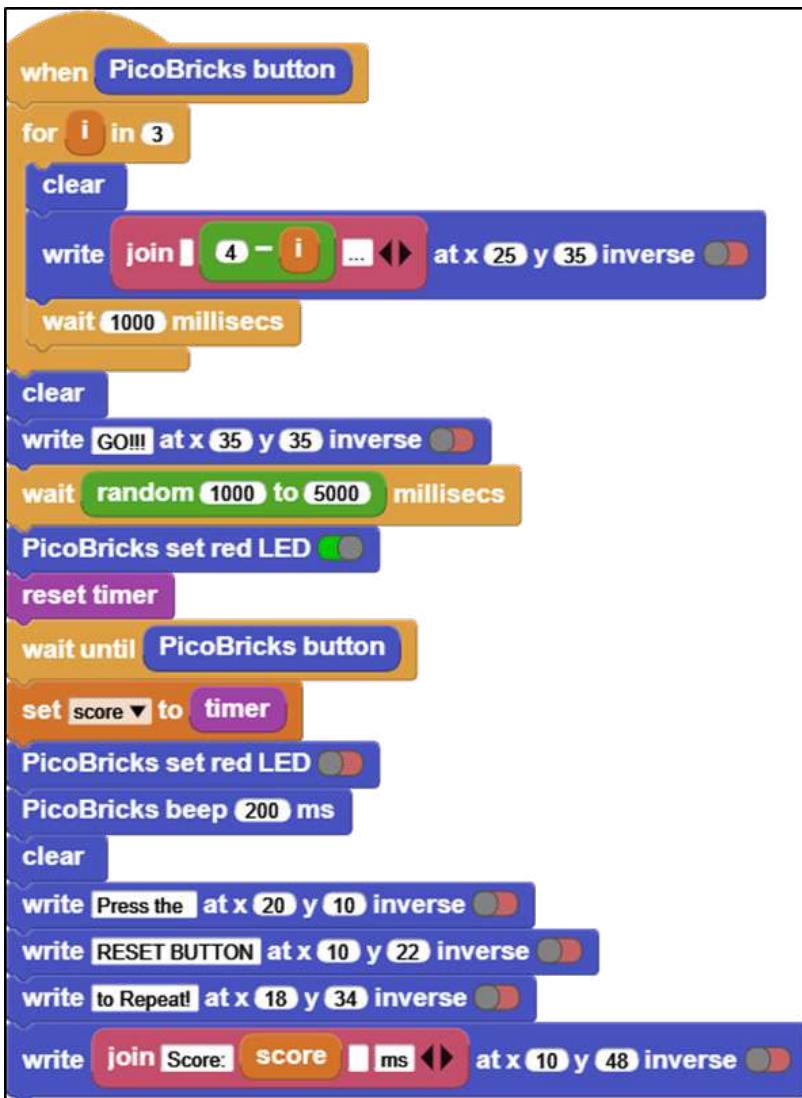
Just below the for loop, we add the block that clears the screen and announces the start of the game. To set the random wait, place the random 1 to 10 block in the wait block. Write 1000 in the first field and 5000 in the second field. This command will generate a random time in the range of 1 second to 5 seconds. Add the code that will turn on the red LED right after. We reset the timer with the reset timer block from the Input category. By adding the PicoBricks button block to the wait until block in the Control category, we make it wait until the button is pressed.

```

clear
write GO!! at x 35 y 35 inverse
wait random 1000 to 5000 millisecs
PicoBricks set red LED
reset timer
wait until PicoBricks button

```

Create a score variable to measure the timer value when the button is pressed. Place the timer value block in the score variable under the wait until block. Then turn off the red LED and make a 200 ms beep. Finally, clean the screen. We express pressing the reset button on the Picoboard to restart the game and print the reaction time on the screen. Enjoyable games.



[Click](#) to access the project's Microblocks codes.

2.7.6. Micropython Codes of the Project

```
from machine import Pin, I2C,Timer
from picobricks import SSD1306_I2C
import utime
import urandom
#define the library
WIDTH=128
HEIGHT=64
#define the width and height values
sda=machine.Pin(4)
scl=machine.Pin(5)
i2c=machine.I2C(0,sda=sda, scl=scl, freq=2000000)
oled= SSD1306_I2C(WIDTH, HEIGHT, i2c)
```



```

button = Pin(10,Pin.IN,Pin.PULL_DOWN)
led=Pin(7,Pin.OUT)
#define our input and output pins
while True:
    led.value(0)
    oled.fill(0)
    oled.text("press the button",0,10)
    oled.text("TO START!",25,25)
    oled.show()
#print "Press the button" and "TO START!" on the OLED screen
while button.value()==0:
    pass
    oled.fill(0)
    oled.text("Wait For LED",15,30)
    oled.show()
#write "wait for LED" on the screen when the button is pressed
utime.sleep(urandom.uniform(1,5))
led.value(1)
timer_start=utime.ticks_ms()
#wait for a random second and turn on the led
while button.value()==0:
    pass
    timer_reaction=utime.ticks_diff(utime.ticks_ms(), timer_start)
    pressed=True
    oled.fill(0)
    oled.text("Your Time",25,25)
    oled.text(str(timer_reaction),50,50)
    oled.show()
    led.value(0)
    utime.sleep(1.5)
#print the score and "Your Time" to the screen when the button is pressed.

```

2.7.7. Arduino C Codes of the Project

```

#include <Wire.h>
#include "ACROBOTIC_SSD1306.h"
//define the library

int buzzer = 20;
int button = 10;

```



```
int led = 7;
int La = 440;

int old_time = 0;
int now_time = 0;
int score = 0;
String string_score;
//define our integer and string variables

void setup(){
// put your setup code here, to run once:
Wire.begin();
oled.init();
oled.clearDisplay();

pinMode(led,OUTPUT);
pinMode(buzzer,OUTPUT);
pinMode(button,INPUT);
Serial.begin(9600);
//define the input and output pins
}

void loop(){
// put your main code here, to run repeatedly:
oled.setTextXY(3,0);
oled.putString("Press the button");
oled.setTextXY(5,4);
oled.putString("TO START");

if (digitalRead(button) == 1){

for (int i=3;i>0;i--){

String string_i = String(i);
oled.clearDisplay();
oled.setTextXY(4,8);
oled.putString(string_i);
delay(1000);

}
//count backwards from three
```



```
oled.clearDisplay();
oled.setTextXY(4,6);
oled.putString("GO!!!");
//print "GO!!" on the OLED at x=4 y=6

int random_wait = random(1000, 5000);
delay(random_wait);
//wait for a random second between 1 and 5

digitalWrite(led, HIGH);
old_time=millis();
 //turn on LED

while(!(digitalRead(button) == 1)){

now_time=millis();

score = now_time-old_time;
string_score = String(score);
 //save score as string on button press
}

digitalWrite(led, HIGH);
tone(buzzer, La);
delay (200);
noTone(buzzer);
//turn on LED and buzzer

oled.clearDisplay();
oled.setTextXY(1,4);
oled.putString("Press the");
//print "Press the" on the OLED at x=1 Y=4
oled.setTextXY(2,3);
oled.putString("RESET BUTON");
//print "RESET BUTTON" on the OLED at X=2 Y=3
oled.setTextXY(3,3);
oled.putString("to Repeat!");
//print "To Repeat!" on the OLED at X=3 Y=3
oled.setTextXY(6,3);
oled.putString("Score: ");
oled.setTextXY(6,9);
```



```
oled.putString(string_score);
oled.setTextXY(6,13);
oled.putString(" ms");
Serial.println(score);
//print score value to screen

delay(10000);
oled.clearDisplay();
//wait ten seconds and clear the screen
}

}
```

GitHub Show Your Reaction Project Page



<http://rbt.ist/reaction>



2.8. My Timer

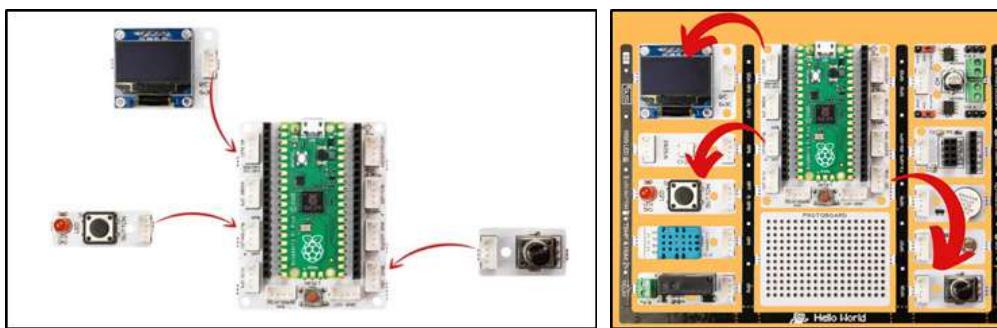
Measuring time is a simple but important task that we do in our daily lives without realizing it. A surgeon in surgery, a business person trying to catch up with a meeting, an athlete trying to win, a student trying to finish an exam or a chess match... Smart wrist watches, phones and even professional chronometers are used to measure time. Time is a variable that should be used very accurately in electronic systems. For example, a washing machine; how long the drum will rotate clockwise, how much counterclockwise, how many seconds water must flow in order to dissolve the detergent are tasks done by measuring time. To develop projects where time is of the essence, you need to know how to use it.

In this project, you will make your own time measuring device using Picobricks, OLED display, button and potentiometer modules. A Timer...

2.8.1. Project Details and Algorithm

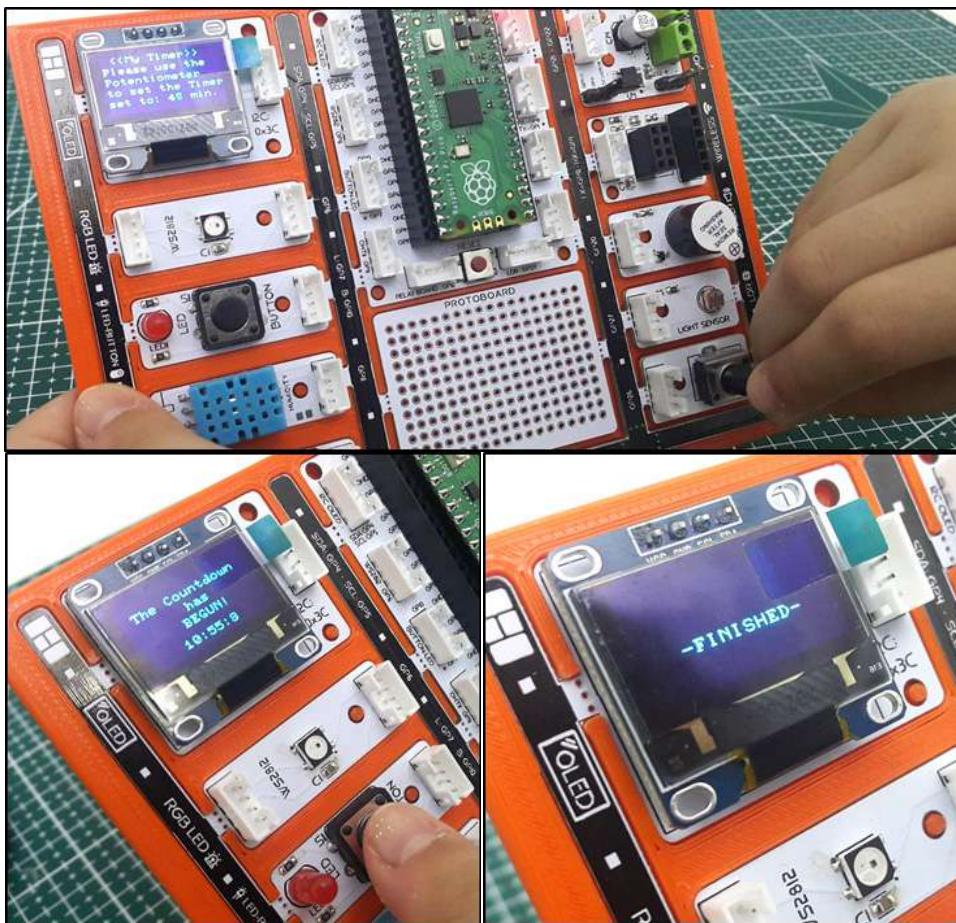
When Picobricks starts, let's put a statement on the screen that introduces the project and contains instructions. As the user turns the potentiometer, it will set a time in the range of 0-60 minutes. When the user presses the button of Picobricks after deciding the time with the potentiometer, it will start counting down in minutes and seconds on the screen. If the button is pressed while the time is running backwards, the Timer will stop and show the remaining time on the screen. If the minute, second and second value reaches zero without pressing the button, a notification stating that the time has expired will be displayed on the screen and the program will be stopped.

2.8.2. Wiring Diagram





2.8.3. Construction Stages of the Project



2.8.4. Project Proposal

You can add a beep to the start of the Timer. When the time is reset, you can give different and high tone warnings with the buzzer and announce that the time is up from afar.

2.8.5. Coding the Project with MicroBlocks

First, create the variables to be used in the project. setTimer, min, sec, msec set the values of the sec and msec variables to 59 and 9 respectively. Then insert the codes of the splash screen.

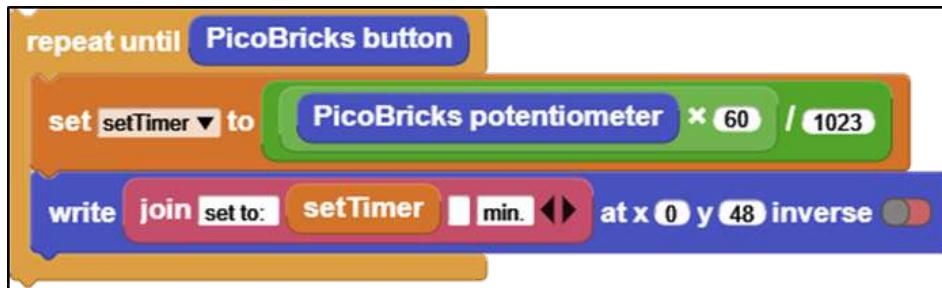
```

when started
set sec ▾ to 59
set msec ▾ to 9
initialize i2c OLED_0.96in ▾ address(hex) 3C reset pin# 0 flip ⚡
write <<My Timer>> at x 10 y 0 inverse ⚡
write Please use the at x 0 y 12 inverse ⚡
write Potentiometer at x 0 y 24 inverse ⚡
write to set the Timer at x 0 y 36 inverse ⚡

```



Let's assign the adjusted minute value to the setTimer variable by multiplying the value read from the potentiometer by 60 and dividing by 1023 until the button is pressed, and print this variable on the screen.



```

repeat until [PicoBricks button]
  set [setTimer v] to [(PicoBricks potentiometer × 60) ÷ 1023]
  write [join [setTimer min.] at x [0] y [48] inverse [red]]
end

```

Pressing the button after the repeat until block clears the screen and waits for a short while, then GO! Let's broadcast.

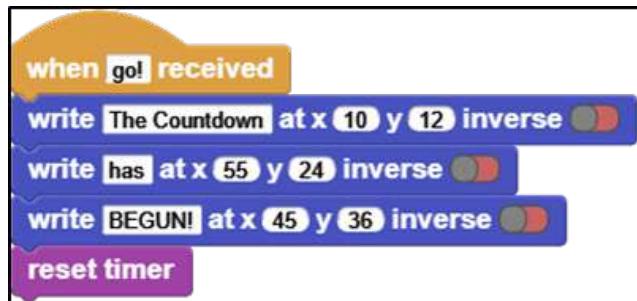


```

clear
wait (100) millisecs
broadcast [go!]

```

GO! Let's place the expression that states that the countdown has started when the broadcast is received. Then let's reset the timer.

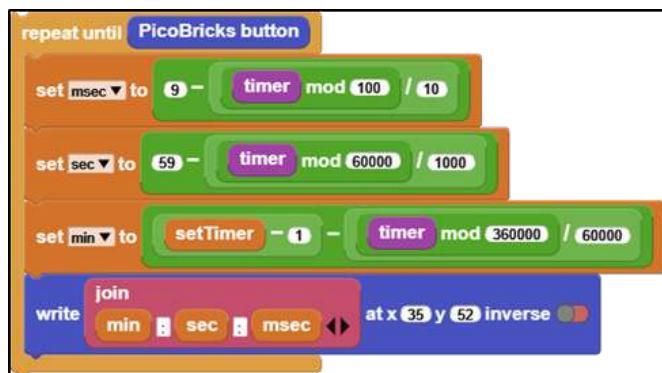


```

when [go!] received
  write [The Countdown] at x [10] y [12] inverse [red]
  write [has] at x [55] y [24] inverse [red]
  write [BEGUN!] at x [45] y [36] inverse [red]
  reset timer
end

```

Let's set the msec, sec and min values by measuring the timer until the button is pressed a second time. msec represents milliseconds (0-9), sec seconds (0-59), min (0-59) minutes. Then let's show the variables on the OLED screen.



```

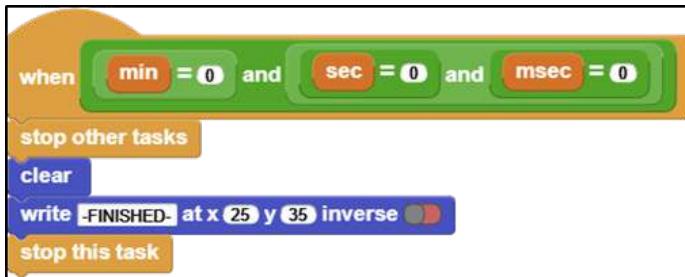
repeat until [PicoBricks button]
  set [msec v] to [(timer mod 100) ÷ 10]
  set [sec v] to [(timer mod 60000) ÷ 1000]
  set [min v] to [(setTimer - 1) - (timer mod 360000) ÷ 360000]
  write [join [min : sec : msec] at x [35] y [52] inverse [red]]
end

```

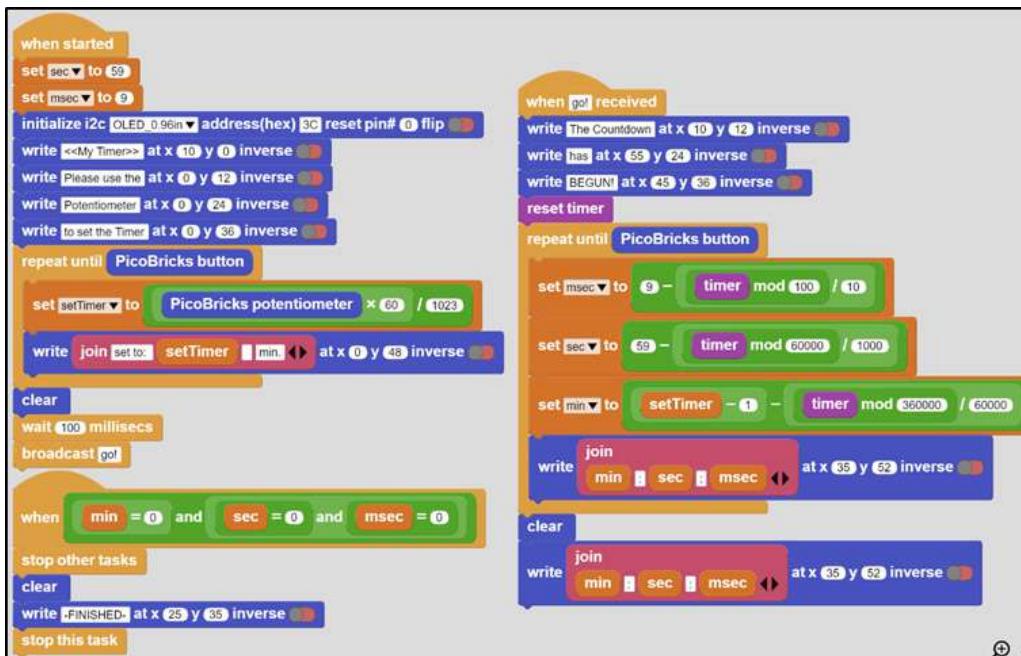
If the timer is to be terminated before the time is up, let's make the remaining time appear on the screen when the button is pressed.



Finally, let's constantly check if the sec, min and msec variables are 0. When all three of these variables are zero, it means the time is up. So, let's stop all the codes from running after we stop the other code blocks from running and print the end message on the screen.



The final code should be as follows. Once you click the start button, your project will run.



[Click](#) to access the project's Microblocks codes.

2.8.6. MicroPython Codes of the Project

```
from machine import Pin, I2C, ADC, Timer #to acces the hardware picobricks
from picobricks import SSD1306_I2C #oled library
import utime #time library
```

WIDTH = 128



```
HEIGHT = 64
#define the width and height values

sda=machine.Pin(4)
scl=machine.Pin(5)
#we define sda and scl pins for inter-path communication
i2c=machine.I2C(0,sda=sda, scl=scl, freq=1000000)#determine the frequency values

oled = SSD1306_I2C(128, 64, i2c)
pot = ADC(Pin(26))
button = Pin(10,Pin.IN,Pin.PULL_DOWN)
#determine our input and output pins

oled.fill(0)
oled.show()
#Show on OLED

time=Timer()
time2=Timer()
time3=Timer()
#define timers

def minute(timer):
    global setTimer
    setTimer -=1

def second(timer):
    global sec
    sec-=1
    if sec== -1:
        sec=59

def msecnd(timer):
    global msec
    msec-=1
    if msec== -1:
        msec=99

#We determine the increments of the minute-second and millisecond values.
sec=59
msec=99
```



```
global setTimer
```

```
while button.value()==0:  
    setTimer=int((pot.read_u16()*60)/65536)+1  
    oled.text("Set timer:" + str(setTimer) + " min",0,12)  
    oled.show()  
    utime.sleep(0.1)  
    oled.fill(0)  
    oled.show()
```

#If the button is not pressed, the value determined by the potentiometer is printed on the OLED screen.

```
setTimer-=1
```

```
time.init(mode=Timer.PERIODIC,period=60000,callback=minute)  
time2.init(mode=Timer.PERIODIC,period=1000,callback=second)  
time3.init(mode=Timer.PERIODIC,period=10,callback=msecond)  
#We determine the periods of minutes, seconds and milliseconds.  
utime.sleep(0.2)#wait for 0.2 second
```

```
while button.value()==0:  
    oled.text("min:" + str(setTimer),50,10)  
    oled.text("sec:" + str(sec),50,20)  
    oled.text("ms:" + str(msec),50,30)  
    oled.show()  
    utime.sleep(0.008)  
    oled.fill(0)  
    oled.show()  
if(setTimer==0 and sec==0 and msec==99):  
    utime.sleep(0.1)  
    msec=0  
    break;
```

#When the button is pressed, it prints the min-sec-ms values to the OLED screen in the determined x and y coordinates.

```
oled.text(str(setTimer),60,10)  
oled.text(str(sec),60,20)  
oled.text(str(msec),60,30)  
oled.text("Time is Over!",10,48)  
oled.show()
```



#Print the minutes, seconds, milliseconds and “Time is Over” values to the X and Y coordinates determined.

2.8.7. Arduino C Codes of the Project

```
#include <Wire.h>
#include "ACROBOTIC_SSD1306.h"
//define the library
int minute;
int second = 59;
int milisecond = 9;
int setTimer;
//define variables
void setup() {
// put your setup code here, to run once:
pinMode(10,INPUT);
pinMode(26,INPUT);

Wire.begin();
oled.init();
oled.clearDisplay();
//define the input-output pins and the oled Display

}

void loop() {
// put your main code here, to run repeatedly:
oled.setTextXY(1,2);
oled.putString("=><My Timer=>");
oled.setTextXY(3,1);
oled.putString("Please use the");
oled.setTextXY(4,1);
oled.putString("Potentiometer");
oled.setTextXY(5,0);
oled.putString("to set the Timer");

//print the "<>My Timer=>", "Please use the", "Potentiometer" and "to set the Timer."
//to the x and y coordinates determinates on the OLED screen.
```



```
delay(3000);
oled.clearDisplay();
//we waited three seconds and cleared

while(!(digitalRead(10) == 1))
{
    setTimer = (analogRead(26)*60)/1023;
    oled.setTextXY(3,1);
    oled.putString("set to:");
    oled.setTextXY(3,8);
    oled.putString(String(setTimer));
    oled.setTextXY(3,11);
    oled.putString("min.");
//determine the min value with the potentiometer and print it on the screen until the
button is pressed.
}
oled.clearDisplay();
oled.setTextXY(1,1);
oled.putString("The Countdown");
oled.setTextXY(2,3);
oled.putString("has begin!");
//print the "The Countdown" and "has begin!" to the x and y coordinates
determineted on the OLED screen
while(!(digitalRead(10) == 1))
{
    milisecond = 9- (millis()%100)/10;
    second = 59-(millis()%60000)/1000;
    minute = (setTimer-1)-((millis()%360000)/60000);

    oled.setTextXY(5,3);
    oled.putString(String(minute));
    oled.setTextXY(5,8);
    oled.putString(String(second));
    oled.setTextXY(5,13);
    oled.putString(String(milisecond));
    oled.setTextXY(5,6);
    oled.putString(":");
    oled.setTextXY(5,11);
    oled.putString(":");
//when the button is pressed, decrease the millisecond, second and minute values
```



and write to the screen.

```
}

oled.setTextXY(5,3);
oled.putString(String(minute));
oled.setTextXY(5,8);
oled.putString(String(second));
oled.setTextXY(5,13);
oled.putString(String(milisecond));
oled.setTextXY(5,6);
oled.putString(":");
oled.setTextXY(5,11);
oled.putString(":");
delay(10000);

//print the minute, second and millisecond values to the x and y coordinates
determined on the OLED screen

if (minute==0 & second==0 & milisecond==0){

oled.setTextXY(5,3);
oled.putString(String(minute));
oled.setTextXY(5,8);
oled.putString(String(second));
oled.setTextXY(5,13);
oled.putString(String(milisecond));
oled.setTextXY(5,6);
oled.putString(":");
oled.setTextXY(5,11);
oled.putString(":");
oled.putString("-finished-");
oled.setTextXY(7,5);
delay(10000);

//print the minute, second, millisecond values and "-finisehd-" to the x and y
coordinates determined on the OLED screen.

}

}
```



GitHub My Timer Project Page



<http://rbt.ist/mytimer>



2.9. Alarm Clock

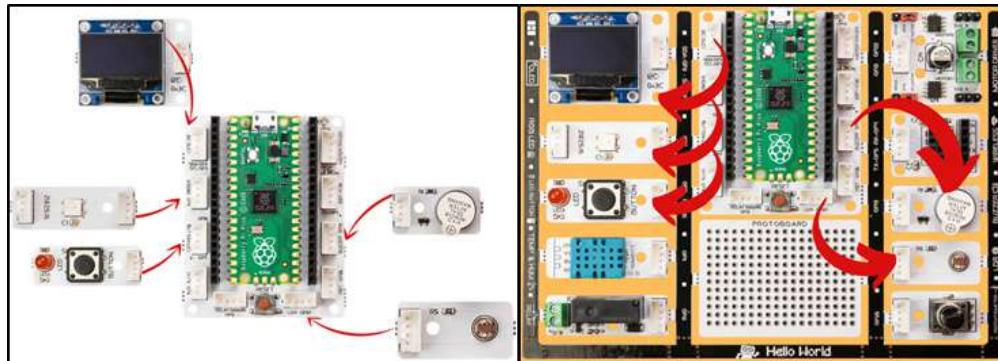
Global warming is affecting the climate of our world worse every day. Countries take many precautions and sign agreements to reduce the effects of global warming. The use of renewable energy sources and the efficient use of energy is an issue that needs attention everywhere, from factories to our rooms. Many reasons such as keeping road and park lighting on in cities due to human error, and the use of high energy consuming lighting tools reduce energy efficiency. Many electronic and digital systems are developed and programmed by engineers to measure the light, temperature and humidity values of the environment and ensure that they are used only when needed and in the right amounts.

In this project, we will create a timer alarm that adjusts for daylight using the light sensor in Picosbricks.

2.9.1. Project Details and Algorithm

In this project we will make a simple alarm application. The alarm system we will design is designed to sound automatically in the morning. For this, we will use LDR sensor in the project..At night, the OLED screen will display a good night message to the user, in the morning, an alarm will sound with a buzzer sound, a good morning message will be displayed on the screen, and the RGB LED will light up in white for light notification. The user will have to press the button of Picosbricks to stop the alarm. After these processes, which continue until the alarm is stopped, when the button is pressed, the buzzer and RGB LED will turn off and a good day message will be displayed on the OLED screen.

2.9.2. Wiring Diagram





2.9.3. Project Image



2.9.4. Project Proposal

You can improve the project by adding a melody as an alarm sound instead of a beep. Or, instead of an alarm set according to daylight with the LDR sensor, you can develop an alarm that sounds at the specified time, where the time information is arranged via the button and OLED screen.

2.9.5. Coding the Project with MicroBlocks

First of all, add the OLED Graphics library to the program and add the initialize i2c and write blocks from the OLED category under the when started block so that “Good night” is written on the screen when Pico starts. If you are doing this project during the day, you will need to cover the LDR sensor with your hand to test the program. For this reason, after the write block, wait for 2 seconds with the wait block and turn off the LDR sensor with your hand within those 2 seconds.



In the autonomous lighting project, we read the LDR sensor values with the say block and saw values above 90 when the environment was bright and below 90 when it was dark. In this project, we can use the value 90 to determine the day and night difference. By using the say block, you can view the LDR sensor value in your environment and change the value that the alarm sounds.

When the value from the LDR sensor is greater than 90, we must use the when block and specify the condition for the alarm to activate. Under the when block, we should use the repeat until block so that the alarm can continue to sound until the user presses the button, and we must specify the block of pressing the button of Picobricks as a condition. You should place the RGB LED, buzzer and OLED screen codes inside the repeat until block, which will run the blocks in it until the specified condition is met. You can change the screen texts, RGB LED color and buzzer playing



time.

```

when 90 < PicoBricks light sensor
    initialize i2c OLED_0.96in address(hex) 3C reset pin# 0 flip
repeat until PicoBricks button
    write Good morning at x 15 y 32 inverse
    PicoBricks set RGB LED color
    PicoBricks beep 500 ms

```

The codes written so far are the codes that perform the actions to be taken when the system starts and in the morning. Now, let's write the necessary codes for the actions to be taken after the button is pressed while the alarm is sounding.

```

when 93 < PicoBricks light sensor
    initialize i2c OLED_0.96in address(hex) 3C reset pin# 0 flip
repeat until PicoBricks button
    write Good morning at x 15 y 32 inverse
    PicoBricks set RGB LED color
    PicoBricks beep 500 ms
    write have a nice day at x 0 y 32 inverse
    PicoBricks turn off RGB LED
stop this task

```

When the button is pressed while the alarm is sounding, the program will go out of the repeat until block and run the codes in the next line. For this reason, you should turn off the RGB LED and write the screen text blocks under the repeat until block.

Alarm Clock

```

when started
    initialize i2c OLED_0.96in address(hex) 3C reset pin# 0 flip
    write Good night at x 25 y 32 inverse
    wait 2000 millisecs

when 93 < PicoBricks light sensor
    initialize i2c OLED_0.96in address(hex) 3C reset pin# 0 flip
repeat until PicoBricks button
    write Good morning at x 15 y 32 inverse
    PicoBricks set RGB LED color
    PicoBricks beep 500 ms
    write have a nice day at x 0 y 32 inverse
    PicoBricks turn off RGB LED
stop this task

```



When you type all the codes and press the Start button, “Good night” will be displayed on the screen and if it is dark, the program will wait until it is bright. If you are testing the project in a bright environment, turn off the LDR sensor with your hand within 2 seconds after pressing the Start button. Then, when the environment is illuminated, you will see that the alarm is working with the OLED screen, buzzer and RGB LED. You can stop the alarm by pressing the button.

[**Click**](#) to access the project’s MicroBlocks codes.

2.9.6. Micropython Codes of the Project

```
from machine import Pin, I2C, ADC, PWM#to access the hardware on the pico
from picobricks import SSD1306_I2C#OLED Screen Library
import utime
from picobricks import WS2812#ws8212 library

#OLED Screen Settings
WIDTH = 128
HEIGHT = 64

sda=machine.Pin(4)
scl=machine.Pin(5)
#initialize digital pin 4 and 5 as an OUTPUT for OLED Communication

i2c=machine.I2C(0,sda=sda, scl=scl, freq=1000000)
neo = WS2812(pin_num=6, num_leds=1, brightness=0.3)#initialize digital pin 6 as an
OUTPUT for NeoPixel

oled = SSD1306_I2C(WIDTH, HEIGHT, i2c)
ldr = ADC(Pin(27))#initialize digital pin 6 as an OUTPUT for NeoPixel
button = Pin(10,Pin.IN,Pin.PULL_DOWN)#initialize digital pin 10 as an INPUT for
button
buzzer = PWM(Pin(20, Pin.OUT))#initialize digital pin 20 as an OUTPUT for buzzer
buzzer.freq(1000)

BLACK = (0, 0, 0)
WHITE = (255, 255, 255)
#RGB black and white color code
oled.fill(0)
oled.show()
```



```
neo.pixels_fill(BLACK)
neo.pixels_show()

if ldr.read_u16()<4000:
    wakeup = True
else:
    wakeup = False

while True:
    while wakeup==False:
        oled.fill(0)
        oled.show()
        oled.text("Good night",25,32)
        oled.show()
        #Show on OLED and print "Good night"
        utime.sleep(1)
    if ldr.read_u16()<4000:
        while button.value()==0:
            oled.fill(0)
            oled.show()
            oled.text("Good morning",15,32)
            oled.show()
        #Print the minutes, seconds, milliseconds and "Goog morning" values to the
        X and Y coordinates determined on the OLED screen.
        neo.pixels_fill(WHITE)
        neo.pixels_show()
        buzzer.duty_u16(6000)
        utime.sleep(1)
        #wait for one second
        buzzer.duty_u16(0)
        utime.sleep(0.5)
        #wait for half second
        wakeup=True
        neo.pixels_fill(BLACK)
        neo.pixels_show()
        oled.fill(0)
        oled.show()
        oled.text("Have a nice day!",0,32)
    #Print the minutes, seconds, milliseconds and "Have a nice day!" values to the X
    and Y coordinates determined on the OLED screen.
    oled.show()
    if ldr.read_u16()>40000:
```



```
wakeup= False
```

```
utime.sleep(1)
#wait for one second
```

2.9.7. Arduino C Codes of the Project

```
#include <Adafruit_NeoPixel.h>
#ifndef __AVR__
#include <avr/power.h>
#endif
#define PIN      6

#define NUMPIXELS 1
Adafruit_NeoPixel pixels(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);
#include <Wire.h>
#include "ACROBOTIC_SSD1306.h"
int button;

void setup() {
    Wire.begin();
    oled.init();
    oled.clearDisplay();

#if defined(__AVR_ATtiny85__) && (F_CPU == 16000000)
    clock_prescale_set(clock_div_1);
#endif
    pinMode(10,INPUT);
    pinMode(27,INPUT);
    pinMode(20,OUTPUT);

    pixels.begin();
    pixels.setPixelColor(0, pixels.Color(0, 0, 0));
    pixels.show();
}

void loop() {

    oled.setTextXY(4,3);
    oled.putString("Good night");

    if (analogRead(27)<200){
```



```
while(!(button == 1)) {  
  
    button=digitalRead(10);  
  
    oled.setTextXY(4,2);  
    oled.putString("Good morning");  
    pixels.setPixelColor(0, pixels.Color(255, 255, 255));  
    pixels.show();  
    tone(20,494);  
}  
oled.clearDisplay();  
oled.setTextXY(4,1);  
oled.putString("Have a nice day");  
noTone(20);  
pixels.setPixelColor(0, pixels.Color(0, 0, 0));  
pixels.show();  
delay(10000);  
}  
}
```

GitHub Alarm Clock Project Page



<http://rbt.ist/alarm>



2.10. Know Your Color

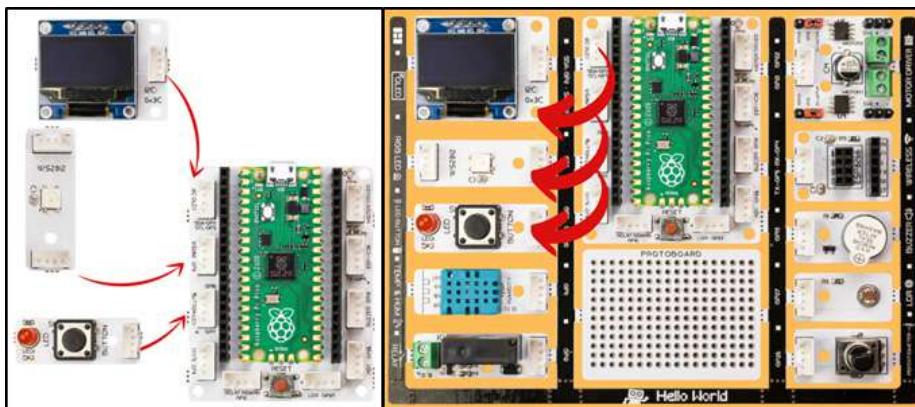
LEDs are often used on electronic systems. Each button can have small LEDs next to each option. By making a single LED light up in different colors, it is possible to do the work of more than one LED with a single LED. LEDs working in this type are called RGB LEDs. It takes its name from the initials of the color names Red, Green, Blue. Another advantage of this LED is that it can light up in mixtures of 3 primary colors. Purple, turquoise, orange...

In this project you will learn about the randomness used in every programming language. We will prepare a enjoyable game with the RGB LED, OLED screen and button module of Picobricks.

2.10.1. Project Details and Algorithm

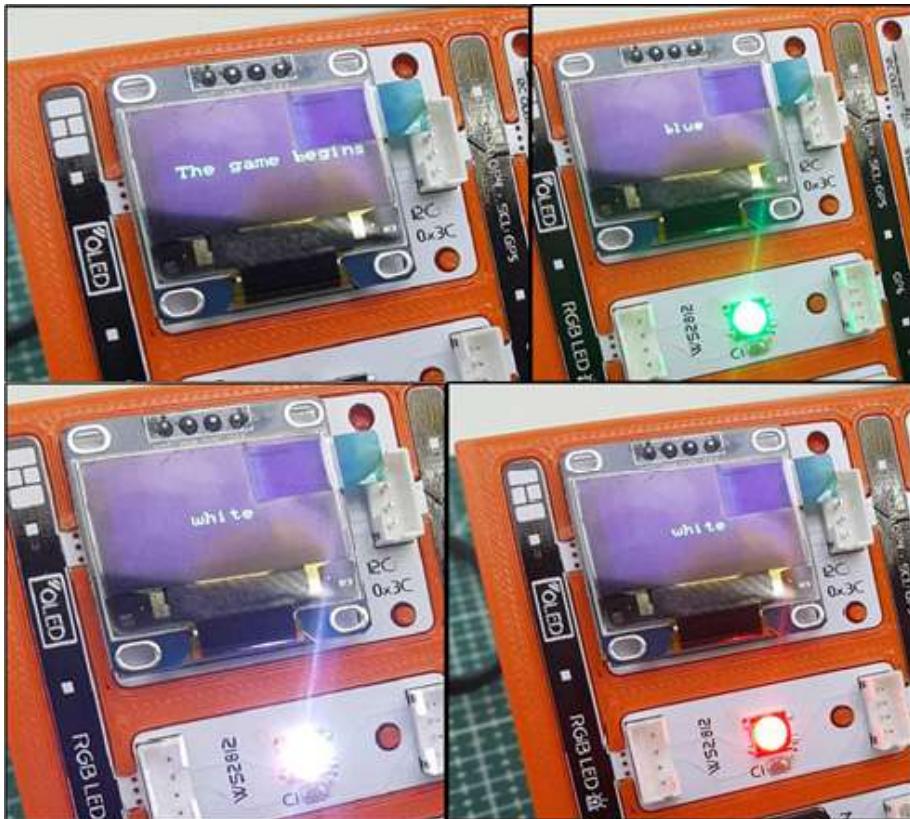
The game we will build in the project will be built on the user knowing the colors correctly or incorrectly. One of the colors red, green, blue and white will light up randomly on the RGB LED on Picobricks, and the name of one of these four colors will be written randomly on the OLED screen at the same time. The user must press the button of Picobricks within 1.5 seconds to use the right of reply. The game will be repeated 10 times, each repetition will get 10 points if the user presses the button when the colors match, or if the user does not press the button when they do not match. If the user presses the button even though the colors do not match, he will lose 10 points. After ten repetitions, the user's score will be displayed on the OLED screen. If the user wishes, he may not use his right of reply by not pressing the button.

2.10.2. Wiring Diagram





2.10.3. Project Image

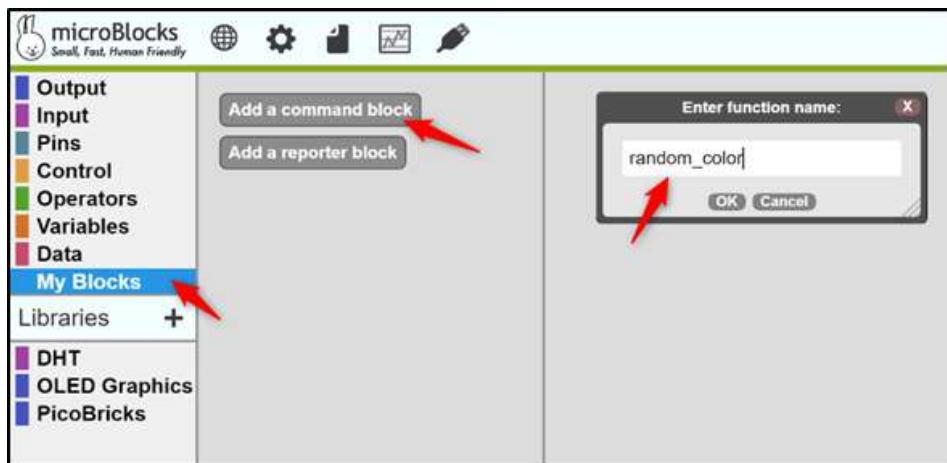


2.10.4. Project Proposal

You can make the game more enjoyable by making it a little more difficult. For example, you can speed up the game by reducing the repetition time of the colors. Or, instead of losing points when the user presses the button in the wrong place, you can finish the game and start it again.

2.10.5. Coding the Project with MicroBlocks

Since we will use an OLED display in the project, you must first add the OLED Graphics library to the program. Then you have to create two variables named OLED color and RGB color to generate random colors on the OLED screen and RGB LED. In order for the program to work more stable, we can perform the random determination of colors by creating a function and calling this function. Functions are created by structuring codes consisting of one or more operation lines as a code block. Once functions are created, they can be called anywhere in the program using just the function name. To create a function, you must click the My Blocks button in the code categories section, then click the Add a command block button and name the function you will create in the Enter function name window that opens. We can give the name random_color to the function we will create in this project.



After creating the function, the define random_color block will come to the code writing area. You can write function codes under this block. In the random_color function, we first assign a random number between 1 and 4 to the OLED_color and RGB_color variables that we created before. For this, you should use the random block in the operators category. Then you should compare the random values assigned to the variables with the if block and edit the RGB LED color and OLED screen texts.

After defining the random_color function, we can write our program under the when started block. In this code group, you must first write the codes that should run when the program starts. After the start blocks, you should call the random_color function you created earlier inside the repeat 10 block. After running random_color 10 times, you should wait for a while, clear the screen and turn off the RGB LED. You have to repeat all these steps 10 times. When you run the codes so far, you will see 10 times random colors appear on the RGB LED and OLED screen at 1.5 second intervals.

```

define [random_color]
  set [OLED color v] to [random (1) to (4)]
  set [RGB_color v] to [random (1) to (4)]
  if [RGB_color = 1] then
    PicoBricks set RGB LED color [red v]
  end
  if [RGB_color = 2] then
    PicoBricks set RGB LED color [green v]
  end
  if [RGB_color = 3] then
    PicoBricks set RGB LED color [blue v]
  end
  if [RGB_color = 4] then
    PicoBricks set RGB LED color [white v]
  end
  if [OLED_color = 1] then
    clear
    write [red] at x [50] y [32] inverse [red]
  end
  if [OLED_color = 2] then
    clear
    write [green] at x [45] y [32] inverse [green]
  end
  if [OLED_color = 3] then
    clear
    write [blue] at x [50] y [32] inverse [blue]
  end
  if [OLED_color = 4] then
    clear
    write [white] at x [45] y [32] inverse [white]
  end
  wait (20) millisecs
end

when started
  initialize i2c [OLED_0.96in v] address(hex) [3C] reset pin# [0] flip [red]
  PicoBricks turn off RGB LED
  clear
  write [The game begins] at x [10] y [30] inverse [red]
  wait (2000) millisecs
  repeat (10)
    repeat (10)
      random_color
      wait (1500) millisecs
    clear
  PicoBricks turn off RGB LED
  end
  end

```

At this stage, when the colors on the OLED screen and RGB LED are the same, create two more variables called Score and x for the user to press the button and earn points. Add the Picobricks button pressing block as a condition to the when block. Then compare the two if blocks to see if the color variables are equal. If the colors are the same, increase the score variable by 10, if not, decrease it by 10. The reason why we use the x variable here is to increase the score only once. If you do not use the variable x as the second condition in the and operator, the Score variable will increase by 10 momentarily while the button is pressed.



```

when PicoBricks button
  if OLED_color = RGB_color and x = 0
    change Score by 10
    set x to 1
  else
    if OLED_color ≠ RGB_color and x = 0
      change Score by -10
      set x to 1

```

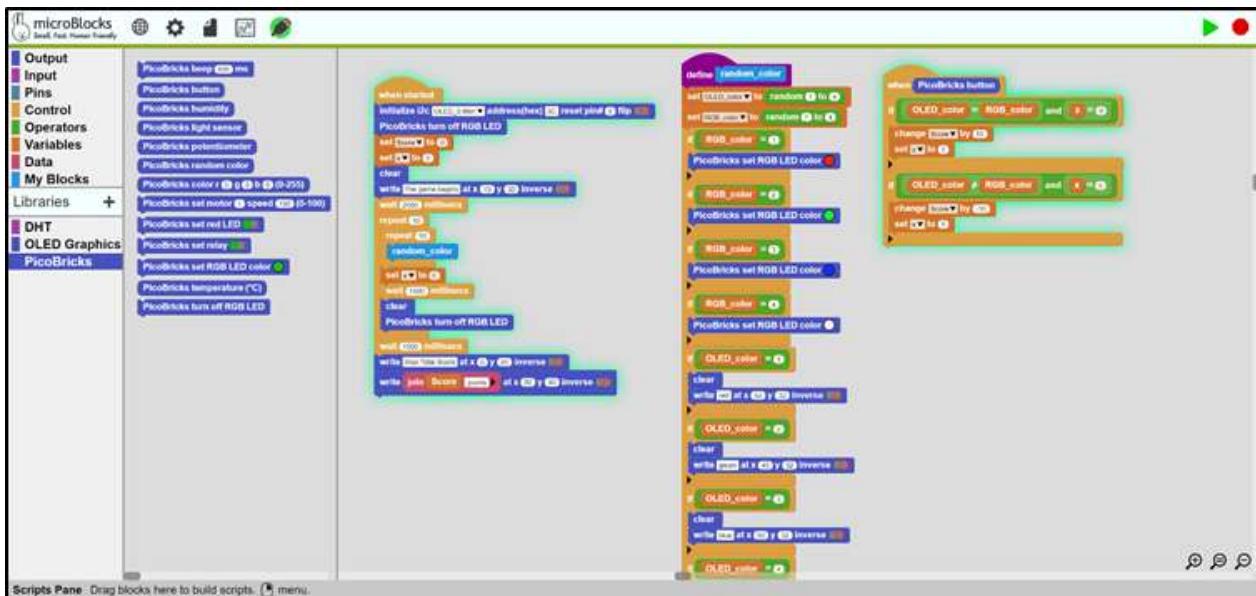
After arranging the actions that the program should do when the button is pressed, when the game is over, you must add the necessary codes to the main program for the user's score to be written on the screen. Specify the initial values of the score and x variables as "0" and assign the value of the x variable to "0" at each iteration within 10 iterations. In this way, each time the button is pressed in the game, a one-time score increase will occur.

```

when started
  initialize i2c OLED_0.96in address(hex) 3C reset pin# 0 flip
  PicoBricks turn off RGB LED
  set Score to 0
  set x to 0
  clear
  write The game begins at x 10 y 30 inverse
  wait 2000 millisecs
  repeat (10)
    repeat (10)
      random_color
      set x to 0
      wait 1500 millisecs
      clear
      PicoBricks turn off RGB LED
      wait 1500 millisecs
      write Your Total Score at x 0 y 20 inverse
      write join Score points at x 30 y 40 inverse

```

After completing the code writing process, start the game with the Start button and test it. If everything went well, random colors will appear on the OLED screen and RGB LED 10 times at 1.5 second intervals. When the colors are the same, if the button is pressed, the score will increase by 10 points and when the game is over, the score will be displayed on the screen.



[Click](#) to access the project's MicroBlocks codes.

2.10.6. MicroPython Codes of the Project

```

from machine import Pin, I2C
from picobricks import SSD1306_I2C
import utime
import urandom
import _thread
from picobricks import WS2812

WIDTH = 128
HEIGHT = 64
sda=machine.Pin(4)
scl=machine.Pin(5)
i2c=machine.I2C(0,sda=sda, scl=scl, freq=1000000)
ws = WS2812(pin_num=6, num_leds=1, brightness=0.3)

oled = SSD1306_I2C(WIDTH, HEIGHT, i2c)

button = Pin(10,Pin.IN,Pin.PULL_DOWN)
RED = (255, 0, 0)
GREEN = (0, 255, 0)
BLUE = (0, 0, 255)
WHITE = (255, 255, 255)

```



BLACK = (0, 0, 0)

```
oled.fill(0)  
oled.show()
```

```
ws.pixels_fill(BLACK)  
ws.pixels_show()
```

```
global button_pressed  
score=0  
button_pressed = False
```

```
def random_rgb():  
    global ledcolor  
    ledcolor=int(urandom.uniform(1,4))  
    if ledcolor == 1:  
        ws.pixels_fill(RED)  
        ws.pixels_show()  
    elif ledcolor == 2:  
        ws.pixels_fill(GREEN)  
        ws.pixels_show()  
    elif ledcolor == 3:  
        ws.pixels_fill(BLUE)  
        ws.pixels_show()  
    elif ledcolor == 4:  
        ws.pixels_fill(WHITE)  
        ws.pixels_show()
```

```
def random_text():  
    global oledtext  
    oledtext=int(urandom.uniform(1,4))  
    if oledtext == 1:  
        oled.fill(0)  
        oled.show()  
        oled.text("RED",45,32)  
        oled.show()  
    elif oledtext == 2:  
        oled.fill(0)  
        oled.show()  
        oled.text("GREEN",45,32)  
        oled.show()
```



```
elif oledtext == 3:  
    oled.fill(0)  
    oled.show()  
    oled.text("BLUE",45,32)  
    oled.show()  
elif oledtext == 4:  
    oled.fill(0)  
    oled.show()  
    oled.text("WHITE",45,32)  
    oled.show()  
  
def button_reader_thread():  
    while True:  
        global button_pressed  
        if button_pressed == False:  
            if button.value() == 1:  
                button_pressed = True  
                global score  
                global oledtext  
                global ledcolor  
                if ledcolor == oledtext:  
                    score += 10  
                else:  
                    score -= 10  
                utime.sleep(0.01)  
  
    _thread.start_new_thread(button_reader_thread, ())  
  
oled.text("The Game Begins",0,10)  
oled.show()  
utime.sleep(2)  
  
for i in range(10):  
    random_text()  
    random_rgb()  
    button_pressed=False  
    utime.sleep(1.5)  
    oled.fill(0)  
    oled.show()  
    ws.pixels_fill(BLACK)  
    ws.pixels_show()
```



```
utime.sleep(1.5)
oled.fill(0)
oled.show()
oled.text("Your total score:",0,20)
oled.text(str(score), 30,40)
oled.show()
```

2.10.7. Arduino C Codes of the Project

```
#include <Adafruit_NeoPixel.h>
#define PIN      6
#define NUMPIXELS 1
Adafruit_NeoPixel pixels(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);
#define DELAYVAL 500
#include <Wire.h>
#include "ACROBOTIC_SSD1306.h" //define libraries
int OLED_color;
int RGB_color;
int score = 0;
int button = 0;

void setup() {
    // put your setup code here, to run once:
    Wire.begin();
    oled.init();
    oled.clearDisplay();

    pixels.begin();
    pixels.clear();
    randomSeed(analogRead(27));

}

void loop() {
    // put your main code here, to run repeatedly:
    oled.clearDisplay();
    oled.setTextXY(3,1);
```



```
oled.putString("The game begins");
pixels.setPixelColor(0, pixels.Color(0, 0, 0));
pixels.show();
delay(2000);
oled.clearDisplay();

for (int i=0;i<10;i++){
    button = digitalRead(10);
    random_color();
    pixels.show();
    unsigned long start_time = millis();
    while (button == 0) {
        button = digitalRead(10);
        if (millis() - start_time > 2000)
            break;
    }
    if (button == 1){

        if(OLED_color==RGB_color){
            score=score+10;
        }
        if(OLED_color!=RGB_color){
            score=score-10;
        }
        delay(200);
    }
    oled.clearDisplay();
    pixels.setPixelColor(0, pixels.Color(0, 0, 0));
    pixels.show();
}

String string_scrore=String(score);
oled.clearDisplay();
oled.setTextXY(2,5);
oled.putString("Score: ");
oled.setTextXY(4,7);
oled.putString(string_scrore);
oled.setTextXY(6,5);
oled.putString("points");
// print final score on OLED screen
```



```
delay(10000);
}

void random_color(){

OLED_color = random(1,5);
RGB_color = random(1,5);
// generate numbers between 1 and 5 randomly and print them on the screen
if (OLED_color == 1){
    oled.setTextXY(3,7);
    oled.putString("red");
}
if (OLED_color == 2){
    oled.setTextXY(3,6);
    oled.putString("green");
}
if (OLED_color == 3){
    oled.setTextXY(3,6);
    oled.putString("blue");
}
if (OLED_color == 4){
    oled.setTextXY(3,6);
    oled.putString("white");
}
if (RGB_color == 1){
    pixels.setPixelColor(0, pixels.Color(255, 0, 0));
}
if (RGB_color == 2){
    pixels.setPixelColor(0, pixels.Color(0, 255, 0));
}
if (RGB_color == 3){
    pixels.setPixelColor(0, pixels.Color(0, 0, 255));
}
if (RGB_color == 4){
    pixels.setPixelColor(0, pixels.Color(255, 255, 255));
}
}
```



GitHub Know Your Color Project Page



<http://rbt.ist/color>



2.11. Magic Lamp

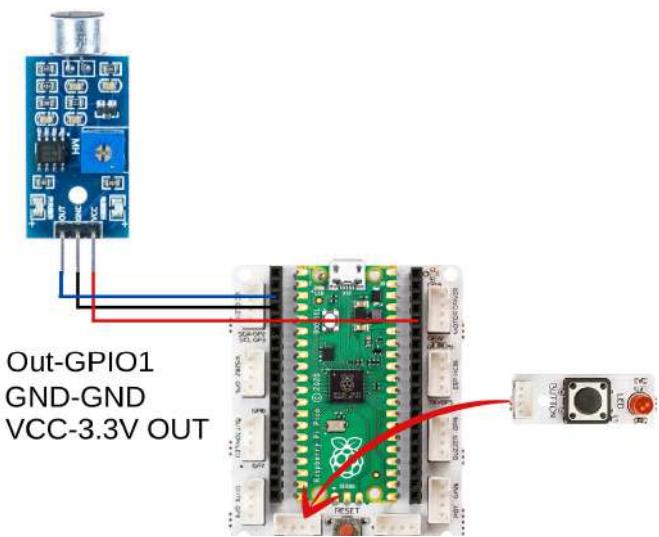
Project Author: Abdullah KAYA

Most of us have seen lamps flashing magically or doors opening and closing with the sound of clapping in movies. There are set assistants who close these doors and turn off the lamps in the shootings. What if we did this automatically? There are sensors that convert the sound intensity change that we expect to occur in the environment into an electrical signal. These are called sound sensors.

2.12.1. Project Details and Algorithm

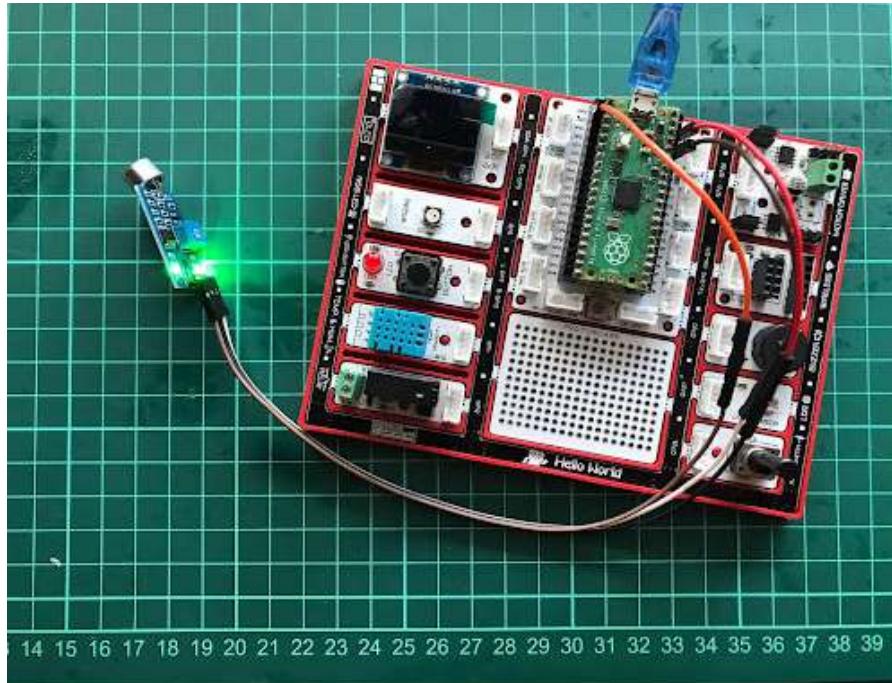
In this project, we will turn the LED module on the picobricks board on and off with the sound. In our project, which we will build using the Picobricks sound level sensor, we will perform the on-off operations by making a clap sound. As in previous projects, in projects where sensors are used, before we start to write the codes, it will make your progress easier to see what values the sensor sends in the operations we want to do by just running the sensor, and then writing the codes of the project based on these values.

2.12.2. Wiring Diagram



2.11.3. Construction Stages of the Project

During the construction of the project, two wire sockets and sockets were used. The two ends, which were cut by cutting the phase cable, were connected to the relay. You should pay attention to the insulation with electrical tape so that a dangerous situation does not occur when you cut the other wire. If you use a three-wire socket, you must cut the brown wire with the phase lead and connect it to the relay.



2.11.4. Project Proposal

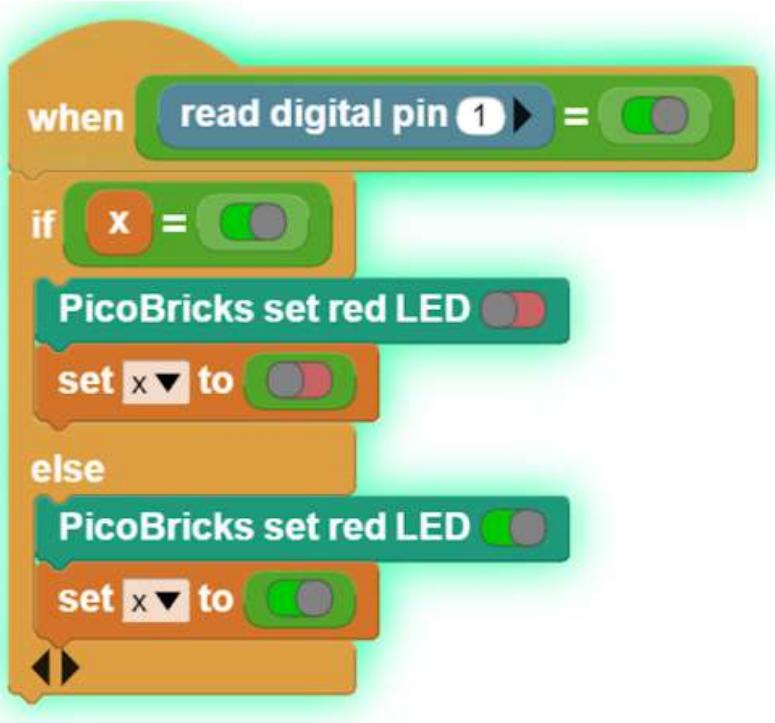
You can present the player with instructions and notifications on the OLED screen. In addition, you can prepare a more exciting game by showing on the OLED screen how many milliseconds after the game starts, the game is over.

2.11.5. Coding the Project with MicroBlocks

While writing the Microblocks codes, we first create a variable called x and set the initial value of both the variable and the relay as false. We will use the x variable to turn the lamp off if it is on and on if it is off.



The digital sound sensor sends us a value of “0” when sound is detected through the digital pin it is connected to, and “1” when idle. MicroBlocks takes these values as “True” and “False”. In the codes we wrote, if the value coming from digital pin 16 is “0”, that is “False”, the relay will be activated. When the relay is activated, the value of the x variable will be checked, if x is False, the relay will turn on and the value of the x variable will be changed to “True”, otherwise the relay will be closed and the value of the x variable will be assigned as “False” again. These codes will make the lamp turn on when we clap and turn off the lamp when we clap again.



[Click](#) to access the project's MicroBlock codes.

2.11.6. MicroPython Codes of the Project

```

from machine import Pin
sensor=Pin(16,Pin.IN)
relay=Pin(12,Pin.OUT)
x=0
while True:
  if sensor.value()==0:
    if x==0:
      relay.value(1)
      x=1
    else:
      relay.value(0)
      x=0
  
```

2.11.7. Arduino C Codes of the Project

```
void setup() {
```



```
// put your setup code here, to run once:  
pinMode(1,INPUT);  
pinMode(7,OUTPUT);  
//define the input and output pins  
}  
  
void loop() {  
    // put your main code here, to run repeatedly:  
  
    Serial.println(digitalRead(1));  
  
    if(digitalRead(1)==1){  
        digitalWrite(7,HIGH);  
        delay(3000);  
    }  
    else{  
        digitalWrite(7,LOW);  
        delay(1000);  
    }  
}
```

GitHub Magic Lamp Project Page



<http://rbt.ist/lamp>



2.12. Smart Cooler

Project Author: Abdullah KAYA

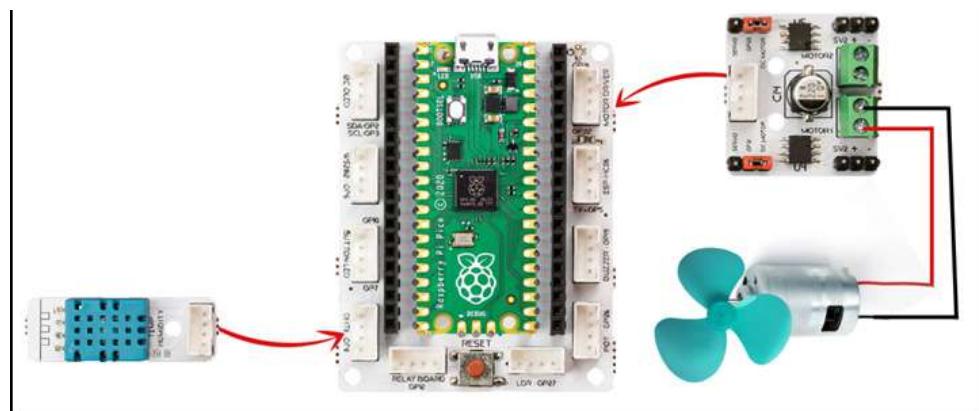
Air conditioners are used to cool in the summer and warm up in the winter. Air conditioners adjust the degree of heating and cooling according to the temperature of the environment. While cooking the food, the ovens try to rise to the temperature value set by the user and maintain that temperature. These two electronic devices use special temperature sensors to control the temperature. In addition, temperature and humidity are measured together in greenhouses. In order to keep these two values in balance at the desired level, it is tried to provide air flow with the fan.

In Picobricks, you can measure temperature and humidity separately and interact with the environment with these measurements. In this project, we will prepare a cooling system that automatically adjusts the fan speed according to the temperature with Picobricks. In this way, you will learn the DC motor operating system and motor speed adjustment.

2.12.1. Project Details and Algorithm

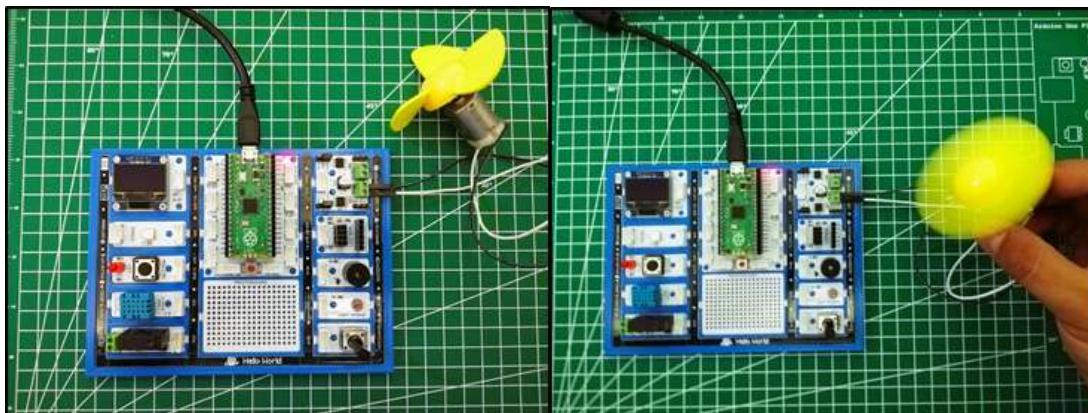
In our project, we will firstly display the temperature values measured by the DHT11 temperature and humidity sensor on Picobricks. Then, we will define a temperature limit and write the necessary codes for the DC motor connected to Picobricks to start rotating when the temperature value from the DHT11 module reaches this limit, and for the DC motor to stop when the temperature value falls below the limit we have determined.

2.12.2. Wiring Diagram





2.12.3. Project Image



2.12.4. Project Proposal

Using the OLED screen on Picobricks, you can print the temperature on the screen and keep track the temperature at which the fan is activated.

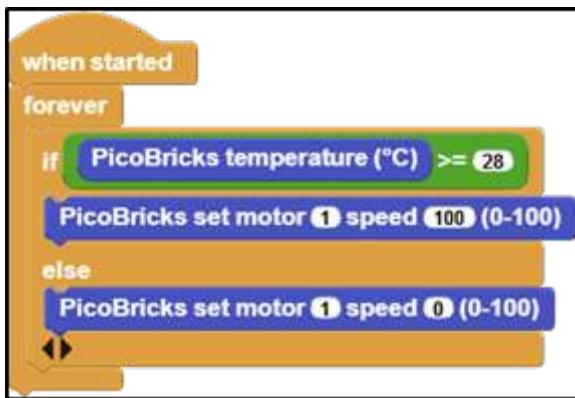
Picobricks has a modular structure, modules can be separated by breaking and can be used by connecting to Pico board with grove cables. By mounting the smart cooling circuit we made in our project to the robot car chassis, you can develop a project that navigates autonomously in your environment and cools the environment at the same time.

2.12.5. Coding the Project with MicroBlocks



In order to decide when the fan will start and work, we first need to see the values coming from the DHT11 sensor and act according to these values. You can use the say123 block in the Output category for this. Then drag and drop the PicoBricks temperature block in the Picobricks category to the circle that 123 in the say block. See the values from the sensor by pressing the start button. You should see values around 25 degrees at room temperature. Hold for a while by touching the DHT11 module on the Picobricks. You will see that the temperature value from the sensor increases due to the heat on the finger of the DHT11. After you see the values, you can delete the say and temperature blocks.

After you have determined a temperature value for the fan to activate according to your environment, you can carry out the project by using the Picobricks set motor block in the if else block. You can change the fan speed between 0 and 100.



[Click](#) to access the project's MicroBlock codes.

2.12.6. MicroPython Codes of the Project

Codes that print to the shell window of the current temperature:

```
from machine import Pin
from dht import DHT11
from utime import sleep
dht_sensor = DHT11(11)

while True:
    sleep(1) # It was used for DHT11 to measure.
    dht_sensor.measure() # Use the sleep() command before this line.
    temp=dht_sensor.temperature()
    print(temp)
```

Project Codes:

```
from machine import Pin
from picobricks import DHT11
import utime

LIMIT_TEMPERATURE = 20 #define the limit temperature

dht_sensor = DHT11(Pin(11, Pin.IN, Pin.PULL_DOWN))
m1 = Pin(21, Pin.OUT)
m1.low()
dht_read_time = utime.time()
```



```
#define input-output pins

while True:
    if utime.time() - dht_read_time >= 3:
        dht_read_time = utime.time()
        dht_sensor.measure()
        temp= dht_sensor.temperature
        print(temp)
        if temp >= LIMIT_TEMPERATURE:
            m1.high()
            #operate if the room temperature is higher than the limit temperature
        else:
            m1.low()
```

2.12.7. Arduino C Codes of the Project

```
#include <DHT.h>

#define LIMIT_TEMPERATURE 27
#define DHTPIN 11
#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);
float temperature;

void setup() {
    // put your setup code here, to run once:
    Serial.begin(115200);
    dht.begin();
    pinMode(21,OUTPUT);

}

void loop() {
    // put your main code here, to run repeatedly:
    delay(100);
    temperature = dht.readTemperature();
    Serial.print("Temp: ");
    Serial.println(temperature);
    if(temperature > LIMIT_TEMPERATURE){
```



```
digitalWrite(21,HIGH);  
} else{  
    digitalWrite(21,LOW);  
}  
  
}
```

GitHub Smart Cooler Project Page



<http://rbt.ist/coolerm>



2.13. Buzz Wire Game

Projects don't always have to be about solving problems and making things easier. You can also prepare projects to have fun and develop yourself. Attention and concentration are features that many people want to develop. The applications that we can do with this are quite interesting. How about making Buzz Wire Game with Picobricks?

You must have heard the expression that computers work with 0s and 1s. 0 represents the absence of electricity and 1 represents its presence. 0 and 1's come together with a certain number and sequence of combinations to form meaningful data. In electronic systems, 0s and 1s can be used to directly control a situation. Is the door closed or not? Is the light on or off? Is the irrigation system on or not? In order to obtain such information, a status check is carried out.

In this project, we will electronically prepare the attention and concentration developer Buzz Wire Game with the help of a conductor wire using the buzzer and LED module with Picobricks. While preparing this project, you will have learned an input technique that is not a button but will be used like a button.

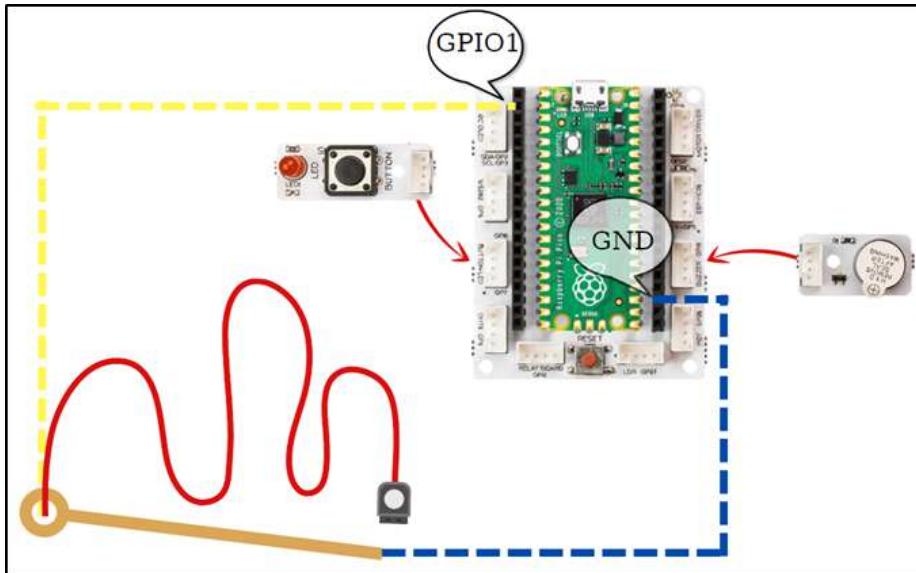
2.13.1. Project Details and Algorithm

To prepare the project, you need 2 male-male jumper cables and a 15 cm long conductor bendable wire. When the player is ready, it will be asked to press the button to start the game. If the jumper cable touches the conductor wire in the player's hand when the button is pressed, Picobricks will detect this and give an audible and written warning. The time from the start of the game to the end will also be displayed on the OLED screen.

We reset the timer after the user presses the button. Then we will give a voltage of 3.3V to the conductor wire connected to the GPIO1 pin of Picobricks. One end of the cable held by the player will be connected to the GND pin on the Picobricks. If the player touches the jumper cable in his hand to the conductive wire, the GPIO1 pin will drop to the Passive/Off/0 position. Then, it will announce that the game is over, and there will be light, written and audio feedback, then the elapsed time will be shown on the OLED screen in milliseconds. After 5 seconds, the player will be prompted to press the button to restart.



2.13.2. Wiring Diagram



2.13.3. Project Proposal

You can make physical and software improvements to the project. By covering the start and end points with insulating tape, you can prevent the player from having problems starting and finishing the game. In terms of software, when the player brings the cable to the other end without touching the wire, press the button and you can see the score on the OLED screen.

2.13.4. Coding the Project with MicroBlocks

When PicoBricks starts, we open an endless loop immediately after the OLED screen starts. Because when the game is over, it will return to the beginning. Inside the endless loop, we firstly turn off the red LED and place the start message expressions on the OLED screen. Then we place the wait until block from the Control category and wait for the loop until Picobricks' button is clicked.

After the button is pressed, we add an expression to the OLED screen that the game starts in the continuation of the codes. Then we open the GPIO1 pin and give a voltage of 3.3V. After resetting the timer, we wait until the GPIO1 pin 0 is closed in the wait until block. Here, the state of the player touching the wire that is in player's hand to the wire is checked. When touch is detected, the cycle continues from where it left off and the necessary illuminated, written and audible notifications are presented. Finally, wait 5 seconds and return to the beginning of the loop.



```

when started
  initialize i2c OLED_0.96in address(hex) 3C reset pin# 0 flip ⚪
forever
  PicoBricks set red LED ⚪
  write <BUZZ WIRE GAME> at x 0 y 0 inverse ⚪
  write Press the Button at x 0 y 17 inverse ⚪
  write TO START! at x 25 y 35 inverse ⚪
  wait until PicoBricks button = ⚪
  clear
  write GAME at x 25 y 35 inverse ⚪
  write STARTED! at x 25 y 45 inverse ⚪
  set digital pin 1 to ⚪
  reset timer
  wait until read digital pin 1 = ⚪
  clear
  set time ▾ to timer
  say time ▶
  write GAME OVER! at x 25 y 35 inverse ⚪
  write join time ms ▶ at x 25 y 45 inverse ⚪
  PicoBricks set red LED ⚪
  PicoBricks beep 500 ms
  wait 5000 millisecs
  clear

```

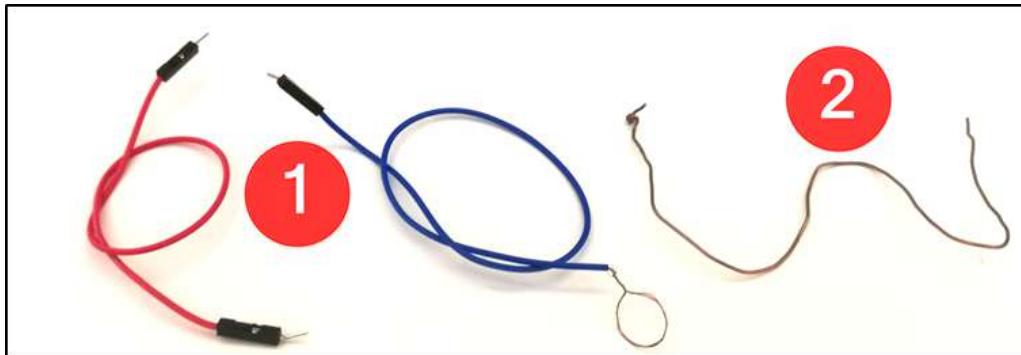
[Click](#) to access the codes of the project.

2.13.5. Construction Stages of the Project

Along with the PicoBricks base kit,

1: 2 20 cm male-male jumper cables. One end of the cable to be attached to the GND will be stripped 4-5 cm and made into a ring.

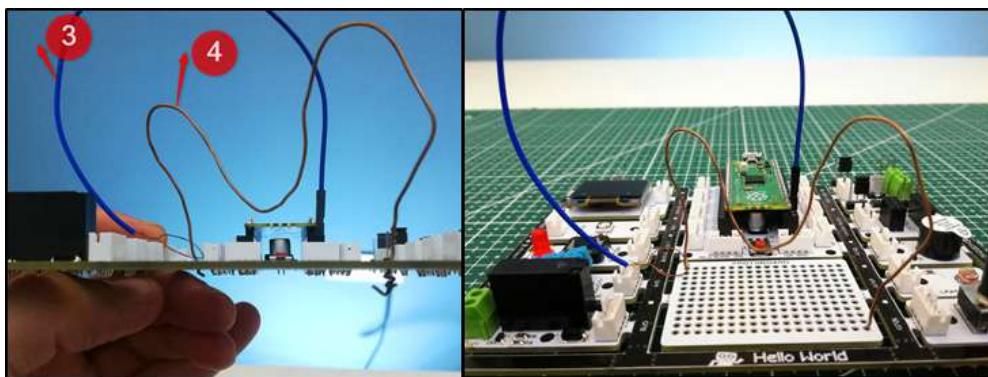
2: 15-20 cm conductive wire with a thickness of 0.8 mm. Prepare your materials.



Bend the conductor wire on the protoboard as you wish and pass it through the holes, before passing one end, you must pass the male end, which is connected to the GND pin on the PicoBoard, the other end of the cable you have made into a ring.

3: Conductor Wire

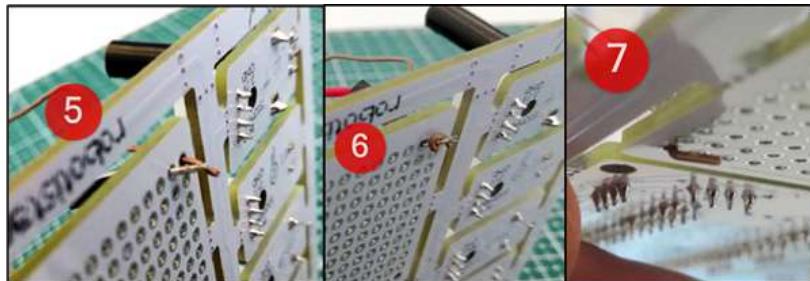
4: Jumper cable with one end connected to the GND pin with a looped end.



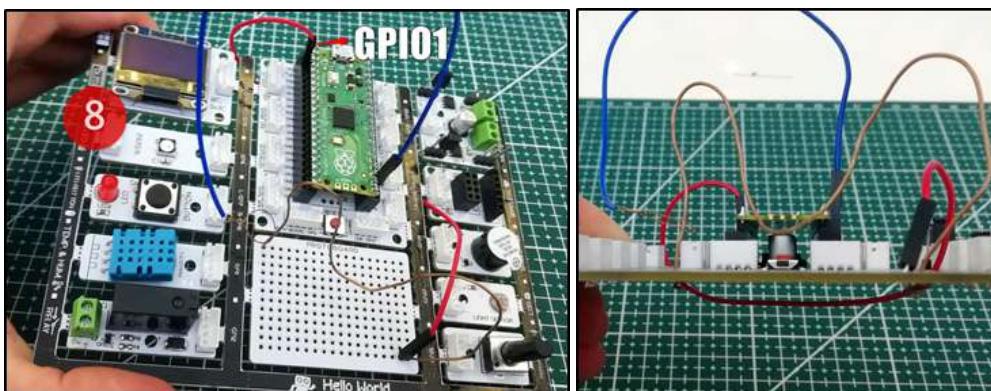
5: One end of the jumper cable, which has both male ends, into the hole right next to the end of the conductive wire you placed on the protoboard

6: Twist the end of the jumper wire and the end of the conductor wire together under the protoboard.

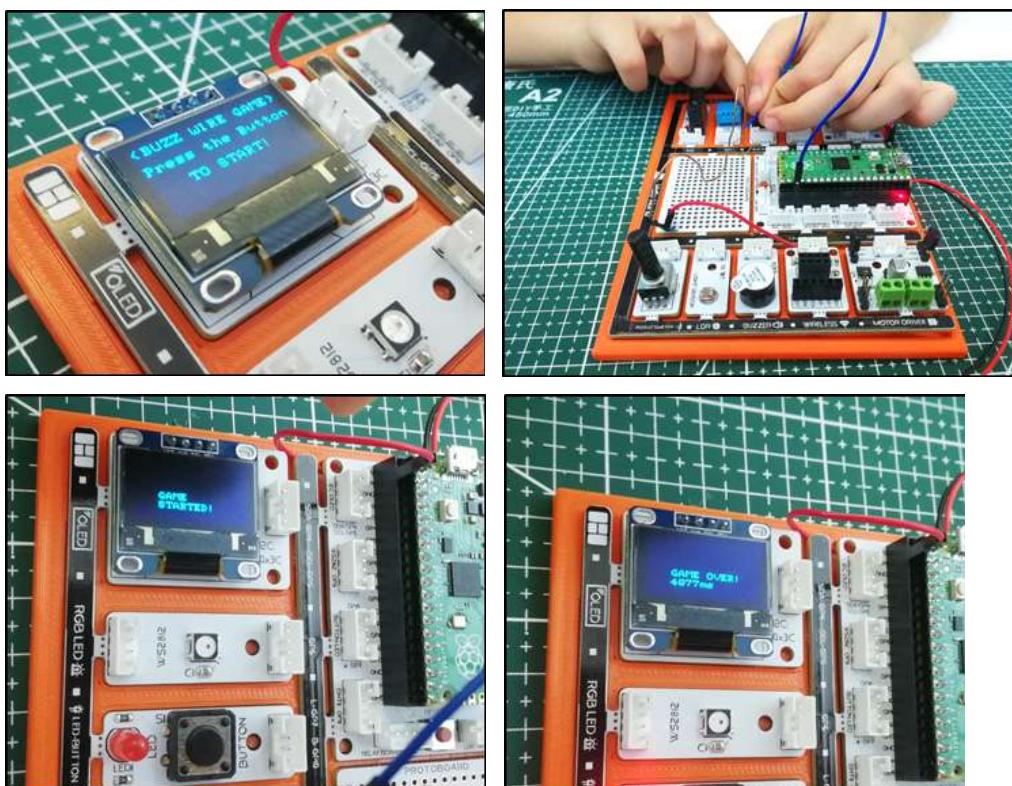
7: Bend the other end of the conductor wire placed on the protoboard so that it does not come out.



8: Connect the other male end of the jumper cable that you wrapped around the end of the conductor wire in step 6 to the pin no. GPIO1 on the Picoboard



If you have completed the installation, you can start the game after installing the codes. Have fun. :)



2.13.6. MicroPython Codes of the Project

```
from machine import Pin, I2C, Timer #to access the hardware on the pico
from picobricks import SSD1306_I2C #OLED Screen Library
from utime import sleep # time library
```

```
#OLED Screen Settings
WIDTH = 128
```



HEIGHT = 64

```
sda=machine.Pin(4)#initialize digital pin 4 and 5 as an OUTPUT for OLED
Communication
scl=machine.Pin(5)
i2c=machine.I2C(0,sda=sda, scl=scl, freq=1000000)
oled = SSD1306_I2C(WIDTH, HEIGHT, i2c)

wire=Pin(1,Pin.OUT)#initialize digital pin 1 as an OUTPUT
led = Pin(7,Pin.OUT)#initialize digital pin 7 and 5 as an OUTPUT for LED
buzzer=Pin(20, Pin.OUT)#initialize digital pin 20 as an OUTPUT for Buzzer
button=Pin(10,Pin.IN,Pin.PULL_DOWN)#initialize digital pin 10 as an INPUT for button
endtime=0
```

while True:

```
    led.low()
    oled.fill(0)
    oled.show()
    oled.text("<BUZZ WIRE GAME>",0,0)
    oled.text("Press the button",0,17)
    oled.text("TO START!",25,35)
    oled.show()
    #When button is '0', OLED says 'GAME STARTED'
    while button.value()==0:
        print("press the button")
        oled.fill(0)
        oled.show()
        oled.text("GAME",25,35)
        oled.text("STARTED",25,45)
        oled.show()
        wire.high()
        timer_start=utime.ticks_ms()
        #When wire is '1', OLED says 'GAME OVER'
        while wire.value()==1:
            print("Started")
            endtime=utime.ticks_diff(utime.ticks_ms(), timer_start)
            print(endtime)
            oled.fill(0)
            oled.show()
            oled.text("GAME OVER!",25,35)
            oled.text(endtime + " ms" ,25,45)
```



```
oled.show()
led.high()#LED On
buzzer.high()#Buzzer On
sleep(5)#Delay
```

2.13.7. Arduino C Codes of the Project

```
#include <Wire.h>
#include "ACROBOTIC_SSD1306.h"

int Time=0;
unsigned long Old_Time=0;

void setup() {

    pinMode(20,OUTPUT);
    pinMode(7,OUTPUT);
    pinMode(1,OUTPUT);
    pinMode(10,INPUT);

    Wire.begin();
    oled.init();
    oled.clearDisplay();

#if defined(__AVR_ATtiny85__) && (F_CPU == 16000000)
    clock_prescale_set(clock_div_1);
#endif
}

void loop() {

    digitalWrite(7,LOW);

    oled.setTextXY(2,1);
    oled.putString("BUZZ WIRE GAME");
    oled.setTextXY(4,2);
    oled.putString("Press Button");
    oled.setTextXY(5,3);
    oled.putString("TO START!");
```



```
while (!(digitalRead(10)==1)){  
  
}  
  
oled.clearDisplay();  
oled.setTextXY(3,6);  
oled.putString("GAME");  
oled.setTextXY(5,4);  
oled.putString("STARTED");  
  
digitalWrite(1,HIGH);  
Old_Time=millis();  
  
while(!(digitalRead(1)==0)){  
  
    Time=millis()-Old_Time;  
}  
  
String(String_Time)=String(Time);  
  
oled.clearDisplay();  
oled.setTextXY(3,4);  
oled.putString("GAME OVER");  
oled.setTextXY(5,4);  
oled.putString(String_Time);  
oled.setTextXY(5,10);  
oled.putString("ms");  
  
digitalWrite(7,HIGH);  
digitalWrite(20,HIGH);  
delay(500);  
digitalWrite(20,LOW);  
delay(5000);  
  
Time=0;  
Old_Time=0;  
oled.clearDisplay();  
}
```



GitHub Buzz Wire Game Project Page



<http://rbt.ist/buzzwire>



2.14. Dinosaur Game

If the electronic systems to be developed will fulfill their duties by pushing, pulling, turning, lifting, lowering, etc., pneumatic systems or electric motor systems are used as actuators in the project. Picobricks supports two different engine types so that you can produce systems that can activate the codes you write in your projects. DC motor and Servo motors in which the movements of DC motors are regulated electronically. Servo motors are motors that rotate to that angle when the rotation angle value is given. In RC boats, servo motors are used with the same logic to change the direction of the vehicle. In addition, advanced servo motors known as smart continuous servos, which can rotate full-round, are also used in the wheels of the smart vacuum cleaners we use in our homes.

In this project you will learn how to control Servo motors with PicoBricks.

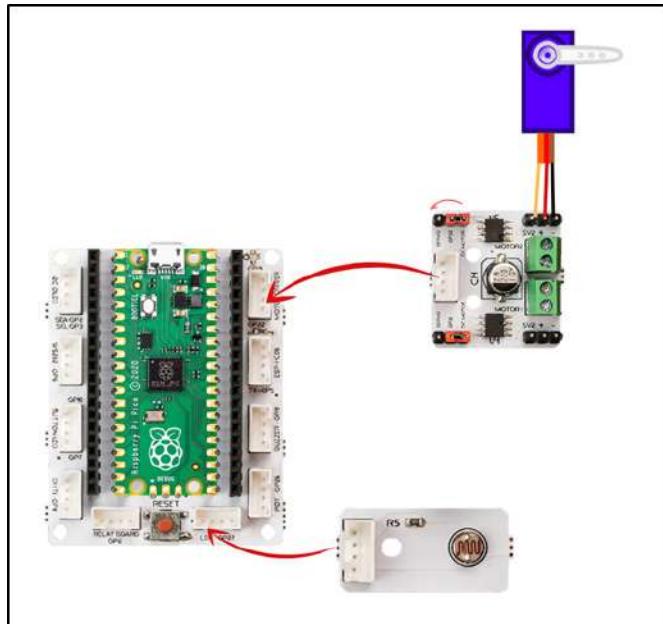
2.14.1. Project Details and Algorithm

While writing the project codes, we will first fix the LDR sensor on the computer screen and read the sensor data on the white and black background, then write the necessary codes for the servo motor to move according to these data. In this project, we will automatically play Google Chrome offline dinasour game to picobricks. In the game, Picobricks will automatically control the dinosaur's movements by detecting obstacles. We will use the picobricks LDR sensor to detect the obstacles in front of the dinosaur during the game. LDR can send analog signals by measuring the amount of light touching the sensor surface. By fixing the sensor on the computer screen, we can detect if there is an obstacle in front of the dinosaur by taking advantage of the difference in the amount of light between the white and black colors. When an obstacle is detected, we can use a servo motor to automatically press the spacebar on the keyboard. In this way, the dinosaur will easily overcome the obstacles. While writing the project codes, we will firstly fix the LDR sensor on the computer screen and read the sensor data on the white and black background, then write the necessary codes for the servo motor to move according to these data.

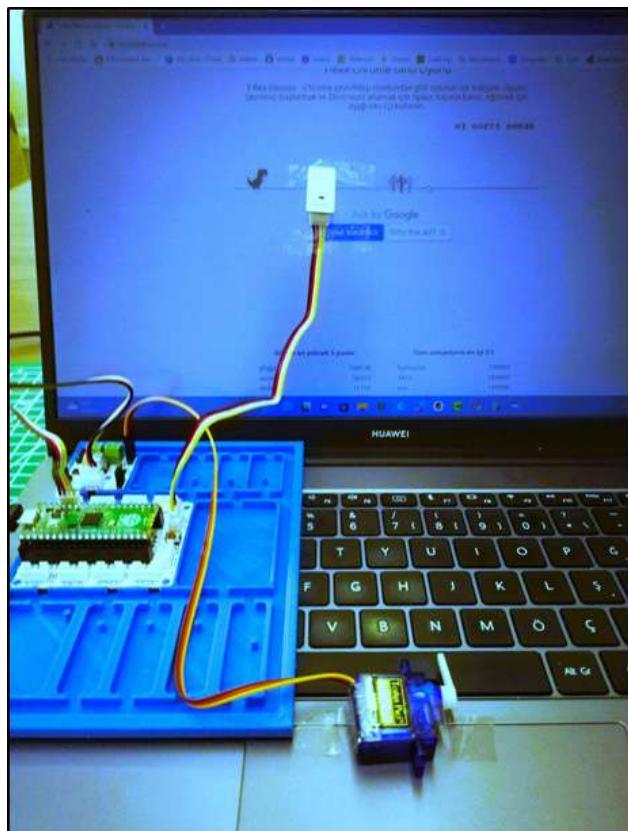


2.14.2. Wiring Diagram

Note: There are triple pins on the right and left side of the motor driver grove cable entry and these pins are short-circuited with 2 jumpers. When using a DC motor, the jumper that should be attached on the DC motor side should be removed when using a servo motor and attached to the servo side.



2.14.3. Project Image





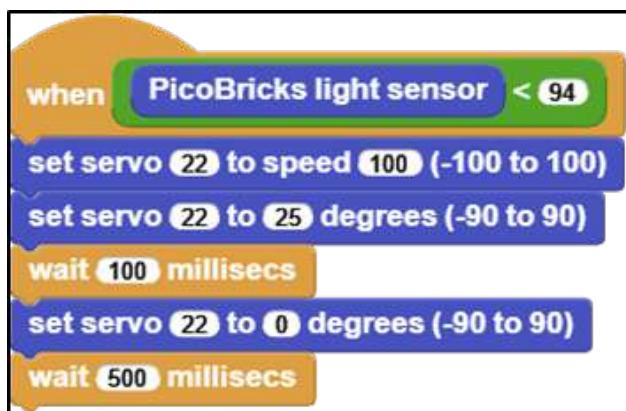
2.14.4. Project Proposal

At first in the game, the ground color is white and the figures are black. After a certain stage, the colors are reversed. For this reason, LDR sensor data is changing. To solve this problem, you can use variables and functions to run one code group when the game is on a white background, another code group when it is on a black background, or you can install a second LDR sensor to detect this difference.

Picobricks and its modules allow us to develop many projects from simple to complex. You can also use it in different games such as minecraft by developing this project, which we automatically play a computer game that we play in daily life on Picobricks.

2.14.5. Coding the Project with MicroBlocks

For the project to work, you must first read the LDR sensor values that will change according to your environment. You can use the say123 block for this. Open Chrome offline dinosaur game. Fix the sensor 3-4 cm to the right from the dinosaur and just above the road line with the help of tape. After making sure that the sensor touches the screen, read the sensor values. The values when on the white ground will be different from the values when the obstacle comes. Convert the limit value, which you will determine by observing the difference, to the code with the when block. When the LDR sensor value is less than the value you set, write the necessary codes to change the angle of the servo by 25 degrees and return to its original position, and fix the servo motor on your keyboard so that it automatically presses the space key.



[Click](#) to access the codes of the project.

2.14.6. MicroPython Codes of the Project

```

from machine import Pin, ADC, PWM #to access the hardware on the pico
from utime import sleep #time library
  
```

ldr=ADC(27) #initialize digital pin 27 for LDR



```
servo=PWM(Pin(21)) #initialize digital PWM pin 27 for Servo Motor  
servo.freq(50)  
  
while True:      #When LDR data higher than 40000  
    sleep(0.01)  
    if ldr.read_u16()>4000:  
        servo.duty_u16(2000) #sets position to 180 degrees  
        sleep(0.1)          #delay  
        servo.duty_u16(1350) #sets position to 0 degrees  
        sleep(0.5)          #delay
```

2.14.7. Arduino C Codes of the Project

```
#include <Servo.h>  
Servo myservo;  
  
void setup() {  
    myservo.attach(22);  
    myservo.write(20);  
    pinMode(27,INPUT);  
}  
  
void loop() {      // put your main code here, to run repeatedly:  
  
    int light_sensor=analogRead(27);  
  
    if(light_sensor>100){  
  
        int x=45;  
        int y=20;  
  
        myservo.write(x);  
        delay(100);  
        myservo.write(y);  
        delay(500);  
    }  
}
```



GitHub Dinosaur Game Project Page



<http://rbt.ist/dinosaur>



2.15. Night and Day

How about playing the Night and Day game you played at school electronically? The game of night and day is a game in which we put our head on the table when our teacher says night, and raise our heads when our teacher says day. This game will be a game that you will use your attention and reflex. In this project, we will use a 0.96" 128x64 pixel I2C OLED display. Since OLED screens can be used as an artificial light source, you can enlarge the characters on the screen using lenses and mirrors and reflect them on the desired plane. Systems that can reflect information, road and traffic information on smart glasses and automobile windows can be made using OLED screens.

Light sensors are sensors that can measure the light levels of the environment they are in, also called photodiodes. The electrical conductivity of the sensor exposed to light changes. We can control the light sensor by coding and develop electronic systems that affect the amount of light.

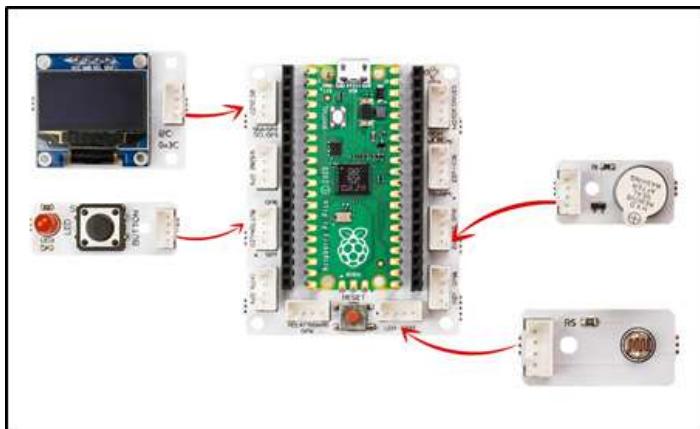


2.15.1. Project Details and Algorithm

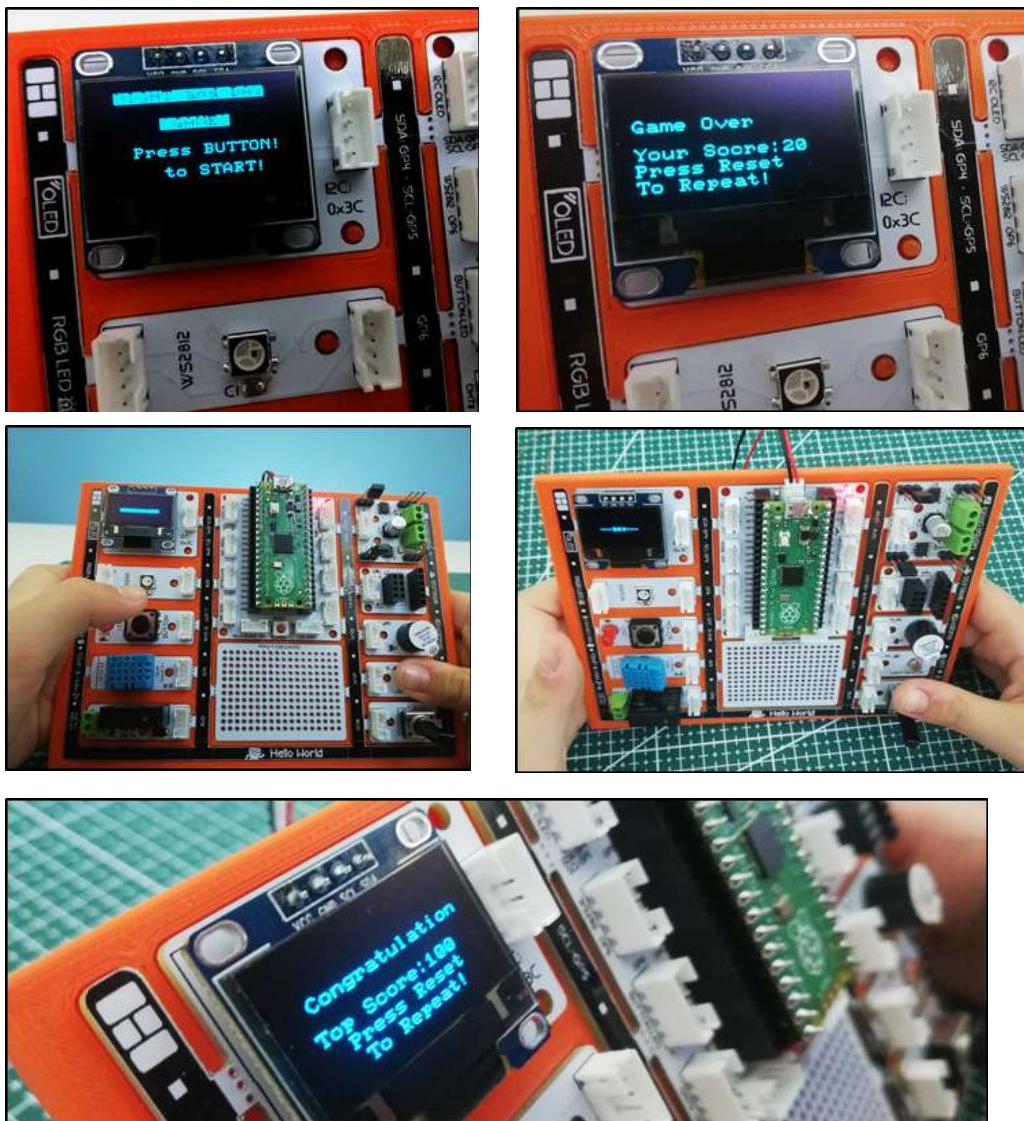
First we will ask the player to press the button to start the game. Then we will make the OLED screen of PicoBricks display NIGHT and DAY randomly for 2 seconds each. The player should cover the LDR sensor with his hand within 2 seconds if the word written on the OLED screen is NIGHT, and if the word DAY is written on the OLED screen, the player should raise his hand over the LDR sensor. Each correct response of the player will earn 10 points. In case of wrong response, the game will be over and there will be a written statement on the screen stating the end of the game, a different tone will sound from the buzzer, and the score information will be displayed on the OLED screen. If the player gives a total of 10 correct responses and gets 100 points, the phrase "Congratulation" will be displayed on the OLED screen and the buzzer will play notes in different tones.



2.15.2. Wiring Diagram



2.15.3. Project Image



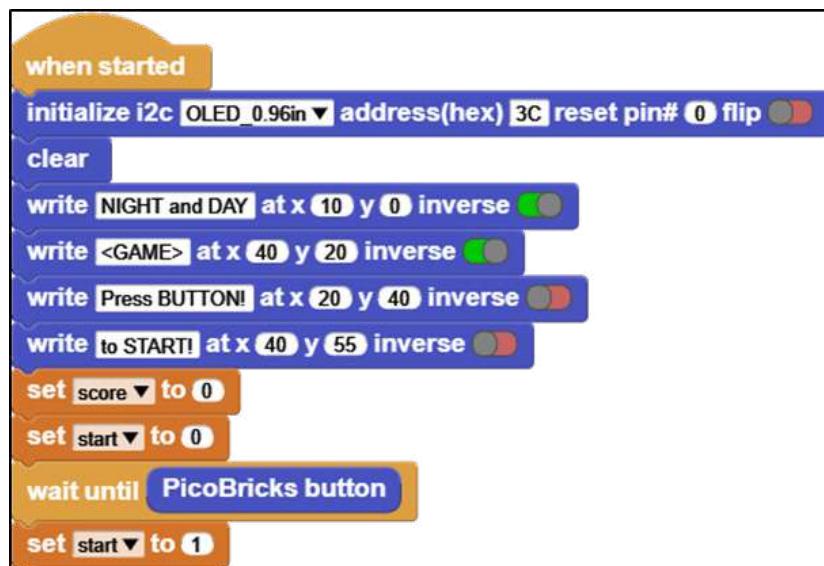


2.15.4. Project Proposal

You can develop the project by taking the values that the LDR sensor sends to the project according to the environment you are in, and automatically determining the limit to be processed according to the sensor value in the game, that is, by adding LDR sensor value calibration codes. You can add difficulty level to the game. With the potentiometer, the difficulty level can be selected as easy, medium and hard. When easy is selected, the change time for words can be 2 seconds, 1.5 seconds when medium is selected, 1 second when hard is selected.

2.15.5. Coding the Project with MicroBlocks

When Picobricks starts, we define the OLED screen and print the startup screen messages. Then create variables named score, start, nightorday and gamerReaction. Since the game will start after the button is pressed, the start variable is set to 1 after the wait until block.



A “change word” will be broadcast every two seconds when the game starts. Thanks to this news, the expression DAY or NIGHT will be printed on the screen. During the 2 seconds that the word remains on the screen, the player’s reaction will be assigned to the gamerReaction variable according to the value read from the LDR sensor. If the LDR value is greater than 80, the top of the sensor is not open, otherwise the sensor is closed. You can change the value of 80 for your own operating conditions.

At the end of the last two seconds, Correct and Wrong broadcasts are made by comparing the randomly determined word (nightorday) with gamerReaction. These broadcasts will be used to earn points in the game and to end the game.



```

when start = 1
broadcast [change word v]
reset timer
repeat (until [timer > (2000)])
  if [PicoBricks light sensor < (80)]
    set [gamerReaction v] to (0)
  else
    set [gamerReaction v] to (1)
  end
  PicoBricks beep [100] ms
  if [nightorday] = [gamerReaction]
    broadcast [Correct v]
  else
    broadcast [Wrong v]
  end
end
stop this task

```

When a change word message is received at two-second intervals, the screen is cleared and 0 or 1 value is randomly assigned to the nightorday variable. If the value is 0, NIGHT, is not 0, DAY is printed on the screen. In order for the broadcast block to run once, this command sequence is stopped at the end with the stop this task command.

```

when [change word] received
clear
set [nightorday v] to [random (0) to (1)]
if [nightorday] = (0)
  write [--NIGHT--] at x [20] y [30] inverse
else
  write [--DAY--] at x [25] y [30] inverse
end
stop this task

```

The code block that contains the codes that increases the score variable by 10 once for the "Correct" message, which is broadcast when the player reacts correctly, is as follows.

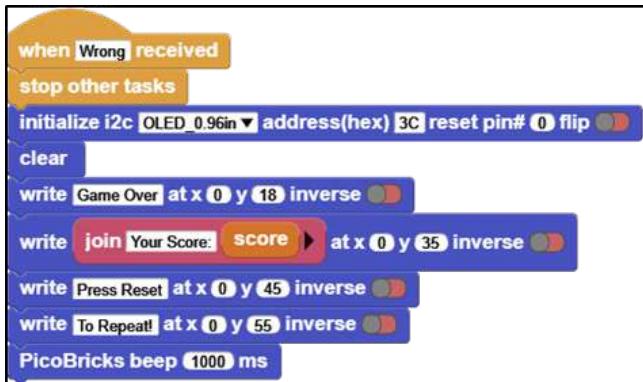
```

when [Correct] received
change [score v] by (10)
stop this task

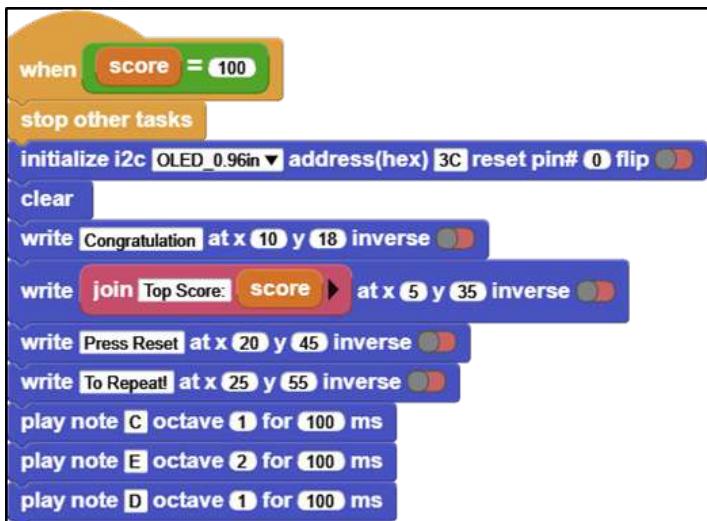
```



The code block, which contains the codes that stop all other codes for the “Wrong” message, which is broadcast when the player reacts incorrectly, and print the text about the end of the game and the score value on the screen, is as follows.



When the player completes the game without any errors, the score variable will take the value 100. The code block containing the codes that caught this and congratulated the player and stopped the game is as follows.



[Click](#) to access the codes of the project.

2.15.6. MicroPython Codes of the Project

```

from machine import Pin, I2C, Timer, ADC, PWM
from picobricks import SSD1306_I2C
import utime
import urandom
#define the libraries
WIDTH = 128
HEIGHT = 64
#OLED Screen Settings

```



```
sda=machine.Pin(4)
scl=machine.Pin(5)
#initialize digital pin 4 and 5 as an OUTPUT for OLED Communication
i2c=machine.I2C(0,sda=sda, scl=scl, freq=1000000)
oled = SSD1306_I2C(WIDTH, HEIGHT, i2c)
buzzer = PWM(Pin(20))
buzzer.freq(440)
ldr=ADC(Pin(27))
button=Pin(10,Pin.IN,Pin.PULL_DOWN)
#define the input and output pins
oled.text("NIGHT and DAY", 10, 0)
oled.text("<GAME>", 40, 20)
oled.text("Press the Button", 0, 40)
oled.text("to START!", 40, 55)
oled.show()
#OLED Screen Texts Settings
def changeWord():
    global nightorday
    oled.fill(0)
    oled.show()
    nightorday=round(urandom.uniform(0,1))
    #when data is '0', OLED texts NIGHT
    if nightorday==0:
        oled.text("---NIGHT---", 20, 30)
        oled.show()
    else:
        oled.text("---DAY---", 20, 30)
        oled.show()
    #waits for the button to be pressed to activate

while button.value()==0:
    print("Press the Button")
    sleep(0.01)

oled.fill(0)
oled.show()
start=1
global score
score=0
while start==1:
    global gamerReaction
```



```
global score
changeWord()
startTime=utime.ticks_ms()
#when LDR's data greater than 2000, gamer reaction '0'
while utime.ticks_diff(utime.ticks_ms(), startTime)<=2000:
    if ldr.read_u16()>20000:
        gamerReaction=0
    #when LDR's data lower than 2000, gamer reaction '1'
    else:
        gamerReaction=1
    sleep(0.01)
#buzzer working
buzzer.duty_u16(2000)
sleep(0.05)
buzzer.duty_u16(0)
if gamerReaction==nightorday:
    score += 10
#when score is 10, OLED says 'Game Over'
else:
    oled.fill(0)
    oled.show()
    oled.text("Game Over", 0, 18, 1)
    oled.text("Your score " + str(score), 0,35)
    oled.text("Press RESET",0, 45)
    oled.text("To REPEAT",0,55)
    oled.show()
    buzzer.duty_u16(2000)
    sleep(0.05)
    buzzer.duty_u16(0)
    break;
if score==100:
    #when score is 10, OLED says 'You Won'
    oled.fill(0)
    oled.show()
    oled.text("Congratulation", 10, 10)
    oled.text("Top Score: 100", 5, 35)
    oled.text("Press Reset", 20, 45)
    oled.text("To REPEAT", 25,55)
    oled.show()
    buzzer.duty_u16(2000)
    sleep(0.1)
```



```
buzzer.duty_u16(0)
sleep(0.1)
buzzer.duty_u16(2000)
sleep(0.1)
buzzer.duty_u16(0)

break;
```

2.15.7. Arduino C Codes of the Project

```
#include <Wire.h>
#include "ACROBOTIC_SSD1306.h"

//define the library

#define RANDOM_SEED_PIN 28
int Gamer_Reaction = 0;
int Night_or_Day = 0;
int Score = 0;
int counter=0;

double currentTime = 0;
double lastTime = 0;
double getLastTime(){
    return currentTime = millis()/1000.0 - lastTime;
}

void _delay(float seconds) {
    long endTime = millis() + seconds * 1000;
    while(millis() < endTime) _loop();
}

void _loop() {
}

void loop() {
    _loop();
}

void setup() {

// put your setup code here, to run once
```



```
pinMode(10,INPUT);
pinMode(27,INPUT);
pinMode(20,OUTPUT);
randomSeed(RANDOM_SEED_PIN);
Wire.begin();
oled.init();
oled.clearDisplay();
```

```
oled.clearDisplay();
oled.setTextXY(1,3);
oled.putString("NIGHT and DAY");
oled.setTextXY(2,7);
oled.putString("GAME");
oled.setTextXY(5,2);
oled.putString("Press BUTTON!");
oled.setTextXY(6,4);
oled.putString("to START!");
```

```
Score = 0;
```

```
while(!(digitalRead(10) == 1))
{
    _loop();
}
_delay(0.2);
```

```
while(1){ //while loop
    if (counter==0){

        delay(500);
        Change_Word();
        lastTime = millis()/1000.0;
```

```
while(!(getLastTime() > 2))
{
    Serial.println(analogRead(27));
    if(analogRead(27) > 500){
        Gamer_Reaction = 0;
    }else{
```



```
Gamer_Reaction = 1;
}
}
//determine the gamer reaction based on the value of the LDR sensor
digitalWrite(20,HIGH); //turn on the buzzer
delay(250);
digitalWrite(20,LOW); //turn off the buzzer

if(Night_or_Day == Gamer_Reaction){
    Correct();
}

}else{
    Wrong();

} _loop();

if(Score==100){
    oled.clearDisplay();
    oled.setTextXY(1,1);
    oled.putString("Congratulation");
    oled.setTextXY(3,1);
    oled.putString("Your Score: ");
    oled.setTextXY(3,13);
    String String_Score=String(Score);
    oled.putString(String_Score);
    oled.setTextXY(5,3);
    oled.putString("Press Reset");
    oled.setTextXY(6,3);
    oled.putString("To Repeat!");
    //write the "Congratulation, Your Score, press Reset, To Repeat!" and score variable
    //on the x and y coordinates determined on the OLED screen
    for(int i=0;i<3;i++){
        digitalWrite(20,HIGH);
        delay(500);
        digitalWrite(20,LOW);
        delay(500);
    } counter=1; //turn the buzzer on and off three times
}
}
}
}
```



```
void Correct (){
    Score += 10;
    oled.clearDisplay();
    oled.setTextXY(3,4);
    oled.putString("10 points");
    //increase the score by 10 when the gamer answers correctly
}

void Change_Word (){
    oled.clearDisplay();
    Night_or_Day=random(0,2);

    if (Night_or_Day==0){
        oled.setTextXY(3,6);
        oled.putString("NIGHT");
    }else{
        oled.setTextXY(3,7);
        oled.putString("DAY");
    }
}
//write "NIGHT" or "DAY" on random OLED screen
void Wrong (){
    oled.clearDisplay();
    oled.setTextXY(1,3);
    oled.putString("Game Over");
    oled.setTextXY(3,1);
    oled.putString("Your Score: ");
    oled.setTextXY(3,13);
    String String_Score=String(Score);
    oled.putString(String_Score);
    oled.setTextXY(5,3);
    oled.putString("Press Reset");
    oled.setTextXY(6,3);
    oled.putString("To Repeat!");
    // write the score variable and the expressions are quotation marks to the
    coordinates determined on the OLED screen.

    digitalWrite(20,HIGH);
    delay(1000);
    digitalWrite(20,LOW);
    counter=1;
}
```



GitHub Night and Day Project Page



<http://rbt.ist/nightday>



2.16. Voice Controlled Robot Car

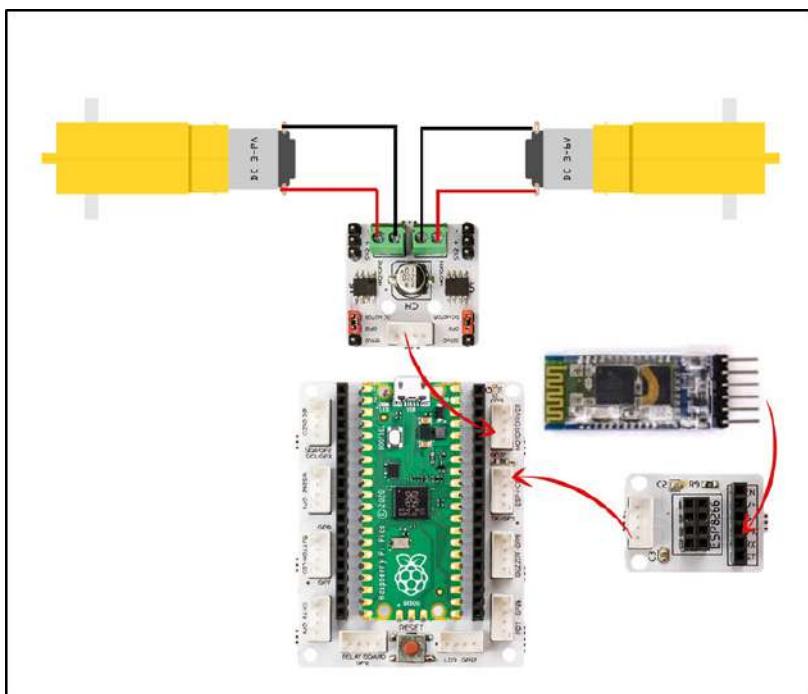
Developing and continuing to develop artificial intelligence applications recognize human characteristics, learn and try to behave like people. We can express artificial intelligence as software that can learn in its shortest form. Sometimes it learns the image, sometimes the sound, and sometimes by using the data it collects from the sensors. It does this thanks to the algorithms determined by the developers, and it helps in the decision-making processes in the areas it is used according to the results it has achieved. In short, artificial intelligence applications are now used in situations where the decision-making process needs to be done quickly and without errors. From the marketing field to the defense industry, from education to health, from economy to entertainment, artificial intelligence increases efficiency and reduces costs.

In this project we will do with PicoBricks, we will make a 2WD car that you can control by talking. PicoBricks allows you to communicate wirelessly with 2 6V DC motors and bluetooth.

2.16.1. Project Details and Algorithm

In the project, the robot car kit that comes out of the set will be assembled and controlled via mobile phone. The HC05 bluetooth module is a module that enables us to communicate wirelessly between PicoBricks and a mobile phone. Thanks to the mobile application installed on the mobile phone in the project, the commands sent from the phone will be transmitted to PicoBricks via the HC05 module and the robot car will move according to these data. We can direct the robot car with the forward, backward, right, left buttons from the mobile phone, as well as send data to PicoBricks with voice command. In the project, we will give voice commands to control the movements of the robot car.

2.16.2. Wiring Diagram





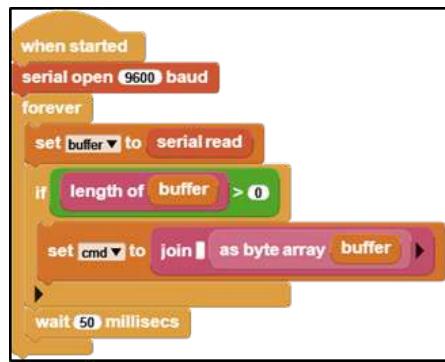
2.16.3. Project Proposal

In this project, we moved the robot car by giving voice commands via the mobile application we installed on the mobile phone. You can control the mechanism with voice commands or buttons by connecting the HC05 bluetooth module to the pan-tilt mechanism in the two-axis robot arm project. Likewise, you can try a mobile application where you can control the robot car in this project using buttons instead of voice commands, or you can develop a mobile application specific to your project with the MIT Appinventor editor.

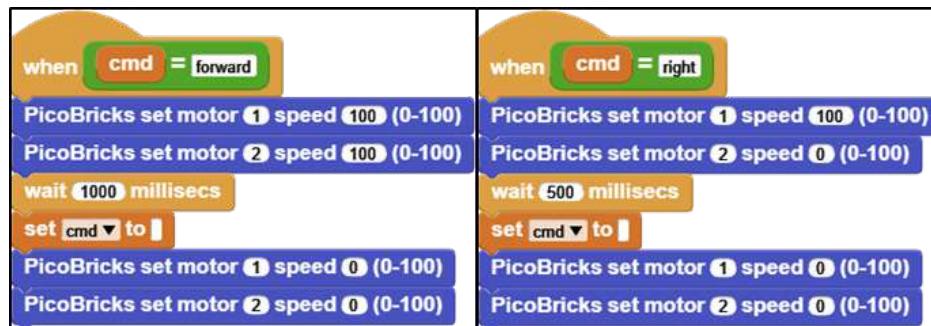
With the HC05 Bluetooth module, you can operate not only the motor driver and motor, but also other modules on PicoBricks. For example, you can light the RGB LED in any color you want through the mobile application, read the temperature and humidity values from the DHT11 module, the light values on the LDR sensor, and print texts on the OLED screen. There is a mobile application specially written for these processes with the MIT Appinventor editor, and ready-made codes written in Microblocks to automatically run the data coming from the application. You can run all these features by downloading and running the Microblocks file from the link below and by downloading the android apk file and installing it on your phone.

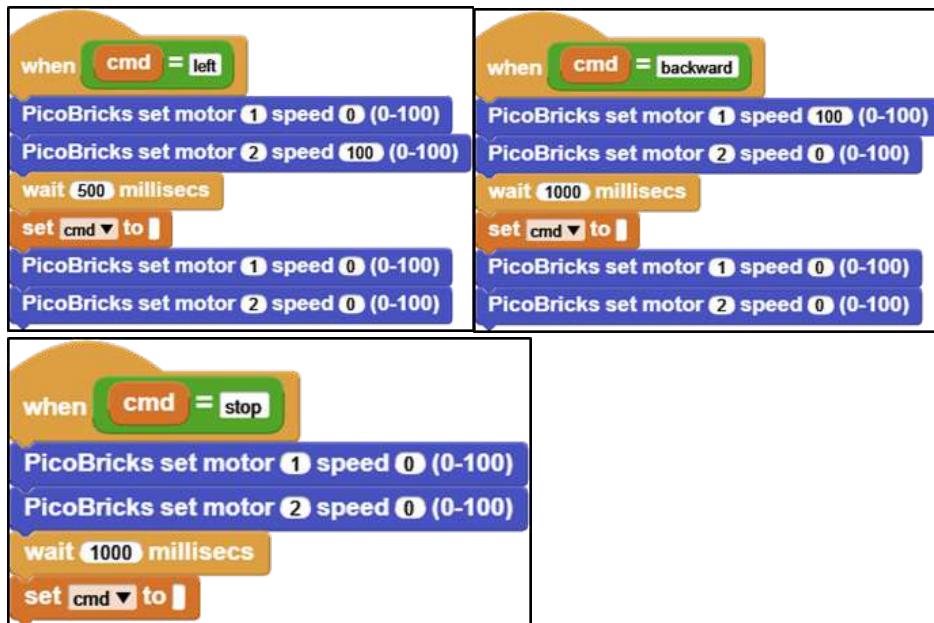
[Download Link](#)

2.16.4. Coding the Project with MicroBlocks



After completing the assembly of the project, we first create two variables called cmd and buffer in the code part, and read the serial port defined in 9600 bandwidth with the buffer variable and transfer the incoming information to the cmd variable. If you have written all the codes and you are not communicating between the phone and PicoBricks, you can try changing the bandwidth from 9600 to 115200.



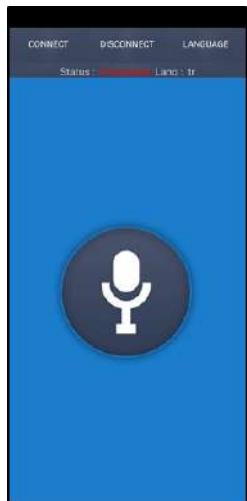


[Click](#) to access the project's MicroBlocks codes.

With the HC05 module, we perform comparison operations using the cmd variable, which keeps the information coming over the serial port, and when blocks. If the incoming information is “forward”, we must run the motor1 and motor2 blocks between 0 and 100 values so that the robot car can move forward. The speed of your vehicle will change in direct proportion to the fullness of your battery. In the same way, we write the necessary codes for the vehicle to turn right if the information received via bluetooth is “right”, to turn left if it is “left”, to go back if it is “backward”, and to stop the vehicle if it is “stop”. In these codes, the vehicle is provided to perform movements for a certain period of time. You can extend or remove the deadlines if you want

After writing the codes and running it on PicoBricks, we download the mobile application for android devices from the link below and install it on the phone.

[Download Link](#)

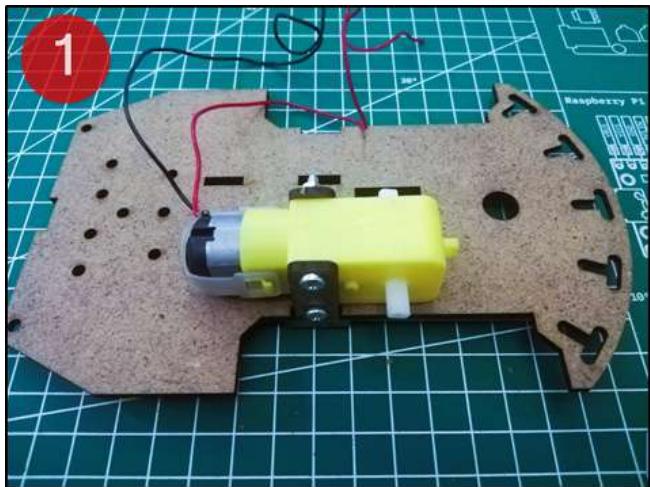




You can open the language options with the Language button and edit the language in English with the "en" option and the language in Turkish with the "tr" option.

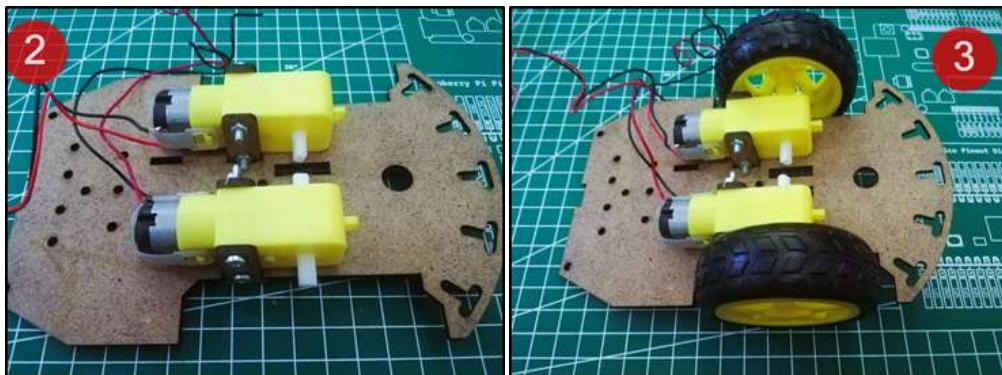
2.16.5. Construction Stages of the Project

1. Screw the first motor to the chassis of the 2WD robot car that comes out of the set and fix it.



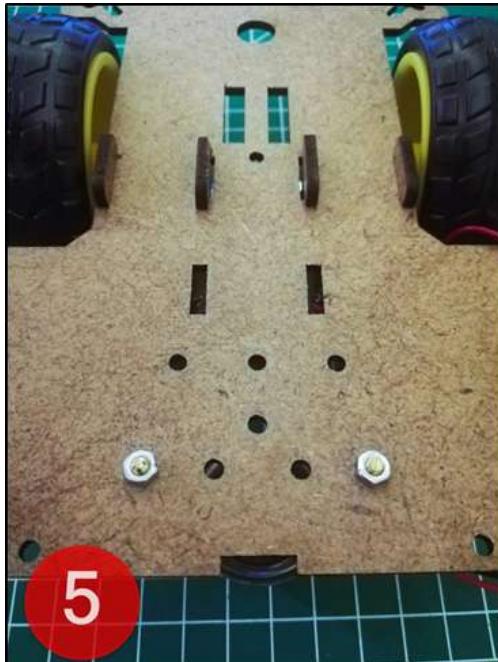
2. Fix the second motor by screwing it to the chassis.

3. Attach the wheels to the motors.



4. Fix the caster under the chassis using spacers.

5. Fix the spacer with the nut from the top of the chassis.



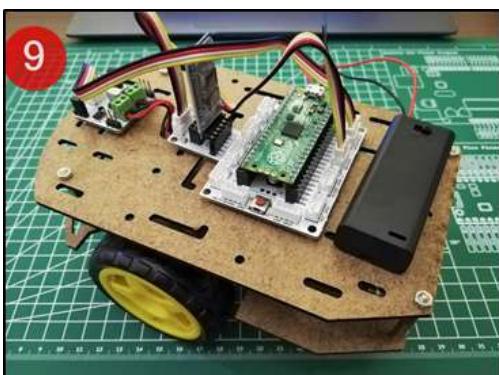
6. Fix 4 spacers on the four corners of the lower chassis.

7. Fix the upper chassis with plug and nuts.



8. Connect the cables of the motors to the terminals on the motor driver.

9. Fix the motor driver, Bluetooth module, Picrobricks board and battery box to the chassis using hot silicone.





2.16.6. MicroPython Codes of the Project

```

from machine import Pin, UART
from utime import sleep

uart = UART(0,9600) #If connection cannot be established, try 115200.
m1 = Pin(21, Pin.OUT)
m2 = Pin(22, Pin.OUT)

m1.low()
m2.low()

while True:
    sleep(0.05)
    if uart.any():
        cmd = uart.readline()
    if cmd==b'F':
        m1.high()
        m2.high()
    elif cmd==b'R':
        m1.high()
        m2.low()
    elif cmd==b'L':
        m1.low()
        m2.high()
    elif cmd==b'S':
        m1.low()
        m2.low()
    cmd=""

```

2.16.7. Arduino C Code of the Project

```

void setup() {
  Serial1.begin(9600);
}

void loop() {
  if (Serial1.available() > 0) {

    char sread = Serial1.read();
    Serial.println(sread);
}

```



```
if (sread == 'f') {
    Forward();
} else if(sread == 'r'){
    Turn_Right();
} else if(sread == 'l'){
    Turn_Left();
} else if(sread == 's'){
    Stop();
}

void Forward(){
    digitalWrite(21,HIGH);
    digitalWrite(22,HIGH);
    delay(1000);
    digitalWrite(21,LOW);
    digitalWrite(22,LOW);
}

void Turn_Left(){
    digitalWrite(21,LOW);
    digitalWrite(22,HIGH);
    delay(500);
    digitalWrite(21,LOW);
    digitalWrite(22,LOW);
}

void Turn_Right(){
    digitalWrite(21,HIGH);
    digitalWrite(22,LOW);
    delay(500);
    digitalWrite(21,LOW);
    digitalWrite(22,LOW);
}

void Stop(){
    digitalWrite(21,LOW);
    digitalWrite(22,LOW);
    delay(1000);
}
```



After uploading the Arduino codes, download the android application from the link below and open the terminal mode and use the letters f, b, r, l, s for vehicle movements.

[Link](#)

GitHub Voice Controlled Robot Car Project Page



<http://rbt.ist/voicecar>



2.17. Two Axis Robot Arm

Robot arms have replaced human power in the industrial field. In factories, robotic arms undertake the tasks of carrying and turning loads of weights and sizes that cannot be carried by a human. Being able to be positioned with a precision of one thousandth of a millimeter is above the sensitivity that a human hand can exhibit. When you watch the production videos of automobile factories, you will see how vital the robot arms are. The reason why they are called robots is that they can be programmed to do the same work with endless repetitions. The reason why it is called an arm is because it has an articulated structure like our arms. How many different directions a robot arm has the ability to rotate and move is expressed as axes. Robot arms are also used for carving and shaping aluminum and various metals. These devices, which are referred to as 7-axis CNC Routers, can shape metals like a sculptor shapes mud.

According to the purpose of use in robot arms, stepper motor and servo motors, which are a kind of electric motor, are used. PicoBricks allows you to make projects with servo motors.

2.17.1. Project Details and Algorithm

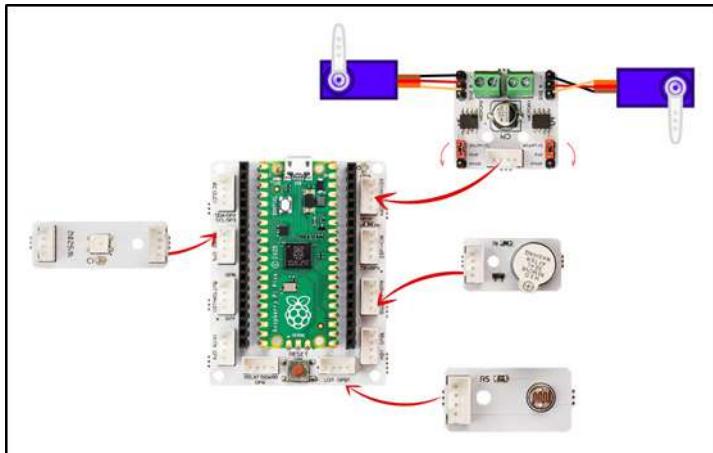
In preparation for the installation, we will first write and upload the codes to set the servo motors to 0 degrees. When an object is placed on the LDR sensor, the robot arm will bend down and close its open gripper. After the gripper is closed, the robot arm will rise again. As a result of each movement of the robot arm, a short beep will be heard from the buzzer. The RGB LED will glow red when an object is placed on the LDR sensor. When the object is held by the robot arm and lifted into the air, the RGB LED will turn green.

Servo motor movements are very fast. In order to slow down the movement, we will code the servo motors with a total of 90 degrees of movement, 2 degrees each at 30 millisecond intervals. We're not going to do this for the gripper to close.

In order for the servo to perform its holding and releasing function, print and assemble the necessary parts from the 3D printer from the [link here](#).



2.17.2. Wiring Diagram



2.17.3. Project Proposal

By adding the HC05 module to the 2 axis robot arm project, you can develop it by controlling it from your mobile phone with the mobile application.

2.17.4. Construction Stages of the Project

Prepare the parts of the Pan-Tilt kit to prepare the project. Carry your 3D printed parts, waste cardboard pieces, hot silicone glue and scissors with you.





1. First of all, we will prepare the fixed arm of the robot arm. Make an 8 cm high cardboard cylinder into the rounded part of part D. Place it on the D part and stick it with silicone.



2. Place the head that came out of the servo motor package on the C part by shortening it a little. Fix with the smallest screws from the Pan Tilt kit.

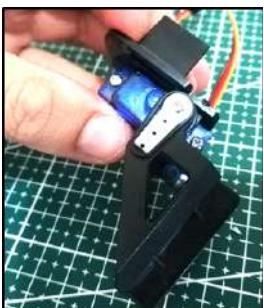


3. Fix parts A and C together with 2 pointed screws.

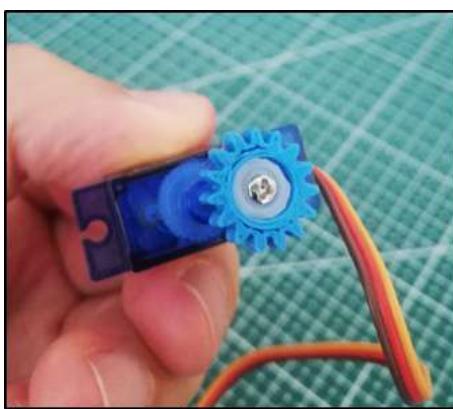




4. Internally attach the servo motor to part C. Then place the servo motor on part B and screw it.



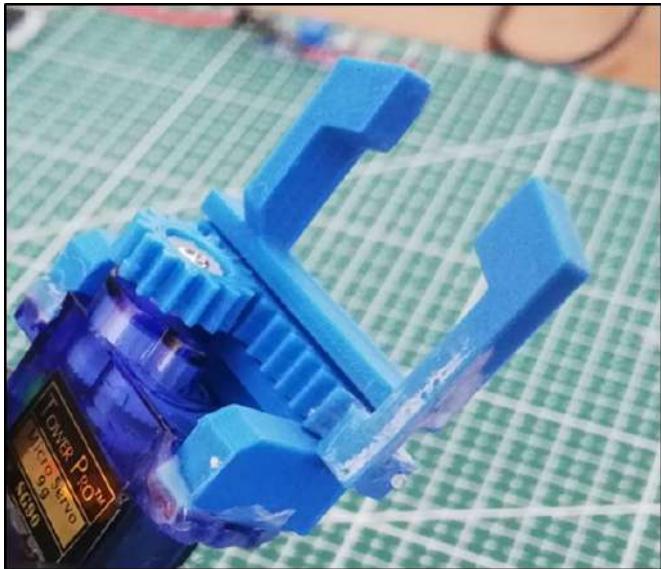
5. For the holder, cut one of the servo motor heads in the middle of the gear part that you printed on the 3D printer and place it into the gear. Then screw it to the servo motor.



6. Adhere together the 3D printed Linear gear and the handle with strong adhesive.



7. Place the servo in the 3D print holder and fix it. You can do this with hot silicone or by screwing. When placing the servo gear on the linear gear, make sure it is fully open.



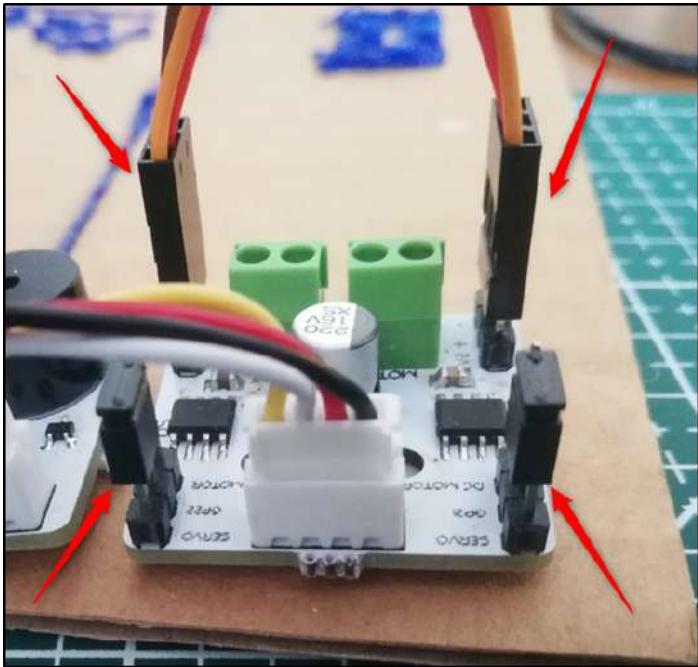
8. Stick the holding servo system to part B with silicone.



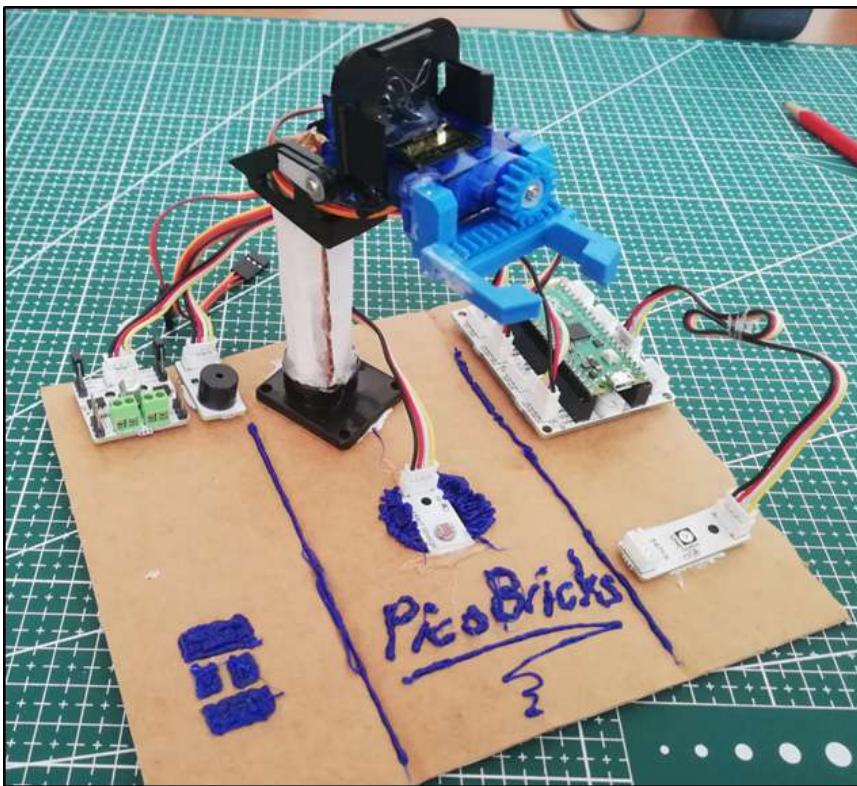
9. Pass the piece we prepared in step 3 over the cylinder we prepared from cardboard in the first step and fix it with silicone.



10. Put the motor drive jumpers on the Servo pins. Connect the cable of the holding servo to the GPIO21 and the cable of the tilting servo to the GPIO22.



11. Place the motor driver, buzzer, LDR and RGB LED module on a platform and place the robot arm on the platform accordingly. With the 3D Pen printer, you can customize your project as you wish.



12. You can operate the Robot arm if you feed Picobricks with USB or 3 pen batteries from the power jack on the Picoboard.

2.17.5. Coding the Project with MicroBlocks

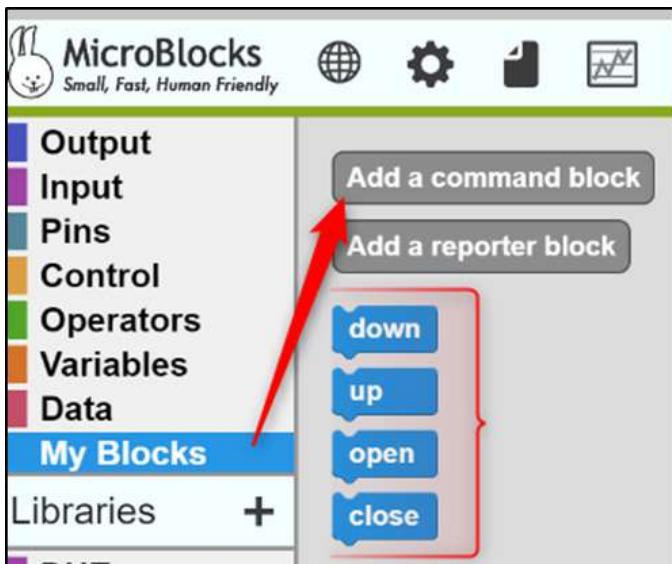
Open Microblocks and connect to Picobricks. Add the servo library. Connect the



holding motor to pin 21 and the bending motor to pin 22. Set the servo value of the handle end of the robot arm to -90. Set the angle of the tilt motor to 0. When you prepare the blocks below and click on them, the codes will run and the servos will turn to the angle you set.

```
set servo 22 to 0 degrees (-90 to 90)
set servo 21 to -90 degrees (-90 to 90)
```

We will prepare the servo motor movements in separate blocks. For this, create four command blocks named up, down, open, close from the My Blocks category.



For up and down arm movement, code the up and down blocks for 45 reps to rotate 2 degrees each in 30 milliseconds. In order to change the angle value, create a variable named angleupdown. There will be a change of -2 degrees in the down block and 2 degrees in the up block.

<pre>define up repeat (45) change angleupdown by 2 set servo 22 to angleupdown degrees (-90 to 90) wait 30 millisecs play note C octave 1 for 100 ms</pre>	<pre>define down repeat (45) change angleupdown by -2 set servo 22 to angleupdown degrees (-90 to 90) wait 30 millisecs play note C octave 1 for 100 ms</pre>
--	---

To turn the gripper servo on, code its angle 90, to turn it off, code its angle -60. Add a sound of your choice by adding the Tone library to the end of all servo movements. 100ms of ringing is enough.

<pre>define open set servo 21 to 90 degrees (-90 to 90) play note D octave 2 for 100 ms</pre>	<pre>define close set servo 21 to -60 degrees (-90 to 90) play note D octave 2 for 100 ms</pre>
---	---



When the picobricks starts, the gripper servo should turn on, the angleupdown variable should be set to its initial value of 90, and the robot arm should turn 90 degrees with the upwards pointing. The RGB LED should go out.

```

when started
  open
  set angleupdown ▾ to 90
  set servo 21 to angleupdown degrees (-90 to 90)
  PicoBricks turn off RGB LED
  
```

We realize that an object is on the LDR sensor when its value drops below 10. This value may be different for your project. You can find out how many reads by clicking on the block. Firstly, the RGB LED lights up red. In order to hold and lift the object, the holding motor is opened, the downward movement is made, the holding motor is closed and the upward movement is realized. Finally, the RGB LED lights up green. Make it easier to watch the movement by putting a 500ms pause between each movement.

```

when PicoBricks light sensor (0-100) % < 10
  PicoBricks set RGB LED color red
  wait 1000 millisecs
  PicoBricks beep 1000 ms
  open
  wait 500 millisecs
  down
  wait 500 millisecs
  close
  wait 500 millisecs
  up
  PicoBricks set RGB LED color green
  
```

[Click](#) to access the codes of the project.



2.17.6. MicroPython Codes of the Project

Codes that calibrate servo motors:

```
from machine import Pin, PWM,  
servo1=PWM(Pin(21))  
servo2=PWM(Pin(22))  
  
servo1.freq(50)  
servo2.freq(50)  
  
servo1.duty_u16(8200) # 180 degree  
servo2.duty_u16(4770) # 90 degree
```

Project Codes:

```
from machine import Pin, PWM, ADC  
from utime import sleep  
from picobricks import WS2812  
#define libraries  
  
ws = WS2812(6, brightness=0.3)  
ldr=ADC(27)  
buzzer=PWM(Pin(20, Pin.OUT))  
servo1=PWM(Pin(21))  
servo2=PWM(Pin(22))  
# define LDR, buzzer and servo motors pins  
  
servo1.freq(50)  
servo2.freq(50)  
buzzer.freq(440)  
# define frequencies of servo motors and buzzer  
  
RED = (255, 0, 0)  
GREEN = (0, 255, 0)  
BLACK = (0, 0, 0) # RGB color settings  
angleupdown=4770  
angleupdown2=8200  
  
def up():  
    global angleupdown
```



```
for i in range (45):
    angleupdown +=76
    servo2.duty_u16(angleupdown)
    sleep(0.03)
    buzzer.duty_u16(2000)
    sleep(0.1)
    buzzer.duty_u16(0)
    # servo2 goes up at specified intervals
def down():
    global angleupdown
    for i in range (45):
        angleupdown -=76
        servo2.duty_u16(angleupdown)
        sleep(0.03)
        buzzer.duty_u16(2000)
        sleep(0.1)
        buzzer.duty_u16(0)
        # servo2 goes down at specified intervals

def open():
    global angleupdown2
    for i in range (45):
        angleupdown2 +=500
        servol.duty_u16(angleupdown2)
        sleep(0.03)
        buzzer.duty_u16(2000)
        sleep(0.1)
        buzzer.duty_u16(0)
        # servol works for opening the clamps
def close():
    global angleupdown2
    for i in range (45):
        angleupdown2 -=500
        servol.duty_u16(angleupdown2)
        sleep(0.03)
        buzzer.duty_u16(2000)
        sleep(0.1)
        buzzer.duty_u16(0)
        # servol works for closing the clamps
open()
servo2.duty_u16(angleupdown)
```



```

ws.pixels_fill(BLACK)
ws.pixels_show()
while True:
    if ldr.read_u16()>20000:
        ws.pixels_fill(RED)
        ws.pixels_show()
        sleep(1)
        buzzer.duty_u16(2000)
        sleep(1)
        buzzer.duty_u16(0)
        open()
        sleep(0.5)
        down()
        sleep(0.5)
        close()
        sleep(0.5)
        up()
        ws.pixels_fill(GREEN)
        ws.pixels_show()
        sleep(0.5)
# According to the data received from LDR, RGB LED lights red and green and
servo motors move

```

2.16.7. Arduino C Codes of the Project

```

##include <Adafruit_NeoPixel.h>
#ifndef __AVR__
#include <avr/power.h>
#endif
#define PIN      6
#define NUMPIXELS 1
Adafruit_NeoPixel pixels(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);
#define DELAYVAL 500
// define required libraries
#include <Servo.h>
Servo myservo1;
Servo myservo2;

int angleupdown;

void setup() {

```



```
pinMode(20,OUTPUT);
pinMode(27,INPUT);
// define input and output pins

pixels.begin();
pixels.clear();

myservo1.attach(21);
myservo2.attach(22); // define servo motor pins
Open();
angleupdown=180;
myservo2.write(angleupdown);

}

void loop() {
if(analogRead(27)>150){

pixels.setPixelColor(0, pixels.Color(255, 0, 0));
pixels.show();
delay(1000);
tone(20,700);
delay(1000);
noTone(20);

Open();
delay(500);
Down();
delay(500);
Close();
delay(500);
Up();
pixels.setPixelColor(0, pixels.Color(0, 255, 0));
pixels.show();
delay(10000);
pixels.setPixelColor(0, pixels.Color(0, 0, 0));
pixels.show();
Open();
angleupdown=180;
myservo2.write(angleupdown);
// If the LDR data is greater than the specified limit, the buzzer will sound, the RGB
will turn red and servo motors will work
}
```



```
// The RGB will turn green when the movement is complete

}

}

void Open(){
    myservo1.write(180);
}

void Close(){
    myservo1.write(30);
}

void Up(){

for (int i=0;i<45;i++){

    angleupdown = angleupdown+2;
    myservo2.write(angleupdown);
    delay(30);
}
}

void Down(){

for (int i=0;i<45;i++){

    angleupdown = angleupdown-2;
    myservo2.write(angleupdown);
    delay(30);
}
}
```



GitHub Two Axis Robot Arm Project Page



<http://rbt.ist/robotarm>



2.18. Smart House

Workplaces, factories, homes and even animal shelters... There are different electronic systems that can be used to protect our living spaces against intruders. These systems are produced and marketed as home and workplace security systems. There are systems where the images produced by security cameras are processed and interpreted, as well as security systems that detect the human body and its movements with sensors and take action. Security systems are set up like a kind of alarm clock and give audible and visual warnings when an unidentified activity is detected in the specified time zone. It notifies the business or the home owner, and it can also make automatic notifications to the security units.

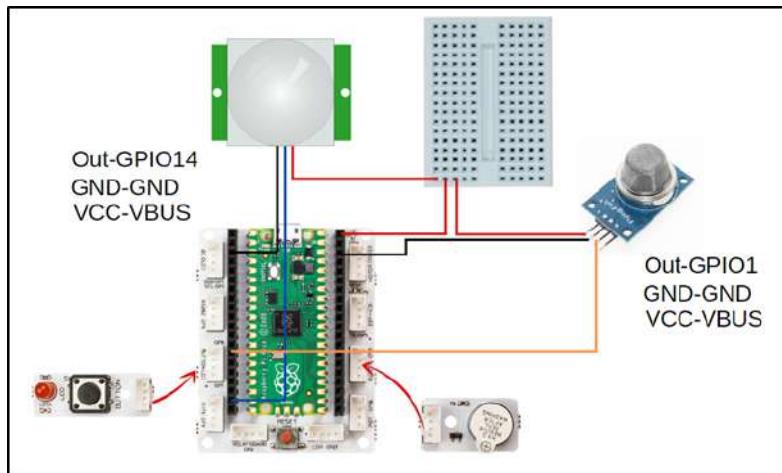
Gas leakage, fire etc. in such cases, gas sensors are used in homes and workplaces to prevent poisoning. In a negative situation, people living in the environment are warned by giving a loud alarm.

We will prepare a model smart home project with PicoBricks using the HC-SR501 and MQ-2 gas sensor. This sensor HC-SR501, also known as PIR sensor, detects motion by capturing the changes of infrared waves reflected by the human body.

2.18.1. Project Details and Algorithm

When the HC-SR501 PIR sensor detects motion, it gives digital output for 3 seconds. We will use a Picoboard, buzzer and button LED module in the project. All parts must be in the model. When Picobricks starts, the button must be pressed to activate the alarm system. After pressing the button, we must wait 3 seconds for the hand to be pulled out of the model. At the end of 3 seconds, the red LED lights up and the alarm system is activated. When the alarm system detects a movement, the red LED will start to flash and the buzzer will sound the alarm. To mute it, Picobricks must be restarted. The MQ-2 sensor is always on. When it detects a toxic gas, it will notify you with a buzzer and red LED.

2.18.2. Wiring Diagram





2.18.3. Project Proposal

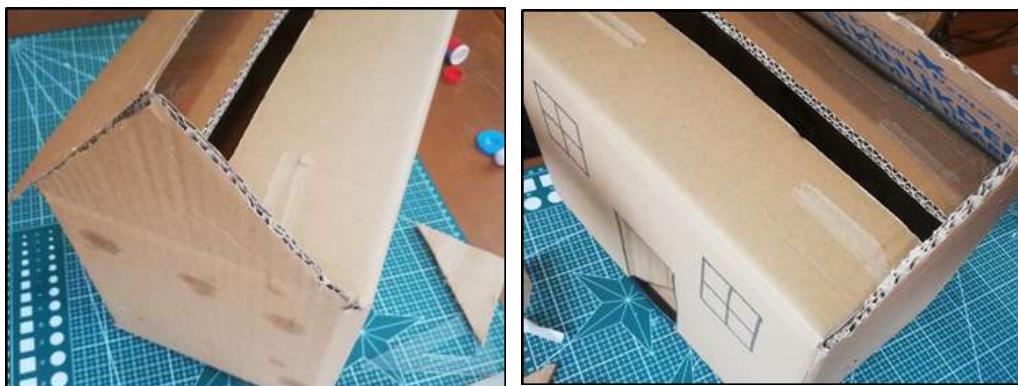
After making the 25th project, the smart greenhouse, by adding the ESP8266 mode to the burglar alarm project, you can send a notification to the home owner's phone when a thief enters to the home, and turn the project into an IOT project. You can install fire extinguishing pipes on the ceiling of the house with a submersible pump, so that when there is a fire in the house, you can automatically extinguish it.

2.18.4. Construction Stages of the Project

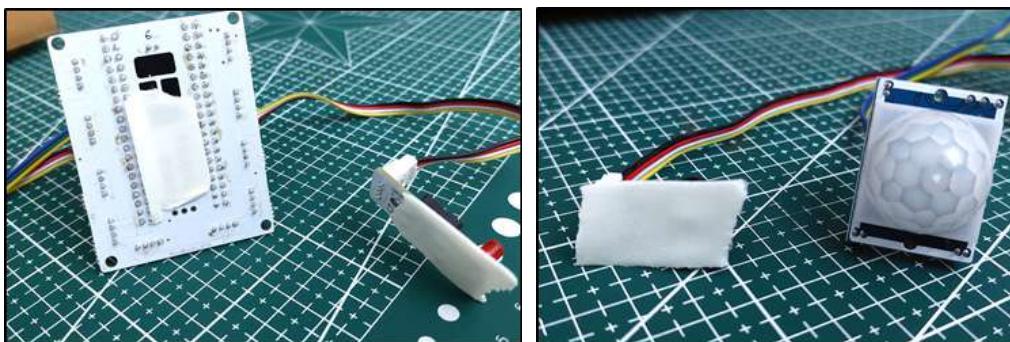
To run the project, you have to turn a cardboard box into a model house. You will need scissors, pencils, tape, glue, and a utility knife. Draw windows and doors on the box with a pencil. Cut the door section with a utility knife.



You can use another cardboard to make the roof part.



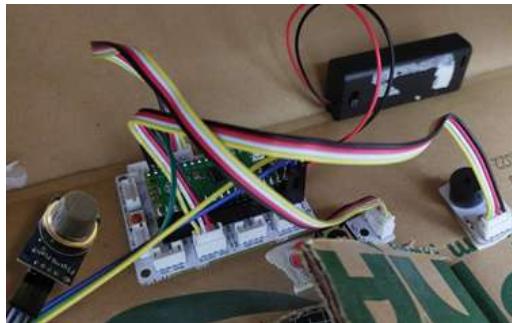
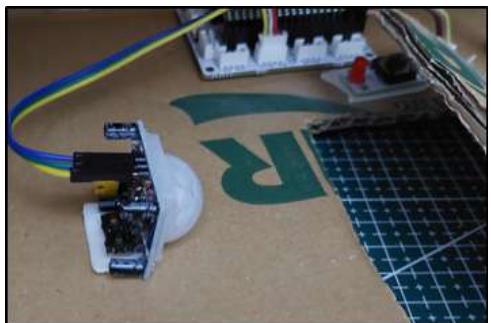
Stick double-sided foam tape under the picobricks pieces.



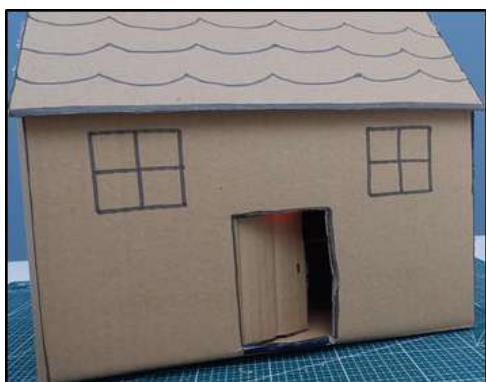
Place pieces of Picobricks inside the model house. Position the PIR sensor to see the



door directly from the inside. Stick the button module just above the door from the inside.



When you connect the battery case to Picoboard and open it, the codes will start to run. 3 seconds after pressing the button, the alarm system will be activated and the red LED will turn on. As soon as you put your hand in the door, the buzzer will start to sound.

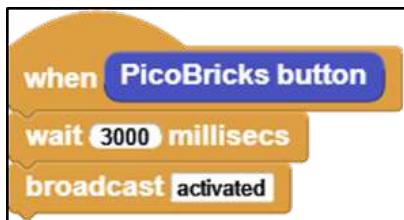


When you hold the lighter gas inside the house, the alarm system is expected to be activated again.

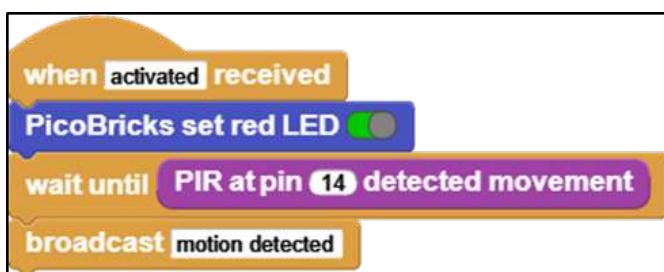


2.18.5. Coding the Project with MicroBlocks

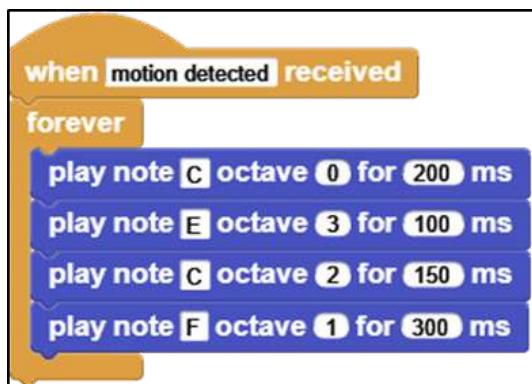
It waits for the button to be pressed to activate the alarm system. Activated broadcast is made 3 seconds after the button is pressed.



Indicate that the alarm system is activated by lighting the red LED. Go to Library>Sensing>PIR.ulb and add the block of the PIR sensor to MicroBlocks. Change the pin number to 14. Hold on code execution until motion is detected. Broadcast motion detected when motion is detected.



When motion detected broadcast is received, the buzzer will sound an alarm continuously. You can customize the alarm sound as you wish by adding the Tone.ulb library.



[Click](#) to access the codes of the project.

2.18.6. MicroPython Codes of the Project

```

from machine import Pin, PWM
from utime import sleep
# define libraries
PIR=Pin(14, Pin.IN)
MQ2=Pin(1,Pin.IN)
buzzer=PWM(Pin(20,Pin.OUT))
  
```



```
redLed=Pin(7,Pin.OUT)
button=Pin(10,Pin.IN,Pin.PULL_DOWN)
# define output and input pins

activated=0
gas=0

while True:
    if button.value()==1:
        activated=1
        gas=0
        sleep(3)
        redLed.value(1)
        buzzer.duty_u16(0)
    if MQ2.value()==1:
        gas=1
    if activated==1:
        if PIR.value()==1:
            buzzer.duty_u16(6000)
            buzzer.freq(440)
            sleep(0.2)
            buzzer.freq(330)
            sleep(0.1)
            buzzer.freq(494)
            sleep(0.15)
            buzzer.freq(523)
            sleep(0.3)
    if gas==1:
        buzzer.duty_u16(6000)
        buzzer.freq(330)
        sleep(0.5)
        redLed.value(1)
        buzzer.freq(523)
        sleep(0.5)
        redLed.value(0)
    # LED will light and buzzer will sound when PIR detects motion or MQ2 detects
    toxic gas
```



2.18.7. Arduino C Codes of the Project

```
void actived (){
    digitalWrite(7,1);
    while(!(digitalRead(14) == 1))
    {
        _loop();
    }
    motion_detected();
}

void motion_detected (){
    while(1) {
        // buzzer settings
        tone(20,262,0.25*1000);
        delay(0.25*1000);
        tone(20,330,0.25*1000);
        delay(0.25*1000);
        tone(20,262,0.25*1000);
        delay(0.25*1000);
        tone(20,349,0.25*1000);
        delay(0.25*1000);
        // sound the buzzer when PIR detected a motion
        _loop();
    }
}

void _delay(float seconds) {
    long endTime = millis() + seconds * 1000;
    while(millis() < endTime) _loop();
}

void _loop() {
}

void loop() {
    _loop();
}

void setup() {
```



```
pinMode(10,INPUT);
pinMode(1,INPUT);
pinMode(20,OUTPUT);
pinMode(7,OUTPUT);
pinMode(14,INPUT);
// define input and output pins

while(1) {
    if(digitalRead(10) == 1){
        _delay(3);
        actived();
    }
    if(digitalRead(1) == 1){
        while(!(digitalRead(10) == 1))
        {
            _loop();
            tone(20,349,0.5*1000);
            delay(0.5*1000);
            digitalWrite(7,1);
            _delay(0.5);
            tone(20,392,0.5*1000);
            delay(0.5*1000);
            digitalWrite(7,0);
            _delay(0.5);
        }
    }
    _loop();
}
}
```

GitHub Smart House Project Page



<http://rbt.ist/smarthouse>



2.19. Piggy Bank

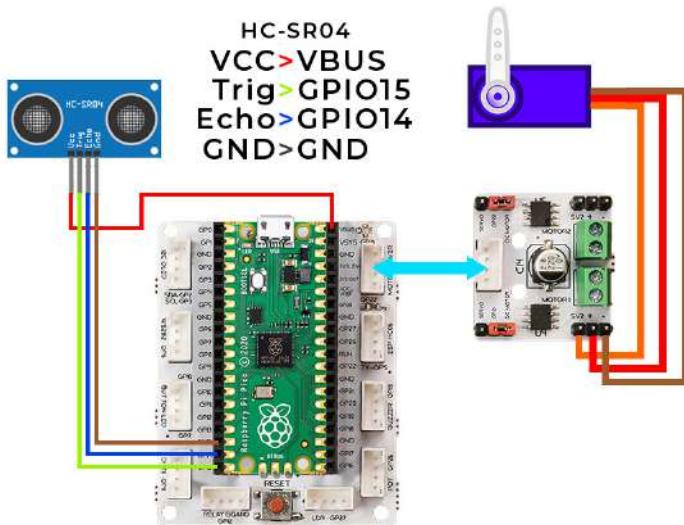
Ultrasonic sensors are sensors that show electrical change by being affected by sound waves. These sensors send sound waves at a frequency that our ears cannot detect and produce distance information by calculating the return time of the reflected sound waves. We, the programmers, develop projects by making sense of the measured distance and the changes in distance. Parking sensors in the front and back of the cars are the places where ultrasonic sensors are most common in daily life. Do you know the creature that finds its way in nature with this method? Because bats are blind, they find their way through the reflections of the sounds they make. [Ses dalgalarının gösterildiği bir görsel olabilir]

Many of us like to save money. It is a very nice feeling that the money we save little by little is useful when needed. In this project, you will make yourself a very enjoyable and cute piggy bank. You will use the servo motor and ultrasonic distance sensor while making the piggy bank.

2.19.1. Project Details and Algorithm

HC-SR04 ultrasonic distance sensor and SG90 servo motor will be used in this project. When the user leaves money in the hopper of the piggy bank, the distance sensor will detect the proximity and send it to the Picobricks. According to this information, Picobricks will operate a servo motor and raise the arm, throw the money into the piggy bank and the arm will go down again.

2.19.2. Wiring Diagram





2.19.3. Project Proposal

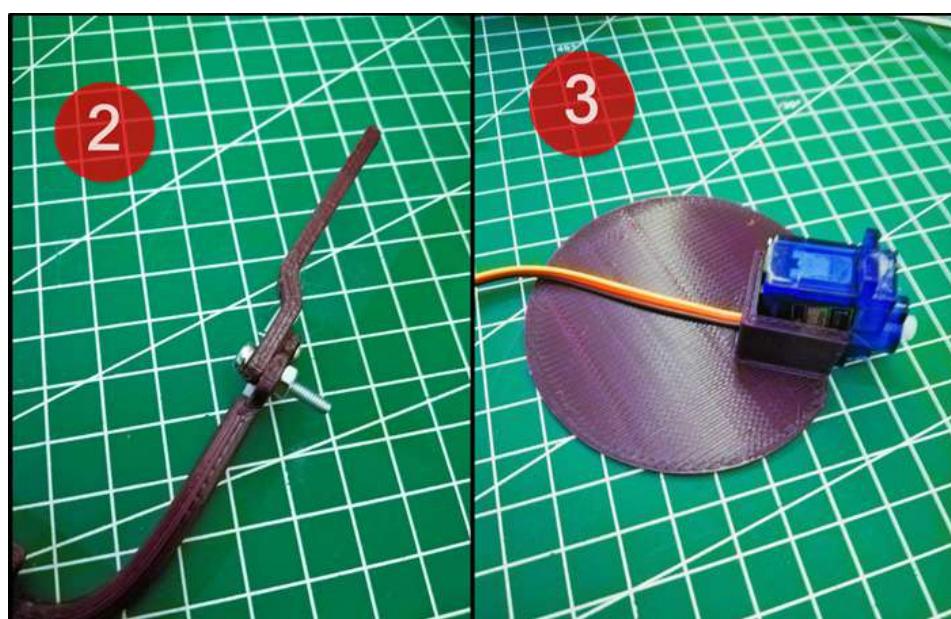
By adding an RGB LED module to the glutton piggy bank project, you can make the light turn on in the color you want every time a coin is thrown, you can add a buzzer and make a sound every time a coin is thrown. You can also print the number of coin flips on the screen by adding an OLED screen.

2.19.4. Construction Stages of the Project

You can access the original files and construction stages of the project by clicking [here](#). Unlike the project in this link, we will use the HC-SR04 ultrasonic distance sensor. You can download the updated 3D drawing files according to the HC-SR04 ultrasonic distance sensor from [this link](#) and get 3D printing.



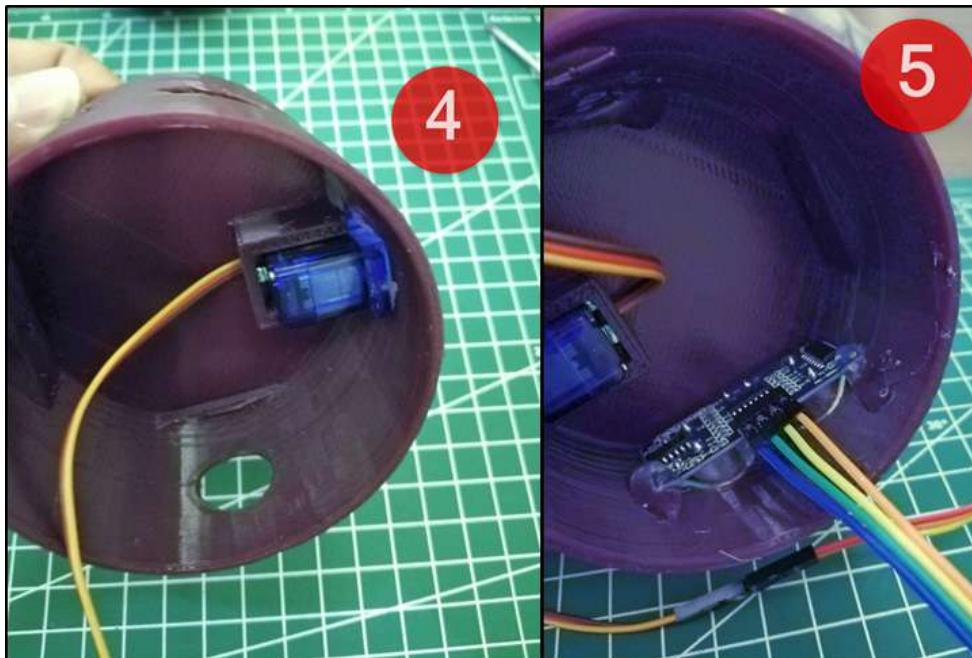
1: Fix the plastic apparatus of the servo motor to the piggy bank arm with 2 screws.



2: Fix the second part of the piggy bank arm with the M3 screw and nut to the first part where the hopper is.



3: Pass the servo motor cable and place it in its slot.



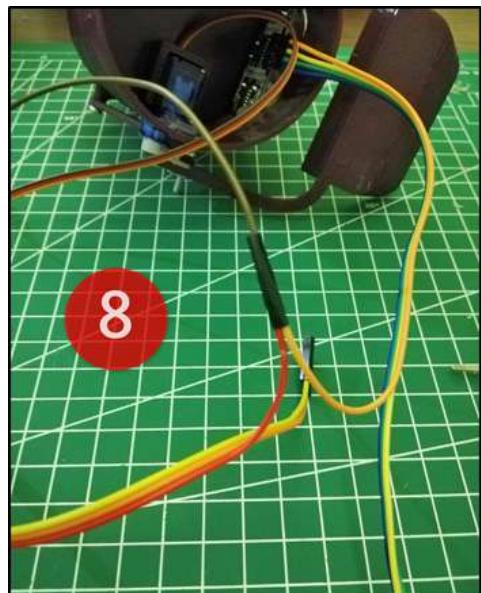
4: Place the servo motor and its housing on the body of the piggy bank. You can use hot glue here.

5: Place the ultrasonic distance sensor in the piggy bank body and fix it with hot glue.



6: Attach the piggy bank arm to the servo motor and fix it to the top cover with M3 screws.

7: Fix the piggy bank arm to the body with M2 screw.



8: Plug the cables of the servo motor and ultrasonic distance sensor and connect the power cables.

9: According to the circuit diagram, connect the cables of the servo motor and ultrasonic distance sensor to the pico.



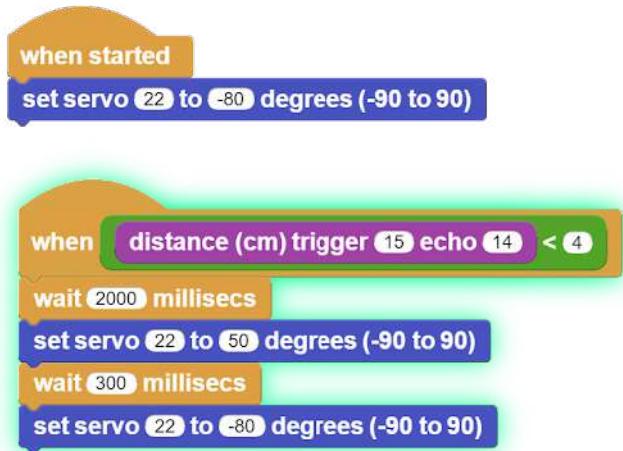
10: Plug Pico's USB cable and reassemble the cables and attach the bottom cover. That is all.

2.19.5. Coding the Project with MicroBlocks

While writing the codes, you must first edit the angle of the servo motor. In order to close the lid of the piggy bank, you must enter the angle values into the servo motor and determine the most appropriate value. Then you need to find the servo motor angle in the open position of the piggy bank. When the piggy bank is first started, the lid should be in the closed position. When the value from the ultrasonic distance



sensor is less than 5 cm, wait 2 seconds, the servo motor should work, lift the cover, and after 300 milliseconds, it should work again and bring the cover to the closed position.



[Click](#) to access the project's MicroBlocks codes.

2.19.6. MicroPython Codes of the Project

```

from machine import Pin, PWM
import utime
#define the libraries

servo=PWM(Pin(21,Pin.OUT))
trigger = Pin(15, Pin.OUT)
echo = Pin(14, Pin.IN)
#define the input and output pins

servo.freq(50)
servo.duty_u16(6750)

def getDistance():
    trigger.low()
    utime.sleep_us(2)
    trigger.high()
    utime.sleep_us(5)
    trigger.low()
    while echo.value() == 0:
        signaloff = utime.ticks_us()
    while echo.value() == 1:
        signalon = utime.ticks_us()
    timepassed = signalon - signaloff

```



```

distance = (timepassed * 0.0343) / 2
print("The distance from object is ",distance,"cm")
return distance
#calculate distance
while True:
    utime.sleep(0.01)
    if int(getDistance())<=5: #if the distance variable is less than 5
        servo.duty_u16(4010)
        utime.sleep(0.3) #wait
        servo.duty_u16(6750)
    
```

2.19.7. Arduino C Codes of the Project

```

#include <Servo.h>
#define trigPin 15
#define echoPin 14
//define the libraries
Servo servo;
void setup() {
    Serial.begin (9600);
    pinMode(trigPin, OUTPUT);
    pinMode(echoPin, INPUT);
    //define the input and output pins
    servo.attach(21); //define the servo pin
}
void loop() {
    long duration, distance;
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);
    duration = pulseIn(echoPin, HIGH);
    distance = (duration/2) / 29.1;
    //calculate distance
    if (distance < 5) { //if the distance variable is less than 5
        Serial.print(distance);
        Serial.println(" cm");
        servo.write(179);
    }
    else if (distance>5) { // if the distance variable is greater than 5
    
```

```
Serial.print(distance);
Serial.println(" cm");
servo.write(100);
}
}
```

GitHub Piggy Bank Project Page



<http://rbt.ist/bank>



2.20. NFC Smart Door

Security systems include technologies that can control authorizations at building and room entrances. Card entry systems, in which only authorized personnel can enter the operating rooms of hospitals, are one of the first examples that come to mind. In addition, the entrance doors of areas that should not be entered by persons or personnel of all levels in military security centers are equipped with card and password entry technologies. These electronic systems used in building and room entrances not only prevent the entrance of unauthorized persons, but also ensure that entry and exit information is kept under record. Password entry, card entry, fingerprint scanning, face scanning, retina scanning and voice recognition technologies are the authentication methods used in electronic entry systems.

Systems such as RFID and NFC are the basic forms of contactless payment technologies today. Although the contactless payment technology in credit cards is technically different, the working logic is the same. The maximum distance between the reader and the card is one of the features that distinguishes the technologies used from each other. When leaving the shopping stores, especially in clothing stores, NFC tags on the products will beep if they are detected to the readers at the entrance. A kind of RFID technology is used in those systems.

In this project, we will prepare a card entry system on a model house. The electronic components we will use are MFRC522 RFID reader and 13.56 Mhz cards.

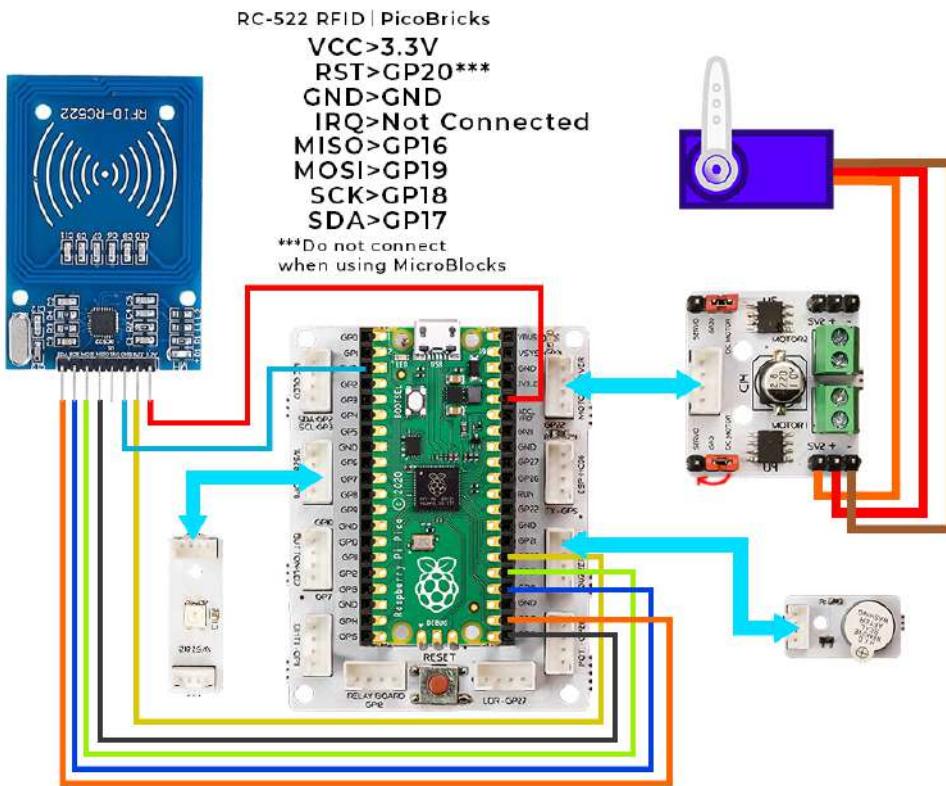
2.20.1. Project Details and Algorithm

Place the MFRC522 reader near the door of the model so that it is visible from the outside. Place the RGB LED and the buzzer on the wall where the door is visible from the outside. Picoboard can remain in the model. The entrance door of the model should be connected to the door of the servo, while the servo is set to 0 degrees, the door should be closed. You should determine the serial number of the RFID / NFC tag that will open the door, create the homeowner variable and assign the serial number to this variable.

Set the door to the closed position when PicoBricks starts. Make the buzzer beep when a card is shown to the RFID reader. If the serial number of the card being read matches the serial number in the homeowner variable, turn the RGB LED on green. Then let the door open. Make sure the door is closed 3 seconds after the door is opened. If the serial number of the card being read does not match the homeowner variable, turn the RGB LED on red. A different tone sounds from the buzzer.



2.20.2. Wiring Diagram



Be sure to make the cable connections of the RC522 RFID card reader module according to the table below.

VCC	3.3V
RST	GP20 ***
GND	GND
IRQ	Not Connected
MISO	GP16
MOSI	GP19
SCK	GP18
SDA	GP17

*** Does not connect when using MicroBlocks

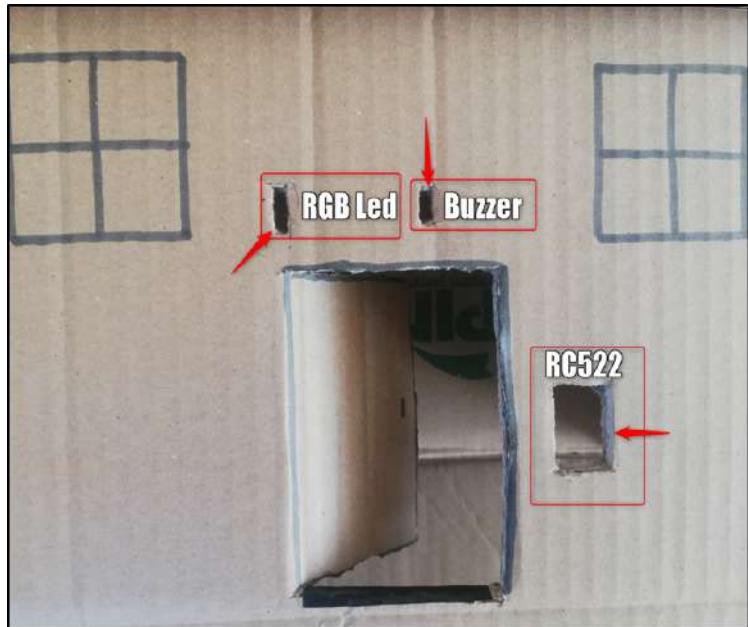
2.20.3. Project Proposal

In the automatic door project, you can give person names to RFID cards, by adding an OLED screen to the project, you can print the name of the person who read the card on the OLED screen when the card is read.

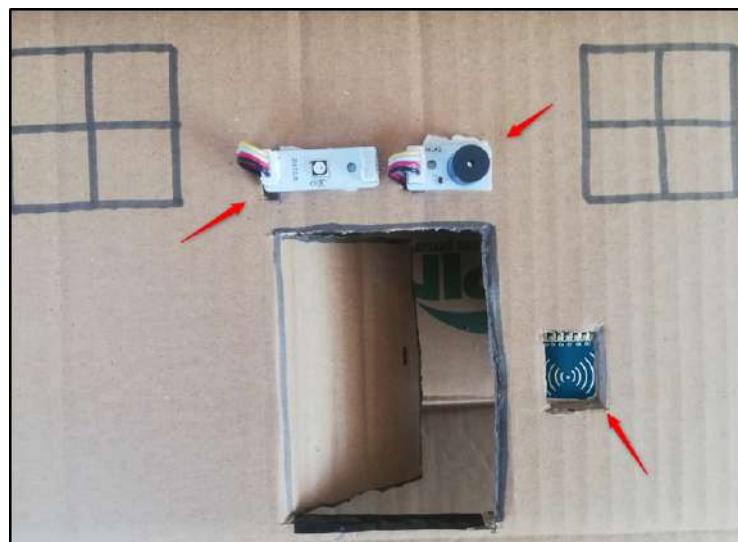


2.20.4. Construction Stages of the Project

We will make the project on the house model you used in the Smart Home project number 18. Drill holes for the RGB LED, Buzzer and RC522 RFID reader on the house model.

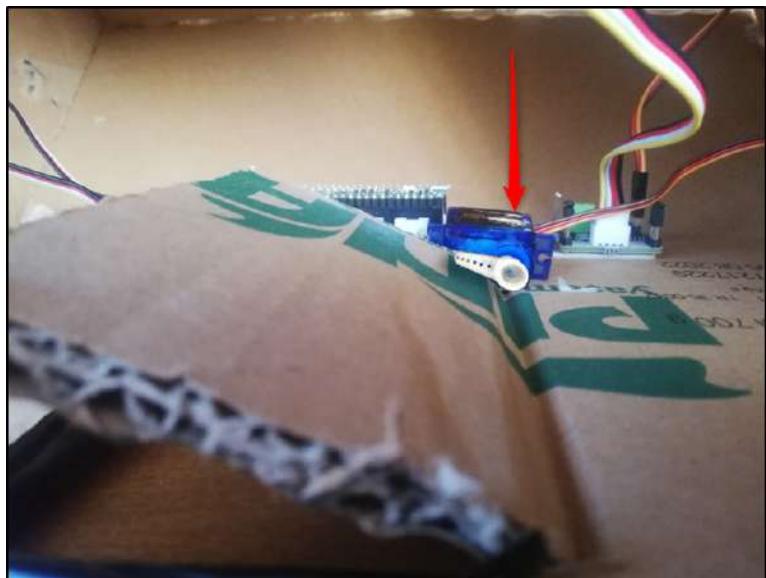


Stick double-sided foam tape on the back of the RGB LED and Buzzer and stick it on the box. Place the RC522 inside the model as in the image.

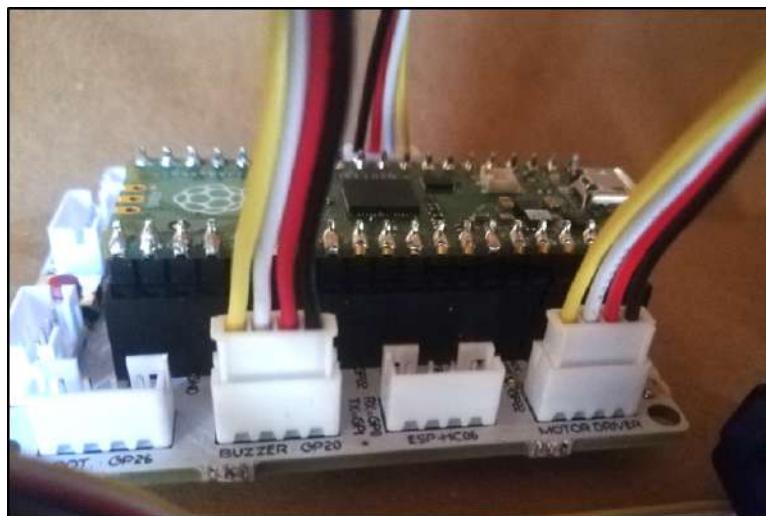




Attach the servo motor to the inside of the model with double-sided tape as a hinge in the upper left corner of the door. Attach the servo head to the door with hot glue or liquid glue.



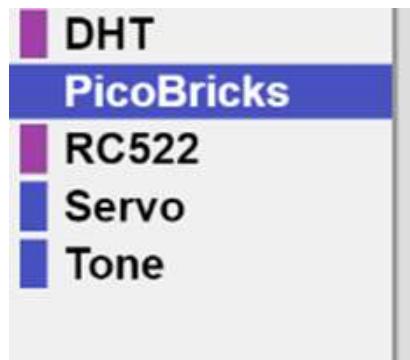
Finally, place the Pico board and the 2-key battery box inside the model house and complete the cable connections. After making the final checks of your project, it is ready to work.





2.20.5. Coding the Project with MicroBlocks

First we need to read the current user's card information. Then we will prepare the codes of the project. Create a variable named homeowner. Add RC522 and Servo libraries.



When Picobricks starts, specify the SDA pin and identify the module with the “RC522_initialize” block. Place the if else structure inside the Forever loop. When the card is read, we will print the card serial number on the screen. The “RC522 card UID” block is a 4-element list. To show or compare this value, you should use the “join items of list” block in the “Data” category.



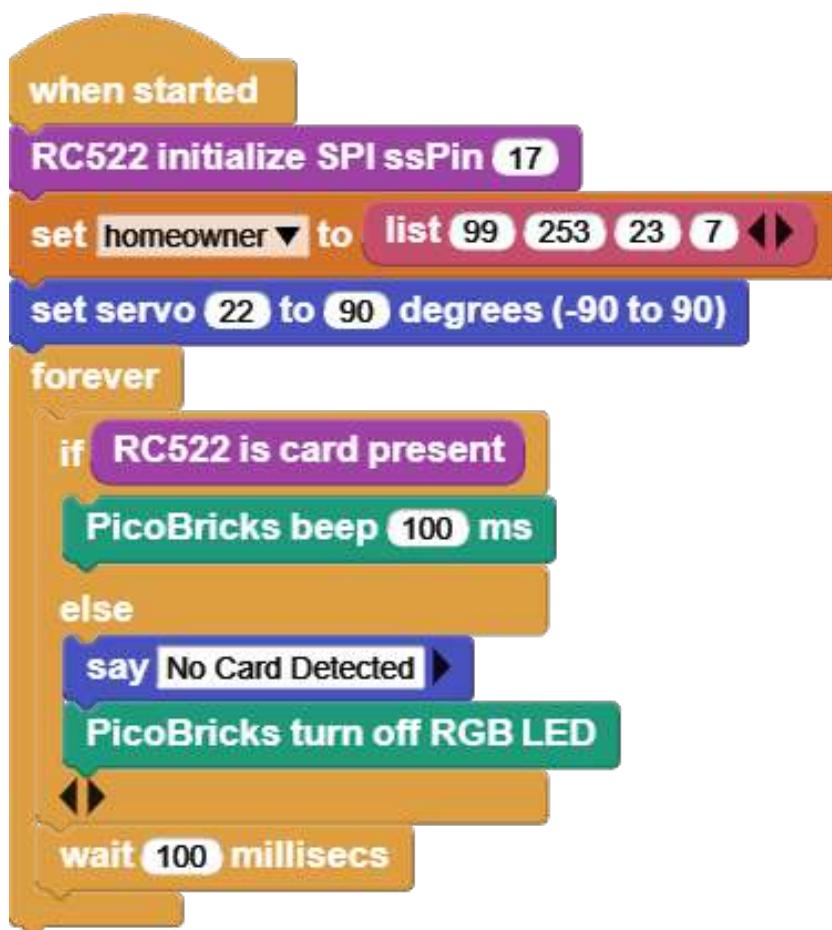
[Click](#) for MicroBlocks codes required to read the Card Serial Number. Show your card to RC522 when you run the code. Assign the card serial number above the code blocks to the homeowner variable with the “list” block in the Data category. The serial number of your card is different from the one written here.



Now we can write the codes of the project's algorithm. Define RC522 when Picobricks starts, define homeowner variable and set servo angle.



The buzzer will beep when the card is shown. In cases where the card is not shown, the statement “No Card detected” will be printed. Prepare the necessary codes as follows by placing an if else structure inside the Forever loop.



Prepare the “if else” structure that will compare the homeowner variable with the serial number of the card read right after the beep sounds. Print “Login Confirmed” if the Serial numbers match. Print “invalid user” if the serial numbers do not match. Your code should be like below.

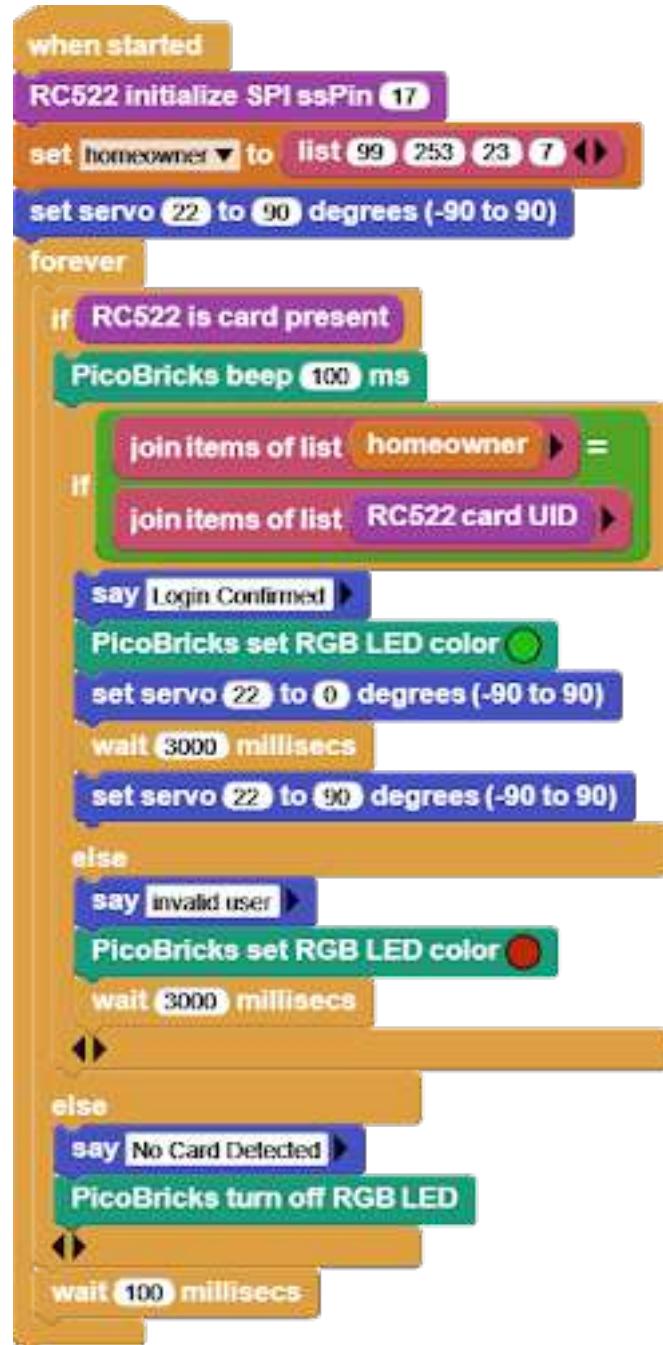


```

when started
  RC522 initialize SPI ssPin 17
  set homeowner ▾ to list 99 253 23 7 ▶▶
  set servo 22 to 90 degrees (-90 to 90)
forever
  if RC522 is card present
    PicoBricks beep 100 ms
    join items of list homeowner ▶ = if
    join items of list RC522 card UID ▶
    say Login Confirmed ▶
  else
    say invalid user ▶
  else
    say No Card Detected ▶
  wait 100 millisecs

```

Your Servo should move and the RGB LED should light up green when the correct card is shown. Wait 3 seconds and the servo should return to its original angle. The RGB LED should light up red when the wrong card is shown. Wait 3 seconds. In cases where the card is not being read, the RGB LED should be turned off. The finished codes of the project should be as follows.



[Click](#) to access the project's MicroBlocks codes.

2.20.6. MicroPython Codes of the Project

The code to be run to learn the Card ID:

```
from machine import Pin, SPI
from mfrc522 import MFRC522
```



```

import utime
#define libraries
sck = Pin(18, Pin.OUT)
mosi = Pin(19, Pin.OUT)
miso = Pin(16, Pin.OUT)
sda = Pin(17, Pin.OUT)
rst = Pin(15, Pin.OUT)
spi = SPI(0, baudrate=100000, polarity=0, phase=0, sck=sck, mosi=mosi, miso=miso)
rdr = MFRC522(spi, sda, rst)
#define MFRC522 pins

while True:
    (stat, tag_type) = rdr.request(rdr.REQIDL)
    if stat == rdr.OK:
        (stat, raw_uid) = rdr.anticoll()
        if stat == rdr.OK:
            uid = ("0x%02x%02x%02x%02x" % (raw_uid[0], raw_uid[1], raw_uid[2], raw_
            uid[3]))
            print(uid)
            utime.sleep(1)
            #read the card and give the serial number of the card

```

Project Codes:

```

from machine import I2C, Pin, SPI, PWM
from mfrc522 import MFRC522
from ws2812 import NeoPixel
from utime import sleep

servo = PWM(Pin(21))
servo.freq(50)
servo.duty_u16(1350) #servo set 0 angle 8200 for 180.

buzzer = PWM(Pin(20, Pin.OUT))
buzzer.freq(440)

neo = NeoPixel(6, n=1, brightness=0.3, autowrite=False)
RED = (255, 0, 0)
GREEN = (0, 255, 0)
BLACK = (0, 0, 0)

sck = Pin(18, Pin.OUT)
mosi = Pin(19, Pin.OUT)
miso = Pin(16, Pin.OUT)

```



```
sda = Pin(17, Pin.OUT)
rst = Pin(15, Pin.OUT)
spi = SPI(0, baudrate=100000, polarity=0, phase=0, sck=sck, mosi=mosi, miso=miso)
homeowner = "0x734762a3"
rdr = MFRC522(spi, sda, rst)
```

while True:

```
(stat, tag_type) = rdr.request(rdr.REQIDL)
if stat == rdr.OK:
    (stat, raw_uid) = rdr.anticoll()
    if stat == rdr.OK:
        buzzer.duty_u16(3000)
        sleep(0.05)
        buzzer.duty_u16(0)
        uid = ("0x%02x%02x%02x%02x" % (raw_uid[0], raw_uid[1], raw_uid[2], raw_
uid[3]))
        print(uid)
        sleep(1)
        if (uid==homeowner):
            neo.fill(GREEN)
            neo.show()
            servo.duty_u16(6000)
            sleep(3)
            servo.duty_u16(1350)
            neo.fill(BLACK)
            neo.show()

    else:
        neo.fill(RED)
        neo.show()
        sleep(3)
        neo.fill(BLACK)
        neo.show()
        servo.duty_u16(1350)
```

2.20.7. Arduino C Codes of the Project

The code to be run to learn the Card ID:

```
#include <SPI.h>
#include <MFRC522.h>
//define libraries

int RST_PIN = 26;
```



```
int SS_PIN = 17;
//define pins

MFRC522 rfid(SS_PIN, RST_PIN);

void setup()
{
    Serial.begin(9600);
    SPI.begin();
    rfid.PCD_Init();
}

void loop() {

if (!rfid.PICC_IsNewCardPresent())
    return;
if (!rfid.PICC_ReadCardSerial())
    return;
rfid.uid.uidByte[0] ;
rfid.uid.uidByte[1] ;
rfid.uid.uidByte[2] ;
rfid.uid.uidByte[3] ;
printid();
rfid.PICC_HaltA();
//Reading your ID.
}
void printid()
{
    Serial.print("Your ID: ");
    for (int x = 0; x < 4; x++) {
        Serial.print(rfid.uid.uidByte[x]);
        Serial.print(" ");
    }
    Serial.println("");
}
```

Project Codes:

```
#include <SPI.h>
#include <MFRC522.h>
#include <Servo.h>
#include <Adafruit_NeoPixel.h>
```



//Define libraries.

```
#define RST_PIN 26
#define SS_PIN 17
#define servoPin 22
#define PIN 6
#define NUMPIXELS 1
#define buzzer 20
//define pins of servo,buzzer,neopixel and rfid.

Adafruit_NeoPixel pixels(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);
Servo motor;
MFRC522 rfid(SS_PIN, RST_PIN);

byte ID[4] = {"Write your own ID."};

void setup() {
    pixels.begin();
    motor.attach(servoPin);
    Serial.begin(9600);
    SPI.begin();
    rfid.PCD_Init();
    pinMode(buzzer, OUTPUT);

}

void loop()
{
    pixels.clear();

    if ( ! rfid.PICC_IsNewCardPresent())
        return;
    if ( ! rfid.PICC_ReadCardSerial())
        return;

    if
    (
        rfid.uid.uidByte[0] == ID[0] &&
        rfid.uid.uidByte[1] == ID[1] &&
        rfid.uid.uidByte[2] == ID[2] &&
        rfid.uid.uidByte[3] == ID[3] )
    {
        Serial.println("Door Opened.");
    }
}
```



```
printid();
tone(buzzer,523);
delay(200);
noTone(buzzer);
delay(100);
tone(buzzer,523);
delay(200);
noTone(buzzer);
pixels.setPixelColor(0, pixels.Color(0, 250, 0));
delay(200);
pixels.show();
pixels.setPixelColor(0, pixels.Color(0, 0, 0));
delay(200);
pixels.show();
motor.write(180);
delay(2000);
motor.write(0);
delay(1000);

//RGB LED turns green and the door opens thanks to the servo motor if the correct
card is read to the sensor.

}

else
{
    Serial.println("Unknown Card.");
    printid();
    tone(buzzer,494);
    delay(200);
    noTone(buzzer);
    delay(100);
    tone(buzzer,494);
    delay(200);
    noTone(buzzer);
    pixels.setPixelColor(0, pixels.Color(250, 0, 0));
    delay(100);
    pixels.show();
    pixels.setPixelColor(0, pixels.Color(0, 0, 0));
    delay(100);
    pixels.show();
    //RGB LED turns red and the door does not open if the wrong card is read to the
sensor

}
rfid.PICC_HaltA();

}

void printid()
```



```
{  
Serial.print("ID Number: ");  
for(int x = 0; x < 4; x++){  
    Serial.print(rfid.uid.uidByte[x]);  
    Serial.print(" ");  
}  
Serial.println("");  
}
```

GitHub NFC Smart Door Project Page



<http://rbt.ist/door>



2.21. Automatic Trash Bin

The Covid 19 pandemic has changed people's daily routines in many areas. In many areas such as cleaning, working, shopping and social life, people were introduced to a series of new rules that they had to comply with. Covid-19 has led to the development of new business areas as well as some products to stand out. At a time when hand hygiene was very important, no one wanted to touch the lid of the trash can to throw away their garbage.

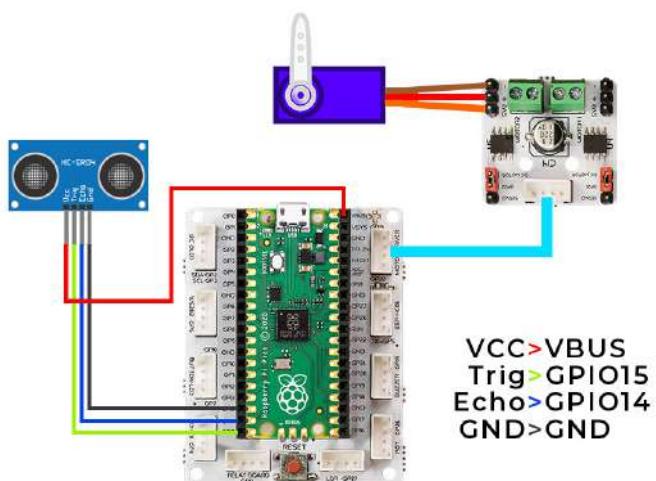
When approached, the lids of which open automatically and when it is full, the trash bins, which make bags ready to be thrown away, found buyers at prices far above their cost. In addition, automatic disinfectant machines provided contactless hygiene by pouring a certain amount of liquid into our palms when we held them under our hands. Automatic disinfectant machines took place on the shelves at prices well above their cost. These two products have similarities in terms of working system. In automatic disinfectant machines, a pump with an electric motor directly transfers the liquid, and some models have devices based on the pumping system with the power of the servo motor. In automatic trash bins, a servo motor that opens the lid was used, and infrared or ultrasonic sensors were used to detect hand movement.

In this project, you will make a mobile and automatic stylish trash bin for your room using an ultrasonic sensor and servo motor with PicoBricks.

2.21.1. Project Details and Algorithm

HC-SR04 ultrasonic distance sensor and SG90 servo motor will be used in this project. When the user puts his hand in front of the lid of the trash can, the distance sensor will detect the proximity and send it to the Picobricks. According to this information, Picobricks will open the lid of the garbage can by running a servo motor and will lower it again after a short while.

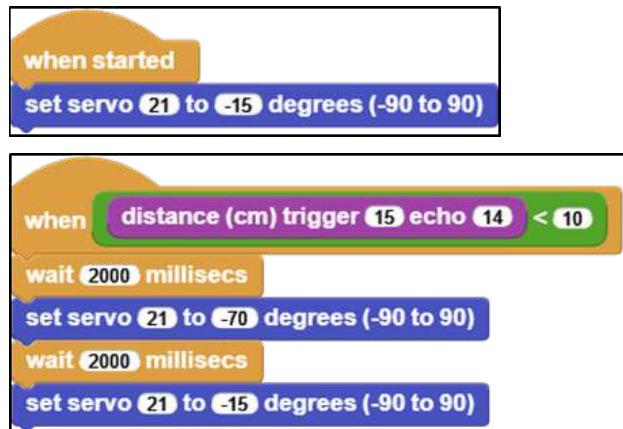
2.21.2. Wiring Diagram





2.21.3. Coding the Project with MicroBlocks

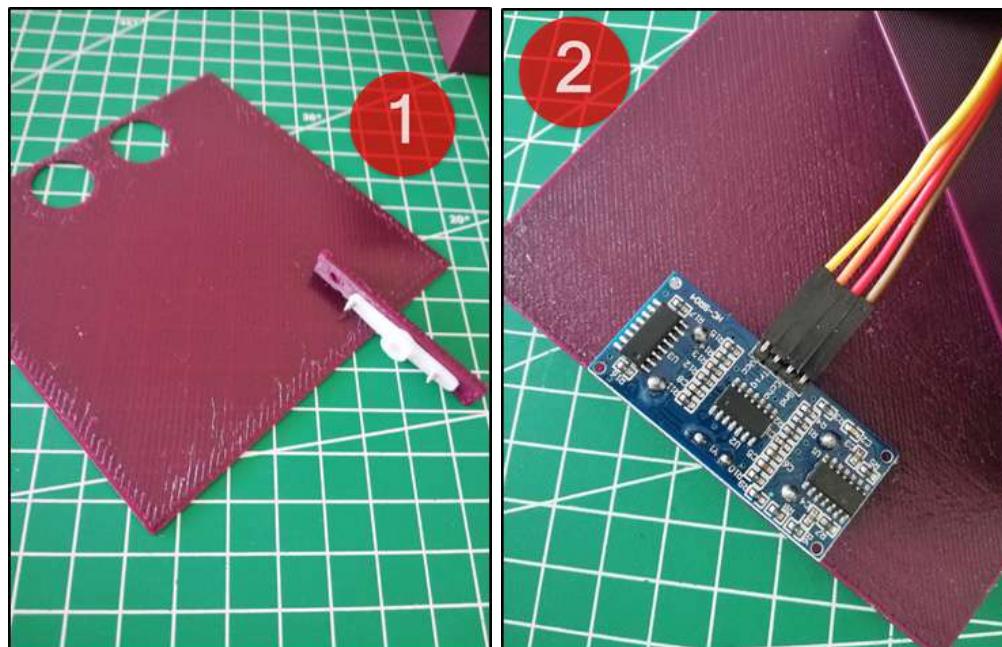
While writing the codes, you must first edit the angle of the servo motor. In order to bring the lid of the garbage can to the closed position, you must enter the angle values to the servo motor and determine the most appropriate value. Then you need to find the servo motor angle in the open position of the trash can. The lid must be in the closed position when the dustbin is first started. When the value from the ultrasonic distance sensor is less than 10 cm, wait 2 seconds, the servo motor should work, lift the cover, and after 2 seconds, it should work again and bring the cover to the closed position.



[Click](#) to access the project's MicroBlocks codes.

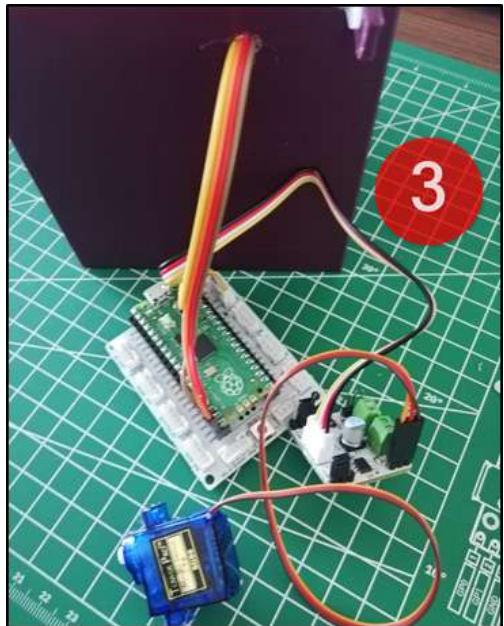
2.21.4. Construction Stages of the Project

You can download the 3D drawing files of the project from [this link](#) and get 3D printing.



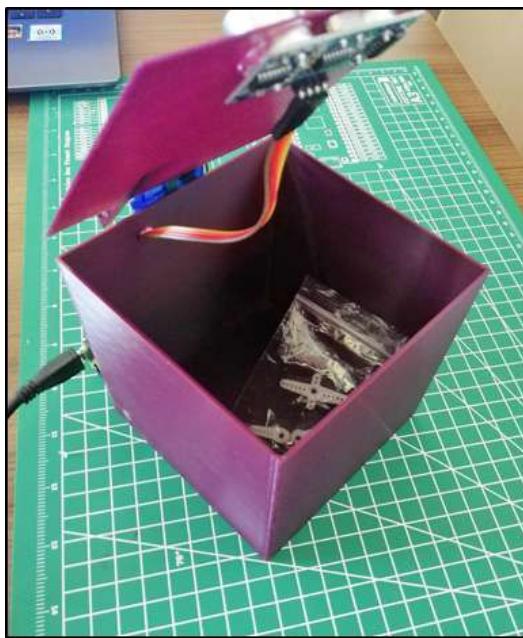
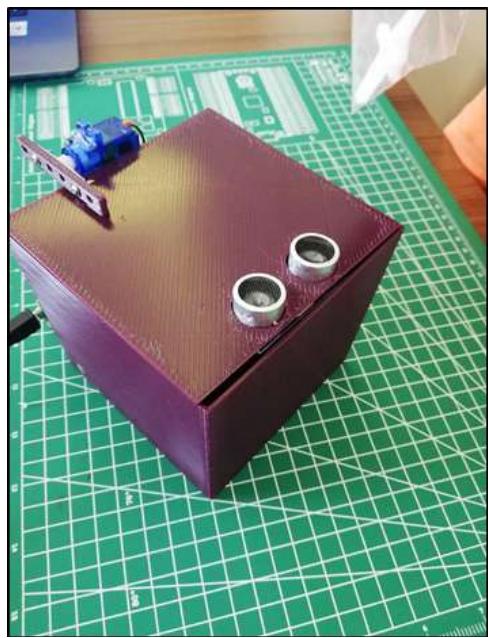


- 1: Fix it by screwing it to the trash bin cover of the servo motor apparatus.
- 2: Fix the ultrasonic distance sensor on the lid of the trash bin with the hot glue.



3: Pass the cables of the ultrasonic distance sensor through the hole in the box and connect them to the pins shown in the picobricks circuit diagram, make the servo motor and motor driver connections.

4: Fix the servo motor, picobricks and motor driver parts to the box with hot glue.



If everything went well, when you put your hand close to the garbage can, the lid of the bucket will open and it will close again after you throw the garbage away.



2.21.5. Project Proposal

As in this project, where we automate a trash can in our house using sensors and motors, you can have a drawer or a cabinet door open automatically with the help of sensors and motors.

2.21.6. MicroPython Codes of the Project

```
from machine import Pin, PWM
from utime import sleep

servo=PWM(Pin(21,Pin.OUT))
trigger = Pin(15, Pin.OUT)
echo = Pin(14, Pin.IN)

servo.freq(50)
servo.duty_u16(1920) #15 degree

def getDistance():
    trigger.low()
    utime.sleep_us(2)
    trigger.high()
    utime.sleep_us(5)
    trigger.low()
    while echo.value() == 0:
        signaloff = utime.ticks_us()
    while echo.value() == 1:
        signalon = utime.ticks_us()
    timepassed = signalon - signaloff
    distance = (timepassed * 0.0343) / 2
    print("The distance from object is ",distance,"cm")
    return distance

while True:
    sleep(0.01)
    if int(getDistance())<=10:
        servo.duty_u16(4010) #70 degree
        utime.sleep(0.3)
        servo.duty_u16(1920)
```



2.21.7. Arduino C Codes of the Project

```
#include <Servo.h>
#define trigPin 14
#define echoPin 15
Servo servo;
void setup() {
    Serial.begin (9600);
    pinMode(trigPin, OUTPUT);
    pinMode(echoPin, INPUT);
    servo.attach(21);
}

void loop() {
    long duration, distance;
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);
    duration = pulseIn(echoPin, HIGH);
    distance = (duration/2) / 29.1;
    if (distance < 80) {
        Serial.print(distance);
        Serial.println(" cm");
        servo.write(179);
    }
    else if (distance<180) {
        Serial.print(distance);
        Serial.println(" cm");
        servo.write(100);
    }
}
```



GitHub Automatic Trash Bin Project Page



<http://rbt.ist/bin>



2.22. Digital Ruler

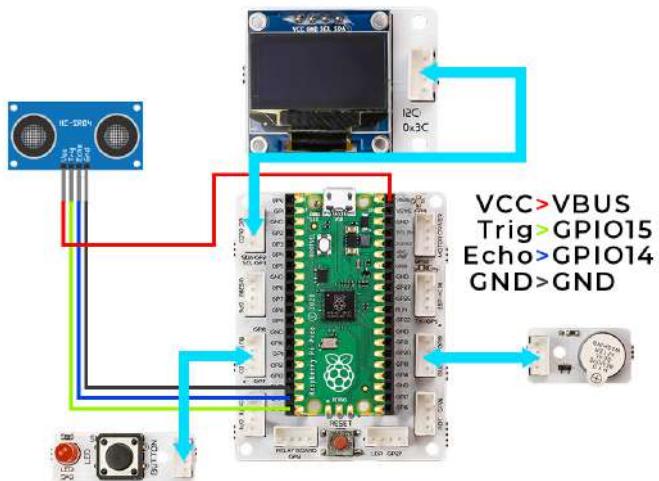
Many tools are used to measure length. At the beginning of these tools are rulers. Our measuring instrument differs according to the place and size to measure. Tape measures are used in construction and architecture, and calipers are used for small objects that require millimeter precision. In addition, if it is desired to measure an area that needs both large and precise measurement, distance meters working with laser and infrared systems are used. Ultrasonography devices used in the health sector also work with the same logic, but convert their measurements into visuals.

In our project, we will use PicoBricks and an ultrasonic sensor to prepare a digital ruler that will display the distance value on the OLED screen when the button is pressed. Ultrasonic sensors detect distance according to the return times of the sound waves they emit.

2.22.1. Project Details and Algorithm

When Picobricks starts, instructions are displayed on the OLED screen. After the user presses the button, 20 measurements are made at 50 millisecond intervals for 1 second and the average is taken. The red LED stays on during the measurement, and the red LED turns off when the measurement is complete. The average value is added to the distance from the tip of the sensor to the back of the box. The last distance value is displayed on the OLED display.

2.22.2. Wiring Diagram





2.22.3. Coding the Project with MicroBlocks

Start MicroBlocks and connect PicoBricks. Add Tone, Distance, OLED Graphics libraries. Create variables named distance to calculate the distance, measure so that you can average the measurement value. Turn on the red LED to let you know that the measurement has started when the button is pressed. After a short beep, make 20 measurements at 50 millisecond intervals and assign the average to the distance value. Broadcast go to OLED to turn off the red LED and print the result on the OLED screen.

```

when PicoBricks button
  set [measure v] to [0]
  PicoBricks set red LED [red]
  PicoBricks beep [50 ms]
  repeat (20)
    change [measure v] by [distance (cm) trigger 14 echo 15]
    wait [50 millisecs]
    set [distance v] to ([measure v] / 20)
  end
  PicoBricks set red LED [off]
  broadcast [go to OLED]
end

```

Add the size of your box to the value measured by the sensor while printing the distance variable on the screen. Since the thickness of the box we used in our project is 6 cm, we added 6 cm to the distance variable and printed it on the screen. Make a low sound to notify the user that the measurement is complete.

```

when [go to OLED] received
  initialize i2c [OLED_0.96in v] address(hex) [3C] reset pin# [0] flip [red]
  write >Digital Ruler< at x [3] y [5] inverse [red]
  write [join [Distance: [distance + 6] cm] at x [15] y [32] inverse [red]]
  play note [C] octave [2] for [50] ms
end

```

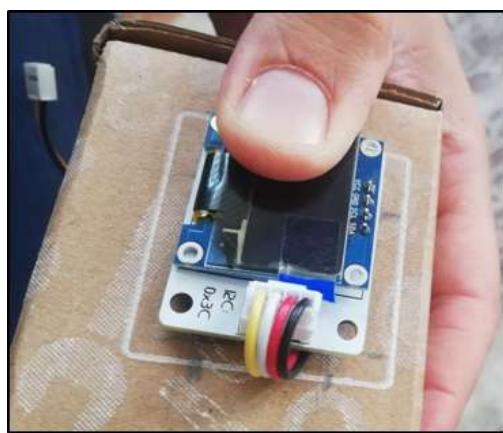
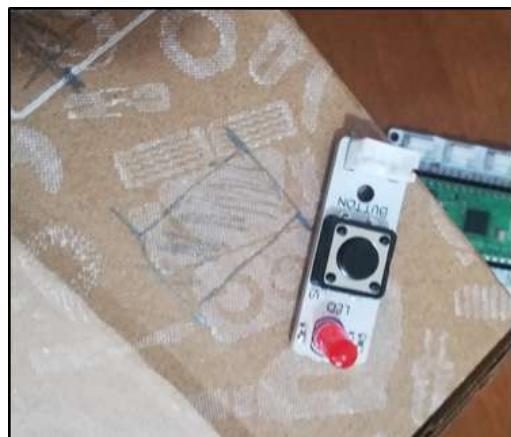
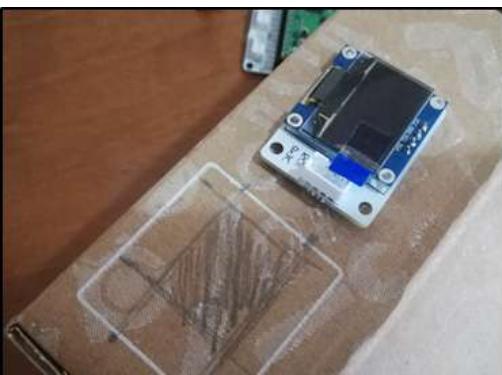
[Click](#) to access the project's MicroBlocks codes.



2.22.4. Construction Stages of the Project

To prepare the project, you need double-sided foam tape, a utility knife, a waste cardboard box of approximately 15x10x5 cm.

1. Cut the holes for the ultrasonic sensor, OLED screen, button LED module, buzzer, battery box to pass the cables with a utility knife.





2. Hang all the cables inside the box and stick the back of the modules to the box with double-sided foam tape. Connect the trig pin of the ultrasonic sensor to the GPIO14 pin and the echo pin to the GPIO15 pin. You should connect the VCC pin to the VBUS pin on the Picoboard.



3. After completing the cable connections of all modules, you can insert the 2-battery box into the power jack of the Picoboard and turn on the switch. That's it for the digital ruler project!





2.22.5. Project Proposal

You can turn the digital ruler into a height meter by fixing it to the wall or ceiling. Since the height meter will be fixed on the wall or ceiling, it will not be possible to measure by pressing the button. For this, you can add the HC05 bluetooth module to the project and have it measure when a command is received from the mobile application.

2.22.6. MicroPython Codes of the Project

```
from machine import Pin, PWM, I2C
from utime import sleep
from picobricks import SSD1306_I2C
import utime
#define the libraries
redLed=Pin(7,Pin.OUT)
button=Pin(10,Pin.IN,Pin.PULL_DOWN)
buzzer=PWM(Pin(20,Pin.OUT))
buzzer.freq(392)
trigger = Pin(15, Pin.OUT)
echo = Pin(14, Pin.IN)
#define input and output pins
WIDTH = 128
HEIGHT = 64
#OLED screen settings
sda=machine.Pin(4)
scl=machine.Pin(5)
i2c=machine.I2C(0,sda=sda, scl=scl, freq=1000000)
#initialize digital pin 4 and 5 as an OUTPUT for OLED communication
oled = SSD1306_I2C(128, 64, i2c)
measure=0
finalDistance=0

def getDistance():
    trigger.low()
    utime.sleep_us(2)
    trigger.high()
    utime.sleep_us(5)
    trigger.low()
    while echo.value() == 0:
        signaloff = utime.ticks_us()
```



```

while echo.value() == 1:
    signalon = utime.ticks_us()
timepassed = signalon - signaloff
distance = (timepassed * 0.0343) / 2
return distance
#calculate the distance
def getMeasure(pin):
    global measure
    global finalDistance
    redLed.value(1)
    for i in range(20):
        measure += getDistance()
        sleep(0.05)
    redLed.value(0)
    finalDistance = (measure/20) + 1
    oled.fill(0)
    oled.show()
    oled.text(">Digital Ruler<", 2,5)
    oled.text("Distance " + str(round(finalDistance)) +" cm", 0, 32)
    oled.show()
#print the specified distance to the specified x and y coordinates on the OLED
screen
print(finalDistance)
buzzer.duty_u16(4000)
sleep(0.05)
buzzer.duty_u16(0)
measure=0
finalDistance=0
#sound the buzzer
button.irq(trigger=machine.Pin.IRQ_RISING, handler=getMeasure)

```

2.22.7. Arduino C Codes of the Project

```

#include <Wire.h>
#include "ACROBOTIC_SSD1306.h"
#include <NewPing.h>
// define the libraries
#define TRIGGER_PIN 15
#define ECHO_PIN 14
#define MAX_DISTANCE 400

```



```
NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE);

#define T_B 493

int distance = 0;
int total = 0;

void setup() {
    pinMode(7,OUTPUT);
    pinMode(20,OUTPUT);
    pinMode(10,INPUT);
    // define input and output pins
    Wire.begin();
    oled.init();
    oled.clearDisplay();

}

void loop() {

    delay(50);
    if(digitalRead(10) == 1){

        int measure=0;
        digitalWrite(7,HIGH);
        tone(20,T_B);
        delay(500);
        noTone(20);

        for (int i=0;i<20;i++){

            measure=sonar.ping_cm();
            total=total+measure;
            delay(50);
        }

        distance = total/20+6; // calculate the distance
        digitalWrite(7,LOW);

        delay(1000);

    }
}
```



```
oled.clearDisplay();
oled.setTextXY(2,1);
oled.putString(">Digital Ruler<");
oled.setTextXY(5,1);
oled.putString("Distance: ");
oled.setTextXY(5,10);
String string_distance=String(distance);
oled.putString(string_distance);
oled.setTextXY(5,12);
oled.putString("cm"); // print the calculated distance on the OLED
screen

measure=0;
distance=0;
total=0;
}
}
```

GitHub Digital Ruler Project Page



<http://rbt.ist/ruler>



2.23. Air Piano

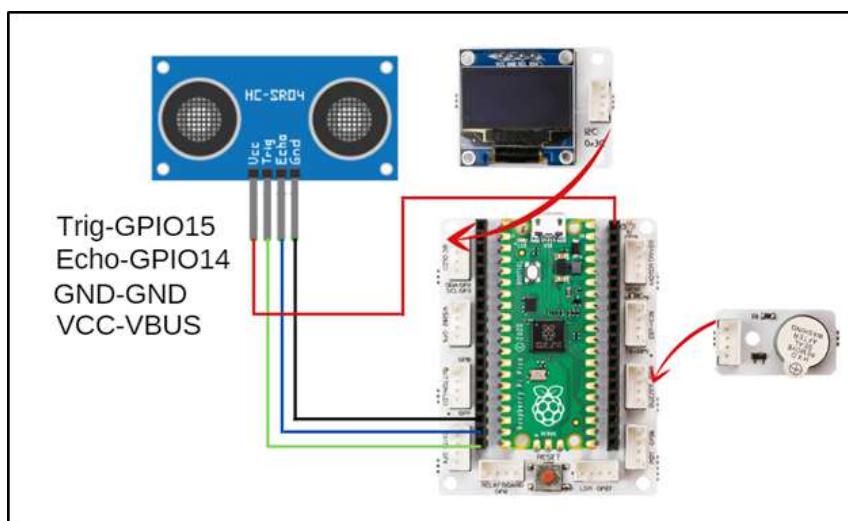
With the development of electronic technology, musical instruments that are difficult to produce, expensive and producing high-quality sound have been digitized. Pianos are one of these instruments. Each key of digital pianos produces electrical signals at a different frequency. Thus, it can play 88 different notes from its speakers. Factors such as the delay time of the keys of digital instruments, the quality of the speaker, the resolution of the sound have appeared as the factors affecting the quality. In electric guitars, vibrations in strings are digitized instead of keys. On the other hand, in wind instruments, the notes played can be converted into electrical signals and recorded thanks to the high-resolution microphones plugged into the sound output. This development in electronic technology has facilitated access to high-cost musical instruments, music education has gained a wider variety and spread to a wider audience.

In this project we will make a simple piano that can play 8 notes with PicoBricks. The speaker of this piano will be the buzzer. The ultrasonic sensor will act as the keys of the piano.

2.23.1. Project Details and Algorithm

In this project, we will make a piano application using the HC-SR04 Ultrasonic distance sensor and the buzzer module on PicoBricks. We will make the buzzer play different notes according to the values coming from the distance sensor, and we will create melodies by moving our hand closer to the sensor and away from it. In addition, we will instantly print the distance played note information on the OLED screen.

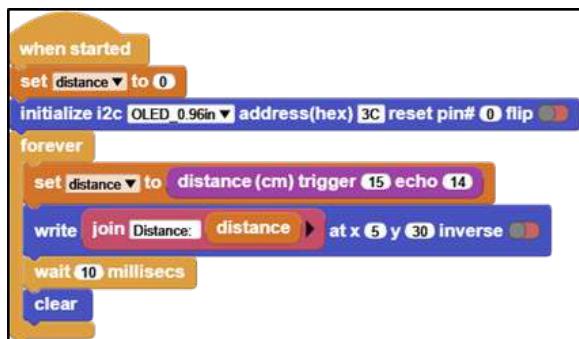
2.23.2. Wiring Diagram





2.23.3. Coding the Project with MicroBlocks

When we start to write the code of the project with MicroBlocks, we first need to import the Tone library, the OLED Graphics library in the Graphics category and the Distance libraries in the Sensing category by clicking the Add Library button. After adding the libraries, we must define the pins for the HC-SR04 module and create a variable called distance to receive and process the data from the sensor, and write the necessary codes to transfer the sensor information to the variable and show it on the OLED screen.



Each note has a letter equivalent. The tone library also contains blocks that we can run by typing the letter equivalents of the notes.



We should write the letter equivalents of the notes we want to play in the field in the Play block. By comparing the distance values in the distance variable, you can run the project by changing the octave value in the play block as you wish, after writing the necessary codes for the notes to change in sequence at intervals of 5 cm.



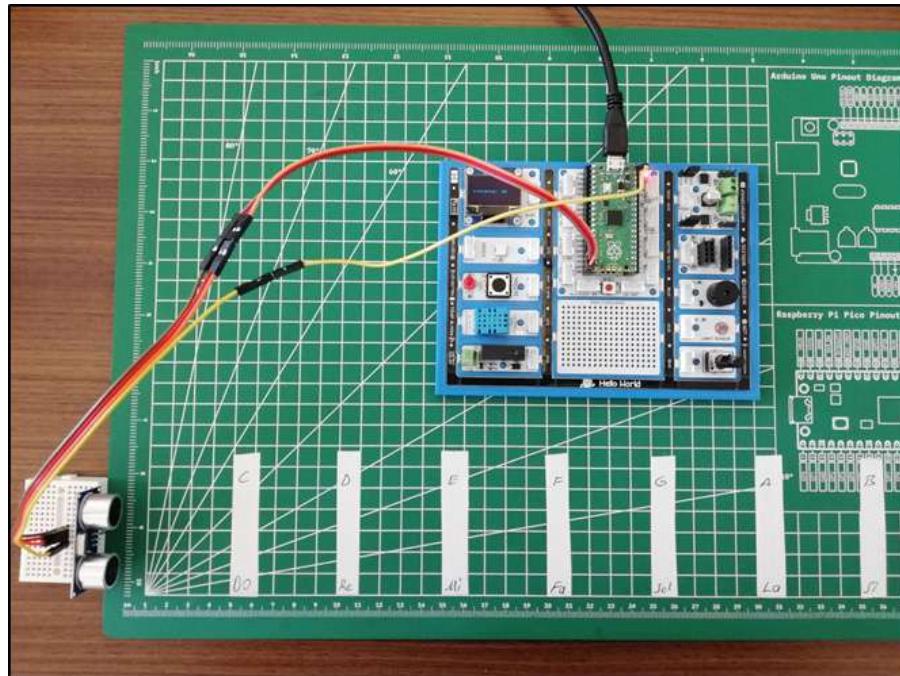
```

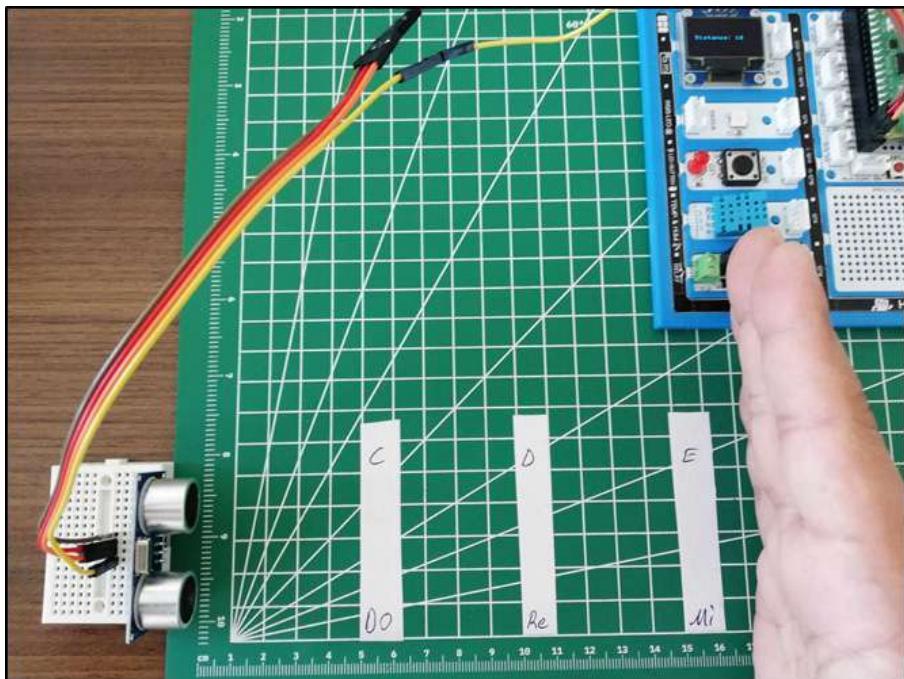
when started
forever
if distance > 5 and distance < 11
    play note C octave 1 for 400 ms
else if distance > 10 and distance < 16
    play note D octave 1 for 400 ms
else if distance > 15 and distance < 21
    play note E octave 1 for 400 ms
else if distance > 20 and distance < 26
    play note F octave 1 for 400 ms
else if distance > 25 and distance < 31
    play note G octave 1 for 400 ms
else if distance > 30 and distance < 36
    play note A octave 1 for 400 ms
else if distance > 35 and distance < 41
    play note B octave 1 for 400 ms
else
    //<>

```

[Click](#) to access the project's MicroBlocks codes.

2.23.4. Construction Stages of the Project





2.23.5. Project Proposal

There is one instant button on PicoBricks. By connecting 7 buttons to Pico, you can make different notes play when each key is pressed, you can use buttons for octave value and beat times, and you can develop your piano project.

2.23.6. MicroPython Codes of the Project

```

from machine import Pin, PWM, I2C
from utime import sleep
import utime
from picobricks import SSD1306_I2C
import _thread
#define the libraries

buzzer=PWM(Pin(20,Pin.OUT))
trigger = Pin(15, Pin.OUT)
echo = Pin(14, Pin.IN)
#define the input and Output pins

WIDTH = 128
HEIGHT = 64
#OLED screen settings

sda=machine.Pin(4)

```



```
scl=machine.Pin(5)
i2c=machine.I2C(0,sda=sda, scl=scl, freq=1000000)
#initialize digital pin 4 and 5 as an OUTPUT for OLED communication

oled = SSD1306_I2C(WIDTH, HEIGHT, i2c)

measure=0

def getDistance():
    trigger.low()
    utime.sleep_us(2)
    trigger.high()
    utime.sleep_us(5)
    trigger.low()
    while echo.value() == 0:
        signaloff = utime.ticks_us()
    while echo.value() == 1:
        signalon = utime.ticks_us()
    timepassed = signalon - signaloff
    distance = (timepassed * 0.0343) / 2
    return distance
#calculate distance

def airPiano():
    while True:
        global measure

        if measure>5 and measure<11:
            buzzer.duty_u16(4000)
            buzzer.freq(262)
            sleep(0.4)

        elif measure>10 and measure<16:
            buzzer.duty_u16(4000)
            buzzer.freq(294)
            sleep(0.4)

        elif measure>15 and measure<21:
            buzzer.duty_u16(4000)
            buzzer.freq(330)
            sleep(0.4)
```



```

        elif measure>20 and measure<26:
            buzzer.duty_u16(4000)
            buzzer.freq(349)
            sleep(0.4)

        elif measure>25 and measure<31:
            buzzer.duty_u16(4000)
            buzzer.freq(392)
            sleep(0.4)

        elif measure>30 and measure<36:
            buzzer.duty_u16(4000)
            buzzer.freq(440)
            sleep(0.4)

        elif measure>35 and measure<41:
            buzzer.duty_u16(4000)
            buzzer.freq(494)
            sleep(0.4)
        else:
            buzzer.duty_u16(0)

_thread.start_new_thread(airPiano, ())
#play the tone determined by the value of the distance sensor

while True:
    measure=int(getDistance())
    oled.text("Distance " + str(measure)+ " cm", 5,30)
    oled.show()
    sleep(0.01)
    oled.fill(0)
    oled.show()
#write the specified texts to the determined x and ye coordinates on the OLED
screen

```

2.23.7. Arduino C Codes of the Project

```

#include <Wire.h>
#include "ACROBOTIC_SSD1306.h"
#include <NewPing.h>

```



```
#define TRIGGER_PIN 15
#define ECHO_PIN 14
#define MAX_DISTANCE 400

NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE);

#define T_C 262
#define T_D 294
#define T_E 330
#define T_F 349
#define T_G 392
#define T_A 440
#define T_B 493

const int Buzzer = 20;

void setup() {
    pinMode(Buzzer,OUTPUT);

    Wire.begin();
    oled.init();
    oled.clearDisplay();

#if defined(__AVR_ATtiny85__) && (F_CPU == 16000000)
    clock_prescale_set(clock_div_1);
#endif
}

void loop() {

    delay(50);
    int distance=sonar.ping_cm();

    if(distance>5 & distance<11)
    {
        tone(Buzzer,T_C);
    }

    else if(distance>10 & distance<16)
    {
        tone(Buzzer,T_D);
    }
}
```



```
}

else if(distance>15 & distance<21)
{
    tone(Buzzer,T_E);
}

else if(distance>20 & distance<26)
{
    tone(Buzzer,T_F);
}

else if(distance>25 & distance<31)
{
    tone(Buzzer,T_G);
}

else if(distance>30 & distance<36)
{
    tone(Buzzer,T_A);
}

else if(distance>35 & distance<41)
{
    tone(Buzzer,T_B);
}

else
{
    noTone(Buzzer);
}

oled.clearDisplay();
oled.setTextXY(2,4);
oled.putString("Distance: ");
oled.setTextXY(4,6);
String string_distance=String(distance);
oled.putString(string_distance);
oled.setTextXY(4,8);
oled.putString("cm");
}
```



GitHub Air Piano Project Page



<http://rbt.ist/piano>



2.24. Maze Solver Robot

Coding education is as old as the history of programming languages. Today, different products are used to popularize coding education and make it exciting and fun. The first of these is educational robots. Preparing and coding robots improves children's engineering and coding skills. Robotics competitions are organized by institutions and organizations to popularize coding education and encourage teachers and students. One of these competitions is the Maze Solver Robot competitions. These robots firstly learn the destination by wandering around the maze and return to the starting point. Then, when they start the labyrinth again, they try to reach their destination in the shortest way possible. Robots use distance sensors while learning about the maze. Infrared or ultrasonic sensors are used in these robots.

Smart robot vacuums used in homes and workplaces also work with logic close to the algorithms of maze-solver robots. Thanks to their algorithms that constantly check and map the obstacles, they try to do it completely and without crashing. Most of the smart vacuums are equipped with LIDAR and infrared sensors, which make high-precision laser measurements for distance measurement and obstacle detection.

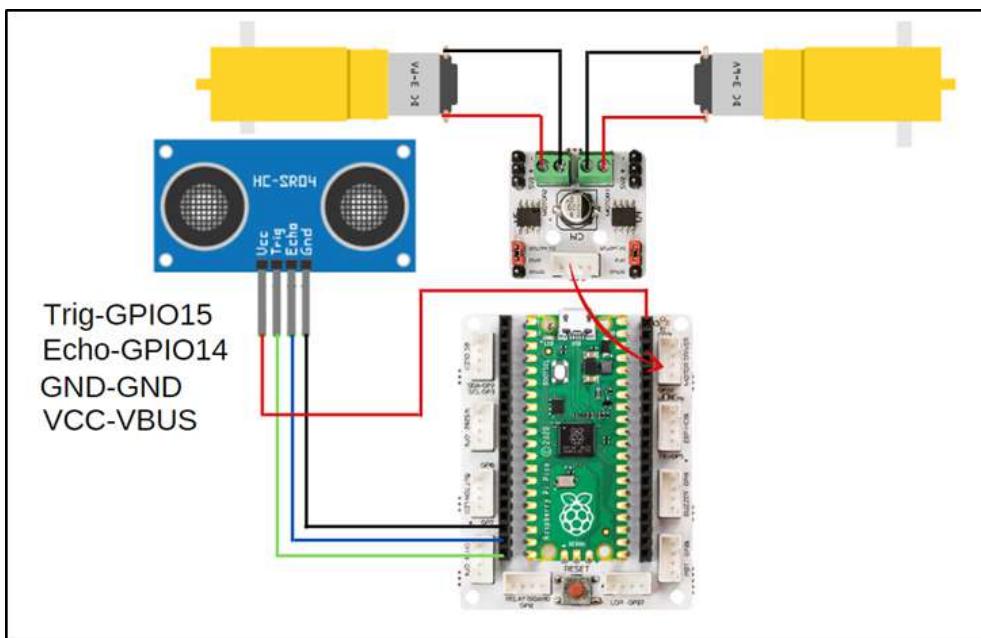
In this project, we will make a simple robot with PicoBricks that you can prepare for maze solver robot competitions.

2.24.1. Project Details and Algorithm

In the maze solving robot project, we will use the 2WD robot car kit that comes out of the set. We will use the HC-SR04 ultrasonic distance sensor so that the robot can detect the distance in front of it and decide its movements on its own. In the maze, the robot will detect the distance in front of the car and move forward if it is empty. If the distance is less than 5 cm, the car will turn right, measure the distance again, if the distance on the right is greater than 5 cm, it will continue on its way, if it is less, it will turn left and move forward. In this way, by turning right and left, we will enable the vehicle to move forward and exit the maze through the empty roads in the maze.

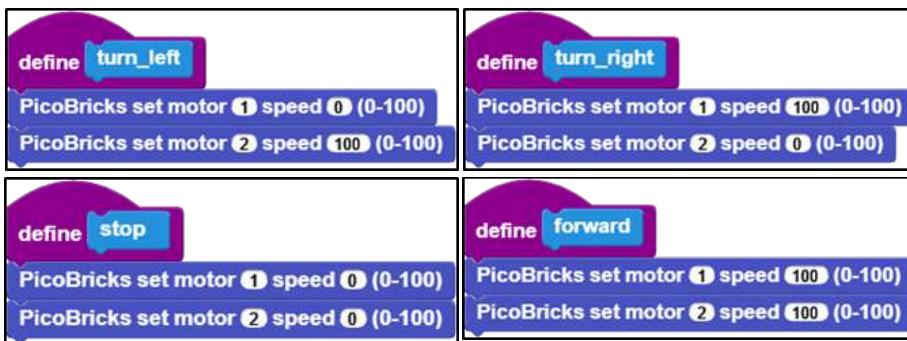


2.24.2. Wiring Diagram



2.24.3. Coding the Project with MicroBlocks

We can start to write the codes of the project by writing the necessary codes for the robot car to make its movements. For this, we must define four blocks (functions) named forward, turn right, turn left and stop. In these functions, we will place the necessary blocks for the motors to rotate.



After defining the functions, we add the Distance extension from the Sensing category and write the trig and echo pin numbers of the HC-SR04 ultrasonic distance sensor into the distance block and create a variable named distance, so that the value of this variable is the value from the sensor. Then, if the sensor value is less than 5 cm, we make the car stop and turn right. If the right side is empty, the robot car will continue on its way forward. However, when turning right, we measure the distance again in case the right side is full, and if the distance is less than 5 cm, we write the necessary codes for the vehicle to turn to the left side and move forward. In this way, the vehicle will go towards the empty road and complete the maze.

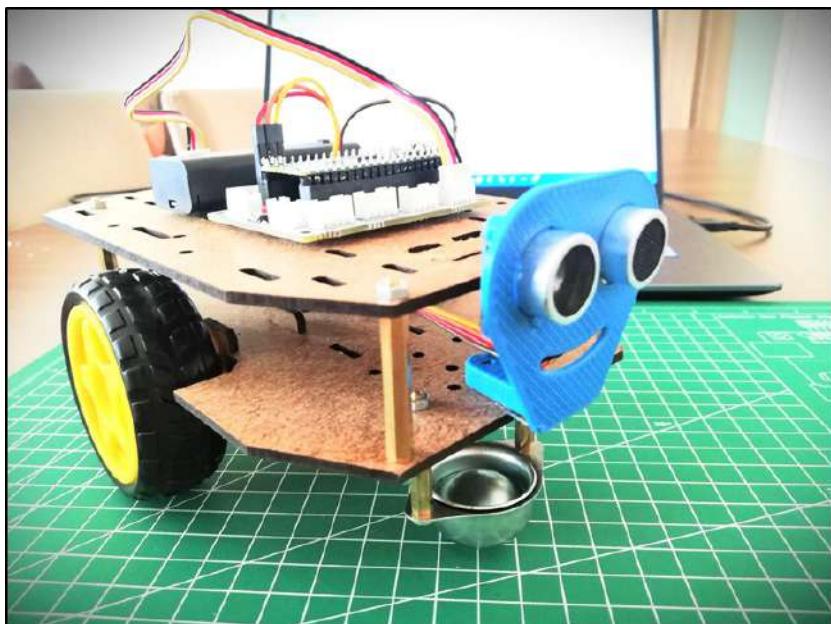


[Click](#) to access the project's MicroBlocks codes.

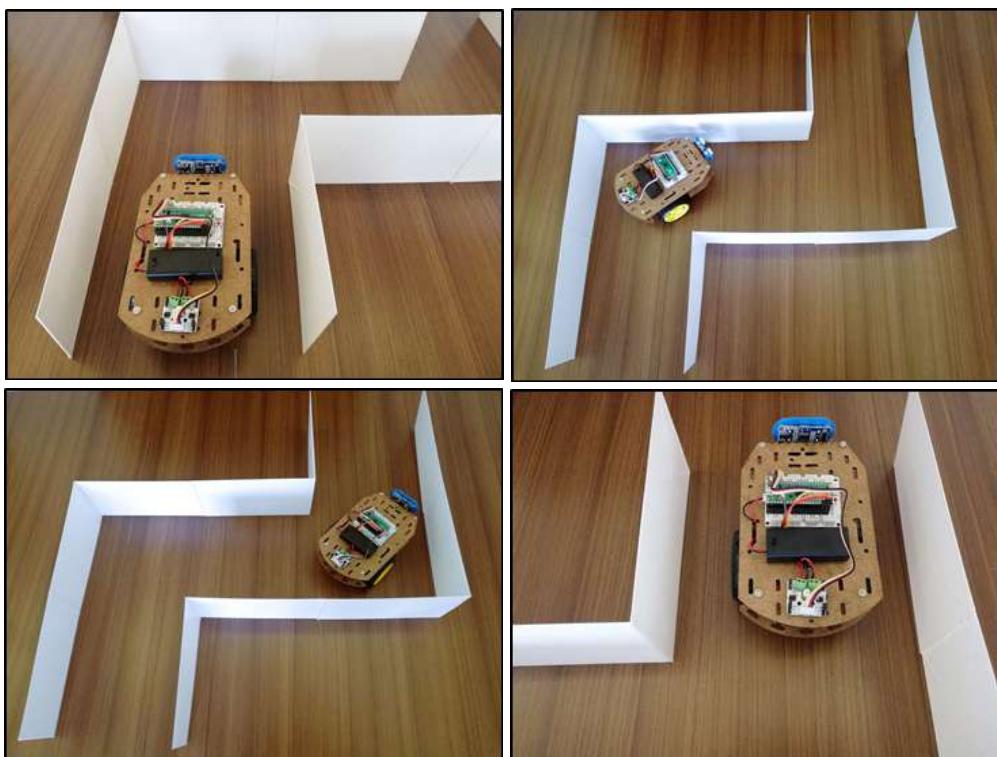
With the wait block, the operating times of the motors are determined. Since the operating times of the motors will be different depending on the state of the battery, you should optimize the waiting times according to your own robot vehicle while doing the project.

2.24.4. Construction Stages of the Project

In this project, you can build the maze-solving robot car by following the steps you did for the 2WD robot car assembly in the voice-controlled car project in the section 2.2.18. We will not use the HC05 bluetooth module in this project. In order to mount the HC-SR04 ultrasonic distance sensor on the robot, you can download the required part from the [link here](#) and print from the 3D printer and mount it on the vehicle.



After assembling the robot, you can use cardboard boxes to build the maze. You can make walls out of cardboard, or you can use 3D printed walls to connect the walls with hot glue to build the maze.



2.24.5. Project Proposal

In this project, by using the HC-SR04 ultrasonic distance sensor, we enabled the 2WD robot car to exit the maze by moving through the empty roads in the maze. By using a servo motor, you can rotate the HC-SR04 module to the right and left, and you can detect the empty roads without the vehicle turning and make it move faster. Or, you can develop the project by mounting 3 HC-SR04 ultrasonic distance sensors on the front, right and left parts of the vehicle, measuring the distances in 3 directions at the same time and directing the vehicle towards the empty road.



2.24.6. MicroPython Codes of the Project

```
from machine import Pin
from utime import sleep
import utime
#define libraries

trigger = Pin(15, Pin.OUT)
echo = Pin(14, Pin.IN)
#define sensor pins

m1 = Pin(21, Pin.OUT)
m2 = Pin(22, Pin.OUT)
#define dc motor pins

m1.low()
m2.low()
signaloff = 0
signalon = 0

def getDistance():
    trigger.low()
    utime.sleep_us(2)
    trigger.high()
    utime.sleep_us(5)
    trigger.low()
    while echo.value() == 0:
        signaloff = utime.ticks_us()
    while echo.value() == 1:
        signalon = utime.ticks_us()
    timepassed = signalon - signaloff
    distance = (timepassed * 0.0343) / 2
    return distance
#calculate distance

measure=0
while True:

    measure=int(getDistance())
    print(measure)
    if measure>5:
```



```

m1.high()
m2.high()
sleep(1) #if the distance is higher than 5, the wheels go straight
else:
    m1.low()
    m2.low()
    sleep(0.5)
    m1.high()
    m2.low()
    sleep(0.5)
    measure=int(getDistance())
if measure<5:
    m1.low()
    m2.low()
    sleep(0.5)
    m1.low()
    m2.high()
    sleep(0.5)
#If the distance is less than 5, wait, move in any direction; if the distance is less
than 5, move in the opposite direction

```

2.24.7. Arduino C Codes of the Project

```

#include <NewPing.h>

#define TRIGGER_PIN 15
#define ECHO_PIN    14
#define MAX_DISTANCE 400
//define sensor pins

NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE);

void setup() {
    pinMode(21,OUTPUT);
    pinMode(22,OUTPUT); //define dc motor pins
}

void loop() {

delay(50);
int distance=sonar.ping_cm();

```



```
Forward();

if(distance<5{

    Stop();
    delay(1000);
    Turn_Right();
    delay(1000);
    int distance=sonar.ping_cm();

    if(distance<5{
        Stop();
        delay(1000);
        Turn_Left();
        delay(500);
        // If the distance is less than 5, wait, turn right; if the distance is less than 5 again,
        move in the opposite direction
    }
}
}

void Forward(){
    digitalWrite(21,HIGH);
    digitalWrite(22,HIGH); //if the distance is higher than 5, go straight
}
void Turn_Left(){
    digitalWrite(21,LOW);
    digitalWrite(22,HIGH); //turn left
}
void Turn_Right(){
    digitalWrite(21,HIGH);
    digitalWrite(22,LOW); //turn right
}
void Stop(){
    digitalWrite(21,LOW);
    digitalWrite(22,LOW); //wait
```



}GitHub Maze Solver Robot Project Page



<http://rbt.ist/solverrobot>



2.25. Smart Greenhouse

The rapid changes in climate due to the effect of global warming cause a decrease in productivity in agricultural activities. In the 1500s, Daniel Barbaro built the first known greenhouse in history. Greenhouses are suitable environments for growing plants that can provide controllable air, water, heat and light conditions. In greenhouses, heaters are used to balance the heat, electric water motors for irrigation, fans are used to regulate humidity and to provide pollination. With the development of technology, the producer can follow the status of the greenhouse with his phone from anywhere and can do the work that needs to be done. The general name of this technology is Internet of Things (IOT).

Special sensors are used to measure temperature, humidity and oxygen content in greenhouses. In addition, special sensors measuring soil moisture are used to decide on irrigation. Electronically controlled drip irrigation systems are used to increase irrigation efficiency.

In this project, we will prepare a simple greenhouse with IOT technology and PicoBricks. We will use PicoBricks with the ESP8266 wifi module in this greenhouse. In this way, we will turn the greenhouse into an object that we can track over the Internet.

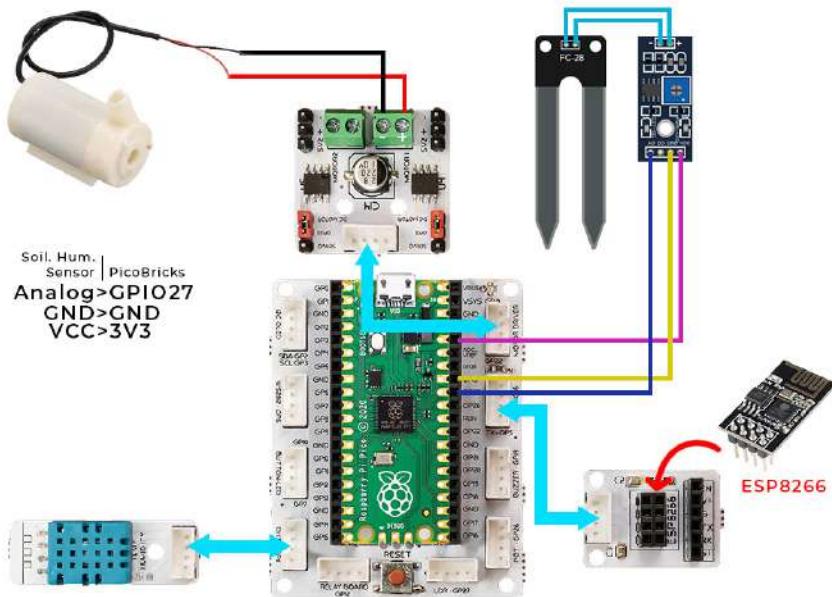
2.25.1. Project Details and Algorithm

The greenhouse model you will prepare will include a soil moisture sensor, and a DHT11 temperature and humidity sensor hanging from the top. A submersible pump will be placed in the water tank outside the model, and the hose coming out of the end of the pump will go to the ground in the greenhouse. Picoboard will be placed in a suitable place outside the greenhouse model.

When Picobricks starts, it starts to broadcast wifi thanks to the ESP8266 wifi module. When we enter the IP address of Esp8266 from the smart phone connected to the same network, we encounter the web page where we will control the Greenhouse. Here we can see the temperature and humidity values. If we wish, we can start the irrigation process by giving the irrigation command.



2.25.2. Wiring Diagram



2.25.3. Coding the Project with MicroBlocks

When Picobricks starts, the code blocks that make the WiFi settings that Greenhouse will connect to and enable the ESP8266 to connect to the wireless network and get IP are as follows. In the `ESP01_connect_to` block, write the SSID and Password information of the wireless network to which you will connect the smart greenhouse.



```

when started
  initialize local [delay] to 50
repeat (9)
  set user LED [green]
  wait [delay] millisecs
  set user LED [off]
  wait [delay] millisecs
set [_espDelay ▾] to 3000
set [_eol ▾] to
  join [string from unicode 13] [string from unicode 10] ▶
  set [_colon ▾] to [string from unicode 58]
  set [_comma ▾] to [string from unicode 44]
  set [ESP01_Log ▾] to [list]
  serial open [115200] baud
  broadcast [COMM Loop]
  say [Preparing SERVER setup.] ▶
  [Please wait till next prompt.]
  ESP01_restore
  ESP01_set_WIFI_mode [Station]
  ESP01_connect_to [yourWifiSSID] [Password] [YourWifiPassword]
  ESP01_server [Create Server]
  say
  [SERVER Ready for transactions.] [_eol] [Open a browser tab and] [_eol] [use IP:]
  [ESP01_IPAddr] [_eol] ▶▶

```

A block of code to which requests and responses are routed, where serial communication is constantly monitored.



```

when COMM Loop received
say COMM Loop started.
wait _espDelay millisecs
initializeResponses
set _serbuffer ▼ to []
forever
  set ESP01_status ▼ to []
  set ESP01_response ▼ to [ ]
  set _serbuffer ▼ to [ as byte array serial read ]
  wait _espDelay millisecs
  if length of _serbuffer > 0
    set _serbuffer ▼ to _byteArray2string _serbuffer
  comment Parse Request and Responses per _serBuffer
  if find +IPD in _serbuffer ▷ ≠ -1
    set ESP01_status ▼ to DATA
    set savebuffer ▼ to _serbuffer
    set ESP01_response ▼ to [ ]
    _handleIPD
  else if find OK in _serbuffer ▷ ≠ -1
    set ESP01_status ▼ to OK
    set savebuffer ▼ to _serbuffer
    set ESP01_response ▼ to [ ]
  else if find ERROR in _serbuffer ▷ ≠ -1
    add ERROR while processing. to list ESP01_Log
    set ESP01_status ▼ to ERROR
    set savebuffer ▼ to _serbuffer
    set ESP01_response ▼ to [ ]
  end
end

```

Code block that captures questions and directs them to the answer.



```

define _handleIPD
set _request ▾ to
copy _serbuffer from find +IPD in _serbuffer ▶▶
initialize local linkPtr to find +IPD, in _request ▶▶ + 5
set _linkID ▾ to
copy _request from linkPtr to
find _comma in _request starting at linkPtr ▶▶ - 1 ▶▶
for response in responses
if ESP01_path_of_request _request = item 1 ▾ of response
processRequest
comment
Following is a TCP messaging sequence to send back any feedback desired.
add join ESP01_path_of_request _request processed. ▶▶ to list
ESP01_Log
join
AT+CIPSEND= _linkID _comma
length of item 2 ▾ of response _eol ▶▶
serial write
wait _espDelay millisecs
serial write join item 2 ▾ of response _eol ▶▶
wait _espDelay millisecs
serial write join AT+CIPCLOSE= _linkID _eol ▶▶
wait _espDelay millisecs
▶▶

```

Code block that sends greenhouse information and irrigation completion as a response.



```

define initializeResponses
comment
APP Inventor APP processes 2 kinds of responses from this program:
1. Web Page displays of info
2. JSON data sent in response to HTTP GET /SERA transaction

#1 type responses are processed in the WebViewer component and
do not require a header info.

#2 type responses are true responses to HTTP GET and require
a proper header:
HTTP/1.1 200 OK\nContent-Type: text/html\n\n is

NOTE: timeout delay for the HTTTP GET has to be long enough
to receive and process the data. Current setting is 10000ms.
However, if more data is sent than the example, it might need
extending.

set responses ▾ to
  list /SERA
    join
      join
        HTTP/1.1 200 OK _eol Content-Type: text/html _eol _eol ["TEMP": "PicoBricks temperature (°C)"]
        "S.Moisture": read analog pin 27 / 1023 × 100 ["Humidity": PicoBricks humidity]
      list /WATERING <H1>SERA CONTROL<br/></H1>
      <p>irrigation is complete..</p>
  list /WATERING

```

Code block in which the irrigation command activates the submersible pump.

```

define processRequest
comment
Here is where all the program actions are evaluated and implemented.
Provide processing code for each function with its specific evaluation.
If the activity code is too long / large, then provide a call to a custom function.

if ESP01_path_of_request = /WATERING
  PicoBricks set motor 1 speed 100 (0-100)
  wait 10000 millisecs
  PicoBricks set motor 1 speed 0 (0-100)

```

About 20 seconds after you connect Picobricks to MicroBlocks and run the codes, Greenhouse will connect to the WiFi network and the assigned IP address will appear in the speech bubble. Make a note of this IP address to write to the Mobile app.

All activity is logged. Use ESP01_display_LOG block to see the contents of

ENJOY.

Preparing SERVER setup.
Please wait till next prompt.

```

when started
  initialize local delay to 50
  repeat (1)
    set user LED [green]
    wait delay millisecs
    set user LED [red]
    wait delay millisecs
ESP01_display_LOG

```

Now you can start testing.

Submitting the IP address in a browser session will display the menu of options
Submitting IPaddress/SERA will obtain SERA values for Temperature, Humidity, and Light Level.

All activity is logged. Use ESP01_display_LOG block to see the contents of the log.

ENJOY.

SERVER Ready for transactions.
Open a browser tab and use IP: 192.168.1.47

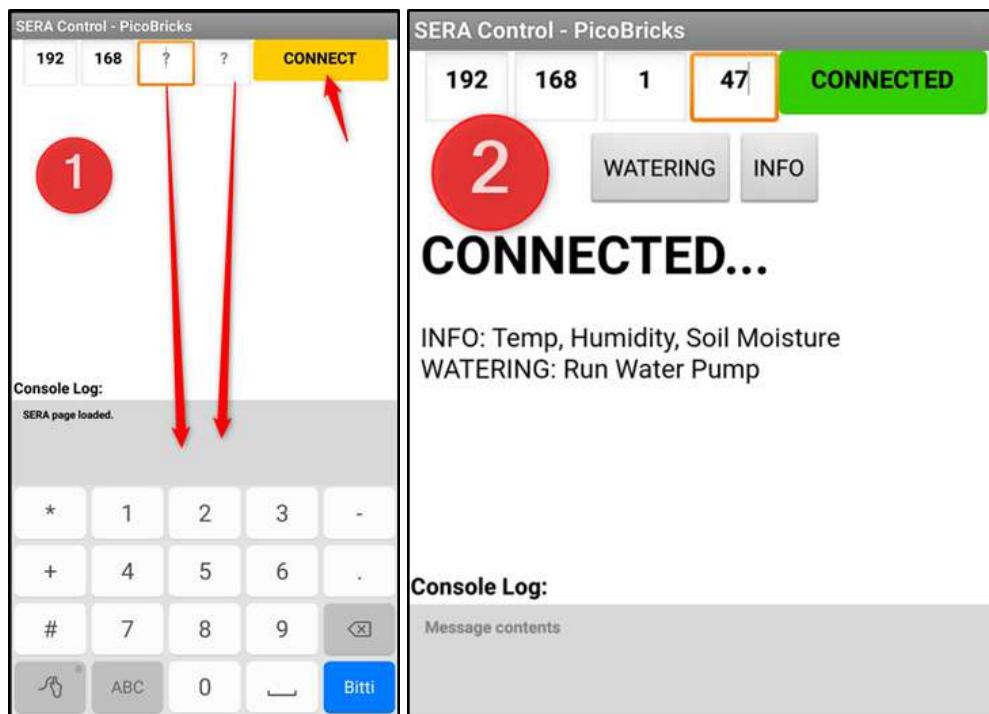
```

when started
  initialize local delay to 50
  repeat (1)
    set user LED [green]
    wait delay millisecs
    set user LED [red]
    wait delay millisecs
ESP01_display_LOG

```

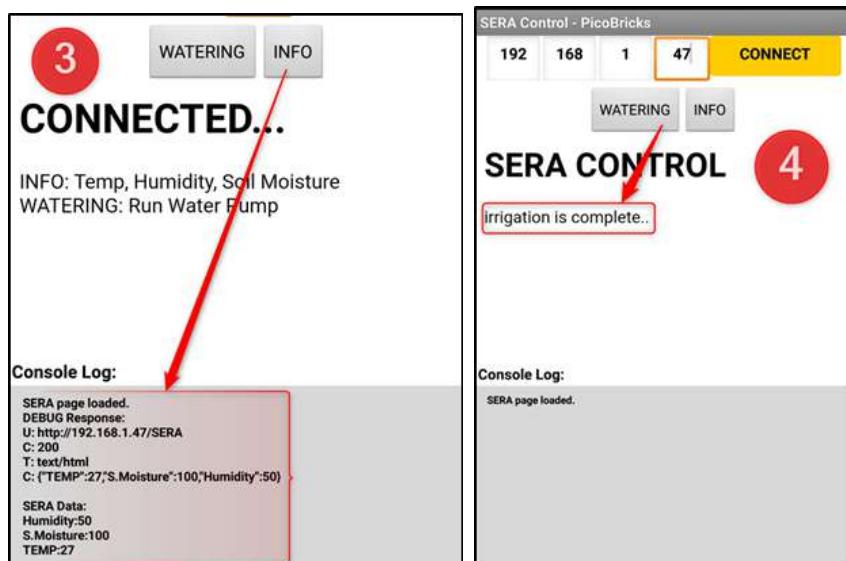


1- Install the Sera Control application on your device. The last two digits of the IP address specified by MicroBlocks codes are “?” type in the boxes and click the CONNECT button. When you encounter the screen in the image number 2, it means that Greenhouse and the mobile application are connected to each other via WiFi.



2- If you click the INFO button, after 20 seconds, the data read by the sensors in the greenhouse will be displayed on the console.

3- If you press the WATERING button, the submersible pump starts within 10 seconds and stops after 10 seconds. Then the statement that the irrigation process is finished appears on the screen.



[Click](#) to access the project's MicroBlocks codes.

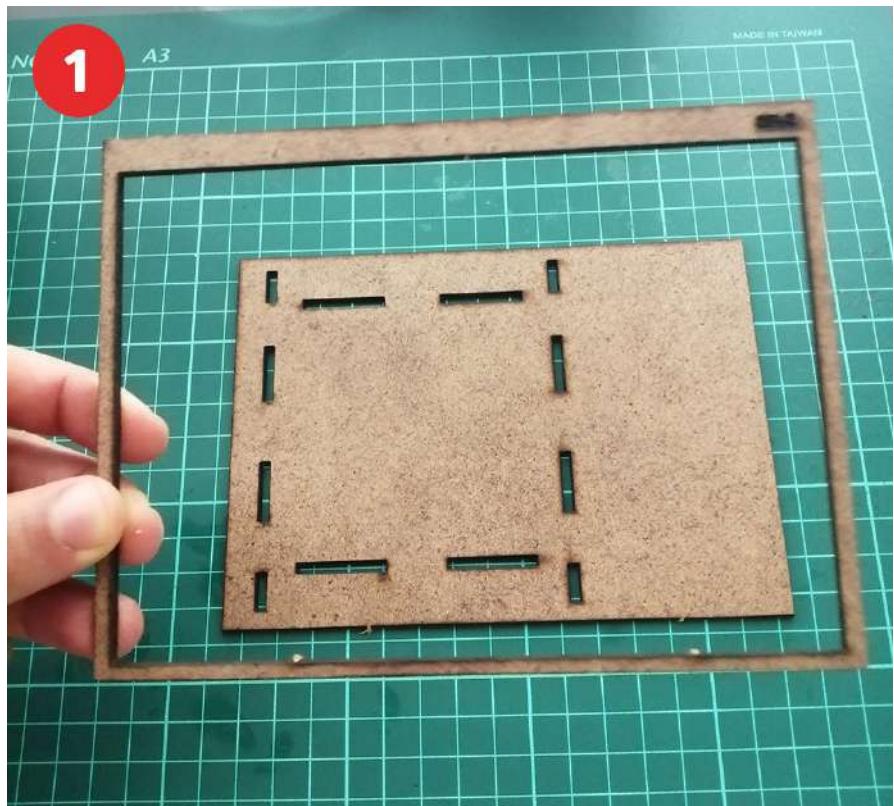
[Click](#) to download the required Android app.

[Click](#) for MIT App Inventor File.

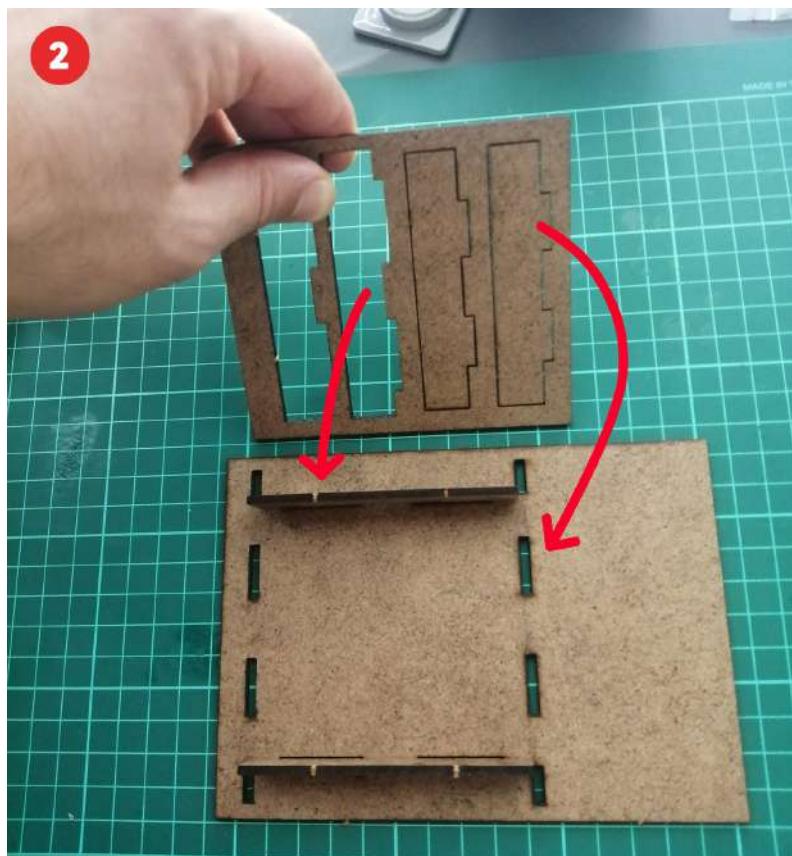


2.25.4. Construction Stages of the Project

1-Detach the floor of the Greenhouse model from the SR-2 coded part in the Greenhouse kit.



2- Attach the pieces in the middle of the SR-3 piece to the floor of the Greenhouse.





3- Remove the inner walls of the greenhouse from the SR-4 part and attach it to the ground of the greenhouse.



4- Remove the Greenhouse arches in SR-1 and SR-3 and place them on the greenhouse floor.





5-Cover the rectangular area where soil will be placed with cling film. After irrigation, you will protect the model parts. Pour the plant soil into the greenhouse. Fill so that there is no empty space.

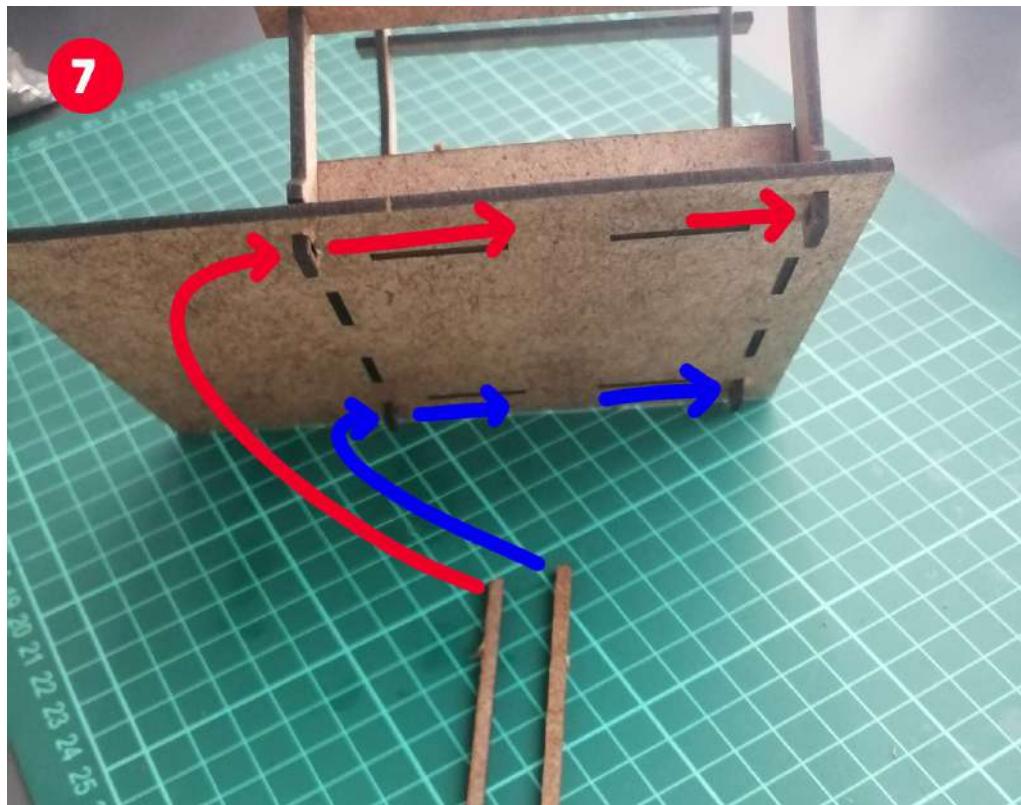


6- Insert the parts of the SR-4 into the notches on the greenhouse.





7-Thread the remaining two thin flat pieces of SR-4 through the holes on both sides of the greenhouse from the underside. This process makes the greenhouse more robust.

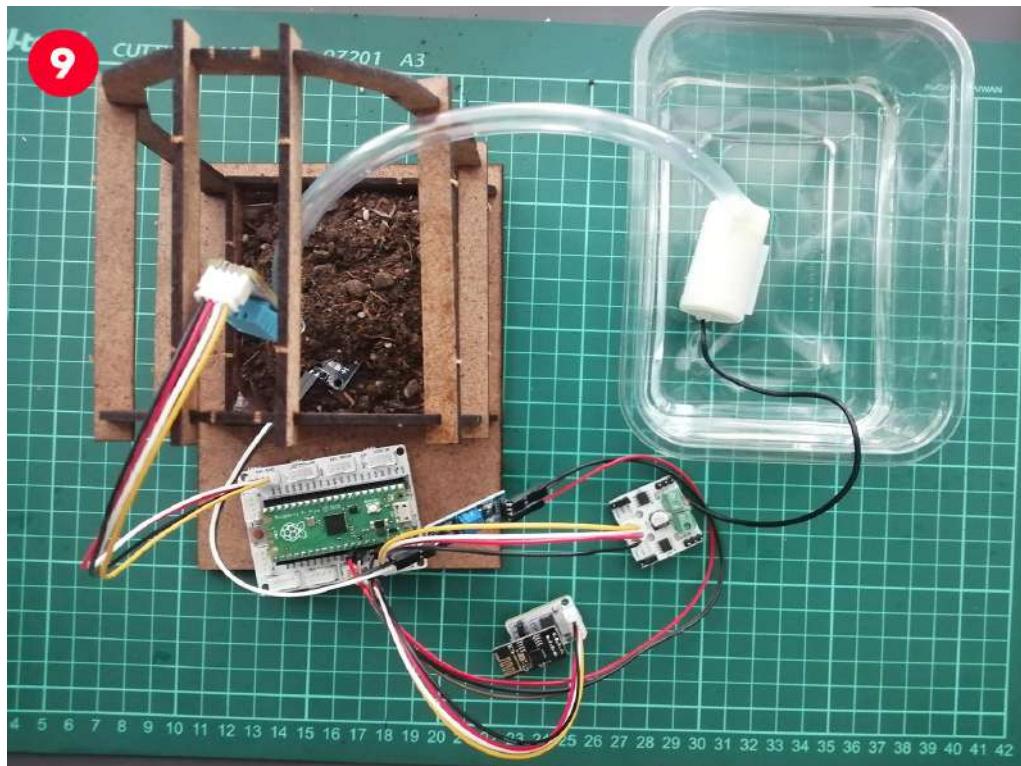


8- Pass the hose to the submersible pump. You will install an irrigation system similar to the drip irrigation system in the greenhouse. Pass the hose where you want the soil to be irrigated. Remember to cut the hose just enough to reach the water tank.





9- Place the DHT11 temperature and humidity sensor on the greenhouse model and the Soil Moisture Sensor in the soil.



10- Plug the Soil Moisture sensor to the pin number GPIO27 on the Picoboard and connect the 2 pen battery to the power input of the Picoboard. Place the submersible pump and the end of the hose in a deep container of water. Be careful not to get the motor drive wet.





2.25.5. Project Proposal

In the smart greenhouse project, by adding an OLED screen to the greenhouse entrance, you can monitor the humidity and temperature values inside, and by adding sensors such as MQ2 gas sensor, carbon dioxide sensor, air quality sensor, you can monitor the weather inside the greenhouse via WiFi with your mobile phone. In addition, by adding a DC fan and relay to the greenhouse, you can turn the ventilation on and off according to the indoor air quality with a mobile phone via WiFi.

2.25.6. MicroPython Codes of the Project

```

import utime
import uos
import machine
from machine import Pin, ADC
from picobricks import DHT11
from utime import sleep

dht_sensor = DHT11(Pin(11))
smo_sensor=ADC(27)
m1 = Pin(22, Pin.OUT)
m1.low()

print("Machine: \t" + uos.uname()[4])
print("MicroPython: \t" + uos.uname()[3])

uart0 = machine.UART(0, baudrate=115200)
print(uart0)

def Connect_WiFi(cmd, uart=uart0, timeout=5000):
    print("CMD: " + cmd)
    uart.write(cmd)
    utime.sleep(7.0)
    Wait_ESP_Rsp(uart, timeout)
    print()

def Rx_ESP_Data():
    recv=bytes()
    while uart0.any()>0:
        recv+=uart0.read(1)
    res=recv.decode('utf-8')
    return res

```



```

def Send_AT_Cmd(cmd, uart=uart0, timeout=2000):
    print("CMD: " + cmd)
    uart.write(cmd)
    Wait_ESP_Rsp(uart, timeout)
    print()

def Wait_ESP_Rsp(uart=uart0, timeout=2000):
    prvMills = utime.ticks_ms()
    resp = b""
    while (utime.ticks_ms() - prvMills) < timeout:
        if uart.any():
            resp = resp + uart.read(1)
    print("resp:")
    try:
        print(resp.decode())
    except UnicodeError:
        print(resp)

Send_AT_Cmd('AT\r\n')      #Test AT startup
Send_AT_Cmd('AT+GMR\r\n')  #Check version information
Send_AT_Cmd('AT+CIPSERVER=0\r\n')
Send_AT_Cmd('AT+RST\r\n')   #Check version information
Send_AT_Cmd('AT+RESTORE\r\n') #Restore Factory Default Settings
Send_AT_Cmd('AT+CWMODE?\r\n') #Query the WiFi mode
Send_AT_Cmd('AT+CWMODE=1\r\n') #Set the WiFi mode = Station mode
Send_AT_Cmd('AT+CWMODE?\r\n') #Query the WiFi mode again
Send_AT_Cmd('AT+CWJAP="ID","Password"\r\n', timeout=5000) #Connect to AP
utime.sleep(3.0)
Send_AT_Cmd('AT+CIFSR\r\n')  #Obtain the Local IP Address
utime.sleep(3.0)
Send_AT_Cmd('AT+CIPMUX=1\r\n')
utime.sleep(1.0)
Send_AT_Cmd('AT+CIPSERVER=1,80\r\n')  #Obtain the Local IP Address
utime.sleep(1.0)

while True:
    res = ""
    res = Rx_ESP_Data()
    utime.sleep(2.0)
    if '+IPD' in res: # if the buffer contains IPD(a connection), then respond with HTML
        handshake
        id_index = res.find('+IPD')

```



```

if '/WATERING' in res:
    print('Irrigation Start')
    m1.high()
    utime.sleep(10)
    m1.low()
    print('Irrigation Finished')
    connection_id = res[id_index+5]
    print("connectionId:" + connection_id)
    print ('! Incoming connection - sending webpage')
    uart0.write('AT+CIPSEND=' + connection_id + ',200' + '\r\n')
    utime.sleep(1.0)
    uart0.write('HTTP/1.1 200 OK' + '\r\n')
    uart0.write('Content-Type: text/html' + '\r\n')
    uart0.write('Connection: close' + '\r\n')
    uart0.write('+' + '\r\n')
    uart0.write('<!DOCTYPE HTML>' + '\r\n')
    uart0.write('<html>' + '\r\n')
    uart0.write('<body><center><H1>CONNECTED...<br/></H1></center>' + '\r\n')
    uart0.write('<body><center><H1>Irrigation Complete.<br/></H1></center>' + \
    '\r\n')
    uart0.write('</body></html>' + '\r\n')
elif '/SERA' in res:
    #sleep(1) # It was used for DHT11 to measure.
    dht_sensor.measure() # Use the sleep() command before this line.
    temp=dht_sensor.temperature
    hum=dht_sensor.humidity
    smo=round((smo_sensor.read_u16()/65535)*100)
    sendStr="\\"TEMP\\":{}, \"Humidity\\":{}, \"S.Moisture\\":{}%".format(temp,hum,smo)
    sendText="{"+sendStr+"}"
    strLen=46+len(sendText)
    connection_id = res[id_index+5]
    print("connectionId:" + connection_id)
    print ('! Incoming connection - sending webpage')
    atCmd="AT+CIPSEND=" + connection_id + "," + str(strLen)
    uart0.write(atCmd + '\r\n')
    utime.sleep(1.0)
    uart0.write('HTTP/1.1 200 OK' + '\r\n')
    uart0.write('Content-Type: text/html' + '\r\n')
    uart0.write('+' + '\r\n')
    uart0.write(sendText + '\r\n')

```



elif '/' in res:

```

print("resp:")
print(res)
connection_id = res[id_index+5]
print("connectionId:" + connection_id)
print ('! Incoming connection - sending webpage')
uart0.write('AT+CIPSEND=' + connection_id + ',200' + '\r\n')
utime.sleep(3.0)
uart0.write('HTTP/1.1 200 OK' + '\r\n')
uart0.write('Content-Type: text/html' + '\r\n')
uart0.write('Connection: close' + '\r\n')
uart0.write('+' + '\r\n')
uart0.write('<!DOCTYPE HTML>' + '\r\n')
uart0.write('<html>' + '\r\n')
uart0.write('<body><center><H1>CONNECTED.<br/></H1></center>' + '\r\n')
uart0.write('<center><h4>INFO:Get Sensor Data</br>WATERING:Run Water Pump</h4></center>' + '\r\n')
uart0.write('</body></html>' + '\r\n')
utime.sleep(4.0)
Send_AT_Cmd('AT+CIPCLOSE=' + connection_id + '\r\n') # once file sent, close connection
utime.sleep(3.0)
recv_buf = "" #reset buffer
print ('Waiting For connection...')
```

2.25.7. Arduino C Codes of the Project

```

#include <DHT.h>
#define RX 0
#define TX 1

#define LIMIT_TEMPERATURE 30
#define DHTPIN 11
#define DHTTYPE DHT11
#define smo_sensor 27
#define motor 22
#define DEBUG true
```

```

DHT dht(DHTPIN, DHTTYPE);
int connectionId;
```



```

void setup() {
  Serial1.begin(115200);
  dht.begin();
  pinMode(smo_sensor, INPUT);
  pinMode(motor, OUTPUT);

  sendData("AT+RST\r\n", 2000, DEBUG); // reset module
  sendData("AT+GMR\r\n", 1000, DEBUG); // configure as access point
  sendData("AT+CIPSERVER=0\r\n", 1000, DEBUG); // configure as access point
  sendData("AT+RST\r\n", 1000, DEBUG); // configure as access point
  sendData("AT+RESTORE\r\n", 1000, DEBUG); // configure as access point
  sendData("AT+CWMODE?\r\n", 1000, DEBUG); // configure as access point
  sendData("AT+CWMODE=1\r\n", 1000, DEBUG); // configure as access point
  sendData("AT+CWMODE?\r\n", 1000, DEBUG); // configure as access point
  sendData("AT+CWJAP=\"WIFI_ID\"\"WIFI_PASSWORD\"\r\n", 5000, DEBUG); // ADD
YOUR OWN WIFI ID AND PASSWORD
  delay(3000);
  sendData("AT+CIFSR\r\n", 1000, DEBUG); // get ip address
  delay(3000);
  sendData("AT+CIPMUX=1\r\n", 1000, DEBUG); // configure for multiple connections
  delay(1000);
  sendData("AT+CIPSERVER=1,80\r\n", 1000, DEBUG); // turn on server on port 80
  delay(1000);
}

void loop() {
  if (Serial1.find("+IPD,")) {
    delay(300);
    connectionId = Serial1.read() - 48;
    String serialIncoming = Serial1.readStringUntil('\r');
    Serial.print("SERIAL_INCOMING:");
    Serial.println(serialIncoming);

    if (serialIncoming.indexOf("/WATERING") > 0) {
      Serial.println("Irrigation Start");
      digitalWrite(motor, HIGH);
      delay(1000); // 10 sec.
      digitalWrite(motor, LOW);
      Serial.println("Irrigation Finished");
      Serial.println("! Incoming connection - sending WATERING webpage");
      String html = "";
      html += "<html>";
    }
  }
}

```



```
html += "<body><center><H1>Irrigation Complete.<br/></H1></center>";
html += "</body></html>";
espSend(html);
}

if (serialIncoming.indexOf("/SERA") > 0) {
    delay(300);

    float smo = analogRead(smo_sensor);
    float smopercent = (460-smo)*100.0/115.0 ; //min ve max değerleri değişken.
    Serial.print("SMO: %");
    Serial.println(smo);

    float temperature = dht.readTemperature();
    Serial.print("Temp: ");
    Serial.println(temperature);

    float humidity = dht.readHumidity();
    Serial.print("Hum: ");
    Serial.println(humidity);

    Serial.println("! Incoming connection - sending SERA webpage");
    String html = "";
    html += "<html>";
    html += "<body><center><H1>TEMPERATURE<br/></H1></center>";
    html += "<center><H2>";
    html += (String)temperature;
    html += " C<br/></H2></center>";

    html += "<body><center><H1>HUMIDITY<br/></H1></center>";
    html += "<center><H2>";
    html += (String)humidity;
    html += "%<br/></H2></center>";

    html += "<body><center><H1>SMO<br/></H1></center>";
    html += "<center><H2>";
    html += (String)smopercent;
    html += "%<br/></H2></center>";

    html += "</body></html>";
    espSend(html);
}
```



```

else
    Serial.println("! Incoming connection - sending MAIN webpage");
String html = "";
html += "<html>";
html += "<body><center><H1>CONNECTED.<br/></H1></center>";
html += "<center><a href='/SERIAL'><h4>INFO:Get Sensor Data</a></br><a href='/WATERING'>WATERING:Run Water Pump</a></h4></center>";
html += "</body></html>";
espSend(html);
String closeCommand = "AT+CIPCLOSE="; ///////////////close the socket connection///
esp command
closeCommand += connectionId; // append connection id
closeCommand += "\r\n";
sendData(closeCommand, 3000, DEBUG);

}

}

/////////////////sends data from ESP to webpage///////////

```

```

void espSend(String d)
{
String cipSend = " AT+CIPSEND=";
cipSend += connectionId;
cipSend += ",";
cipSend += d.length();
cipSend += "\r\n";
sendData(cipSend, 1000, DEBUG);
sendData(d, 1000, DEBUG);
}

```

///////////////gets the data from esp and displays in serial monitor///////////

```

String sendData(String command, const int timeout, boolean debug)
{
String response = "";
Serial1.print(command);
long int time = millis();
while ( (time + timeout) > millis())
{
    while (Serial1.available())
    {

```



```
char c = Serial1.read(); // read the next character.  
    response += c;  
}  
}  
  
if (debug)  
{  
    Serial.print(response); //displays the esp response messages in arduino Serial  
monitor  
}  
return response;  
}
```

GitHub Smart Greenhouse Project Page



<http://rbt.ist/greenhouse>



3. Resources

MicroBlocks Web Site: <http://microblocks.fun/>

3.1. 3D Models

3.1.1 Two Axis Robot Arm Project 3D Parts

[Download](#)

3.1.2. Piggy Bank Project 3D Parts

[Original project page](#)

Updated 3D illustrations:

[Download](#)

3.1.3. Trash Bin

[Project page](#)

3.1.4. Maze Solving Robot Project 3D Parts

[Project page](#)

3.4. Android Apps

3.4.1. Greenhouse Control Android App(.apk)

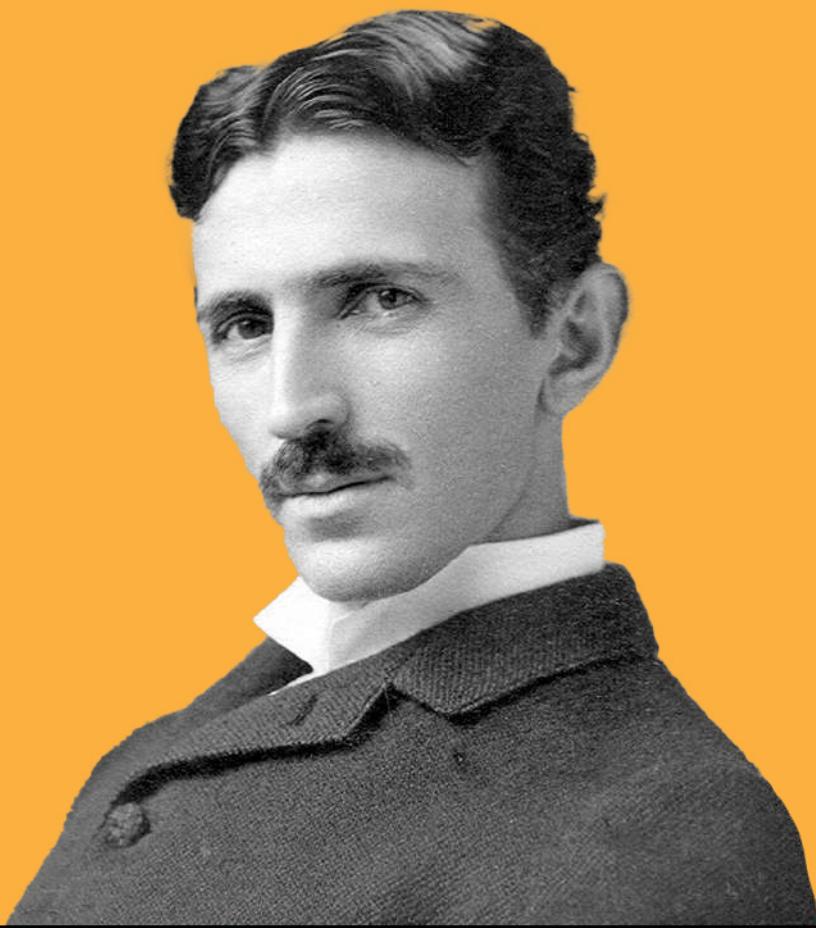
[Download](#)

3.4.2. Sera Control MITAppInventor 2 Project File

[Download](#)

3.4.3. Voice Controlled Robot Car Project Android App (.apk)

[Download](#)



"If you want to find the secrets of the universe, think in terms of energy, frequency and vibration."

Nikola Tesla



Join the community



community.robotistan.com

PicoBricks GitHub



github.com/Robotistan/PicoBricks

picobricks.com