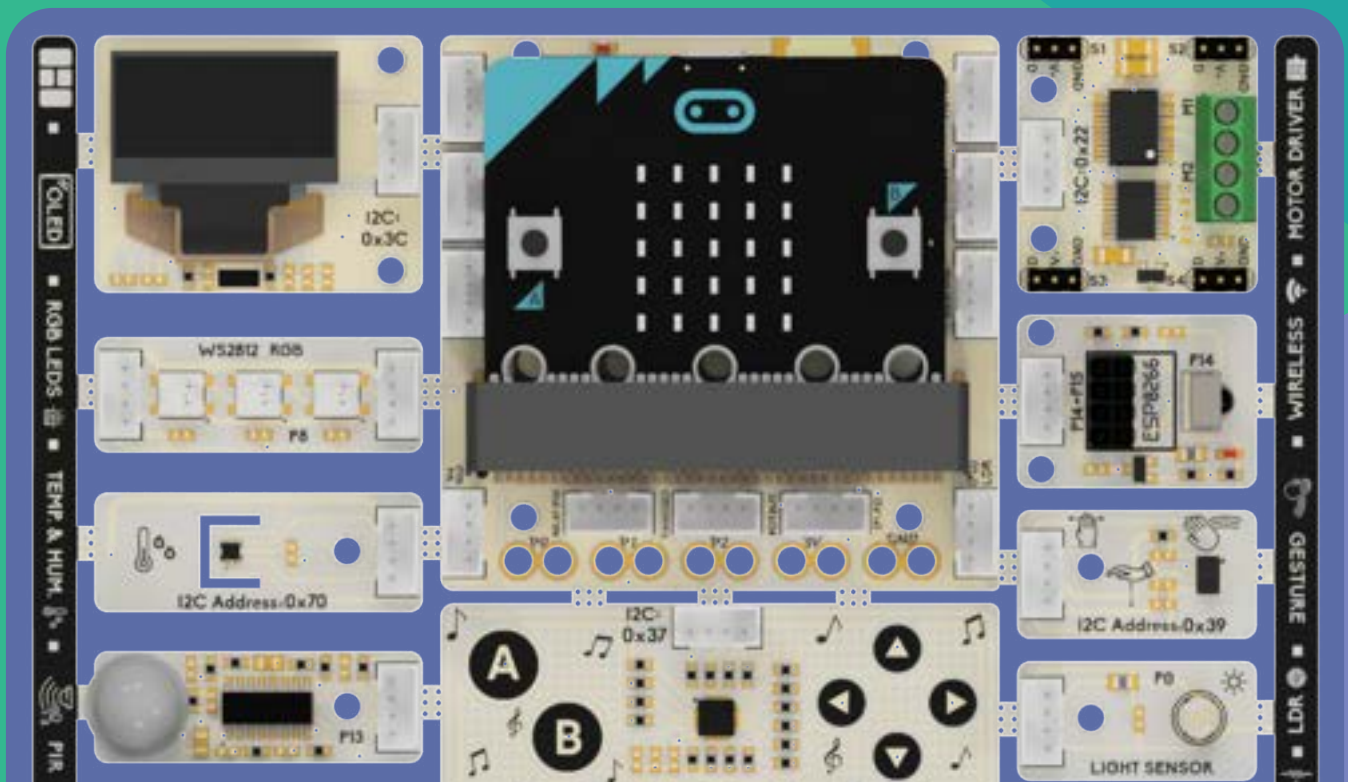
pico bricks for micro:bit | MicroPython

Python Project Book



Author: Selim Gayretli

Translate & Editor: Naze Gizem Özer

Designer: Elanur Tokalak

PicoBricks Developer Team

Project Maneger - Yasir Çiçek

Chief Developer - Suat Morkan

Product Developer - Naze Gizem Özer

Industrial Design Developer - Sercan Okay

Graphic Designer Developer - Elanur Tokalak

Hardware & Software Developer - Atakan Öztürk

Learning Content & Software Developer - Selim Gayretli

robotistan

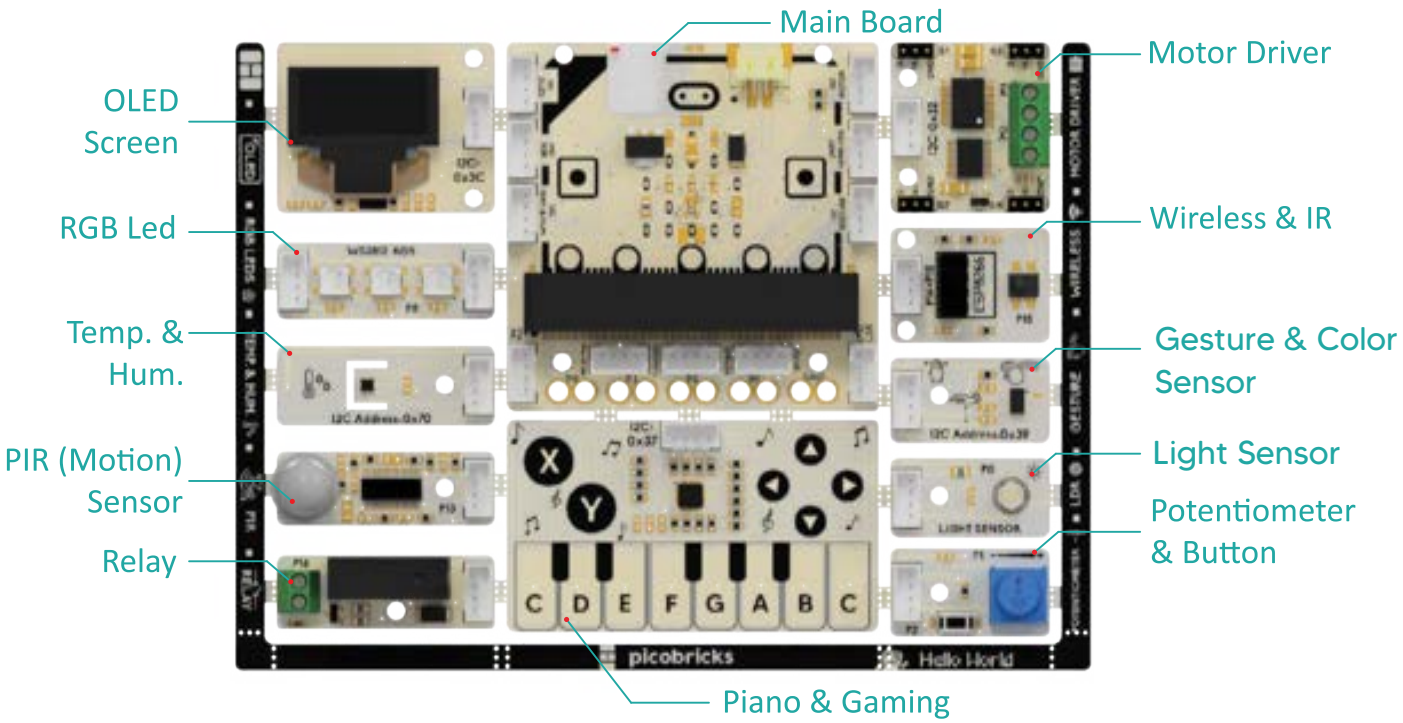
Powered by

pico bricks for micro:bit

Microsoft MakeCode

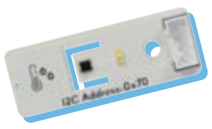
ADD ON

What Is PicoBricks?

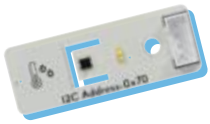


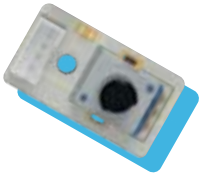
PicoBricks is a development board that eliminates the difficulties experienced in physical programming. You can invest the time saved from these challenges to create more creative projects.

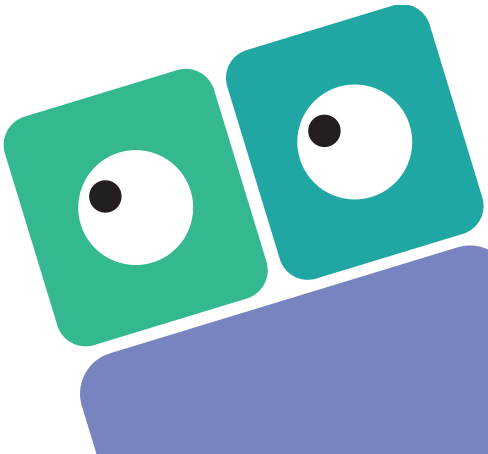
Thanks to its modular structure, PicoBricks eliminates challenges such as soldering, cable clutter, incorrect pin connections, etc., experienced in physical programming. Additionally, its microcontroller board Micro:Bit, being easily programmable and supporting various coding platforms, further eradicates programming difficulties during the coding phase.



PicoBricks supports both block-based and text-based programming tools. With MakeCode Blocks and MicroBlocks IDE, we can code our projects quickly by using block-based programming. Block-based programming tools eliminate many difficulties such as punctuation marks and functions while writing code. This makes it an effective method for developing algorithmic skills necessary for programming education, especially for young age groups or beginners. With PicoBricks, while developing projects, we can easily create complex projects by simply dragging a few code blocks onto our project page by using MakeCode and MicroBlocks programs. Additionally, PicoBricks supports the C programming language in Arduino IDE and the MicroPython programming language in Thonny IDE. Arduino IDE and Thonny IDE are the most commonly used programming tools for physical programming education among text-based programming tools. Thonny IDE eliminates punctuation (Syntax) errors frequently encountered in text-based programming languages due to its support for the MicroPython language.





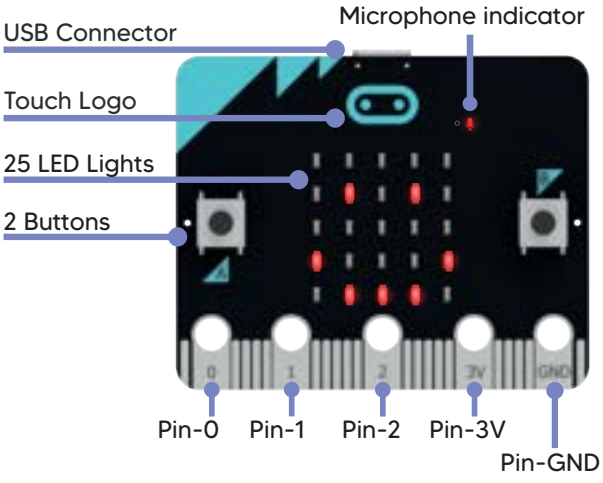


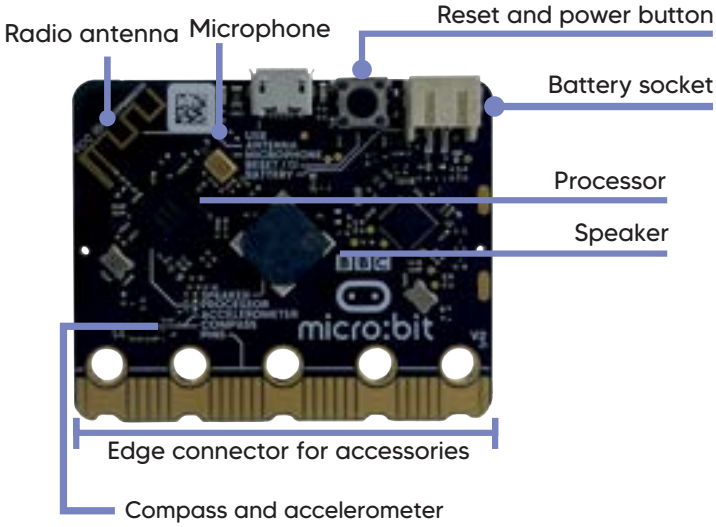
What Is Micro:Bit?

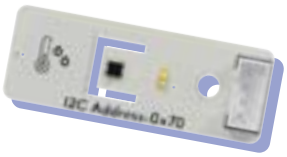
Micro:bit, a microcontroller board that features a 5x5 LED matrix, 2 buttons, an accelerometer, a compass, a speaker, and a microphone sensor on its front surface. Additionally, you can connect various sensors to the micro:bit through the pin points on the underside of the device.

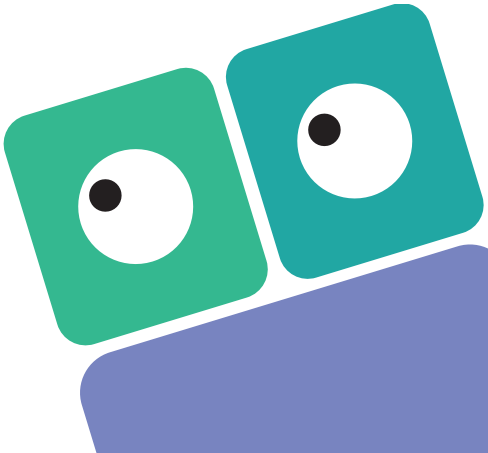
With PicoBricks, we can connect 13 different sensors to the Micro:Bit without the need for jumper cable connections.

After placing the Micro:Bit on the PicoBricks Main Board Module, we can use the following sensors without the need for jumper cable connections.









Python

Python is a great way to deepen your programming skills through text-based coding. Its natural English-like structure makes it easy to start learning, but it's also powerful enough to be used in areas like data science and machine learning.

It's widely used in schools and is supported by a global community of teachers, programmers and engineers. Our Python editor is designed to help teachers and learners get the most out of text-based programming on the micro:bit.

Why learn Python on the micro:bit?

Python is an excellent first text-based language to learn. Its instructions and syntax are based on natural language, making code easy to read, understand and modify.

As well as being widely used in education, it's used in industry, especially in the areas of data science and machine learning. Python is not just used by software developers, but also by people working in fields as diverse as medicine, physics and finance.

Python on the BBC micro:bit brings the benefits of physical computing to students aged 11-14, learning programming fundamentals through text-based coding: immersive, creative experiences for students that help build engagement and knowledge.

PicoBricks can be programmed with the Python (MicroPython) programming language both online using MakeCode Python and offline through Python editors such as Thonny IDE.

What is MakeCode Python?

MakeCode Python for Micro:Bit is a text-based (Python) software development editor developed by Microsoft. With MakeCode, you can quickly and easily create programming steps for robotic coding projects prepared using the Micro:Bit microcontroller board. Using the simulation feature, you can simulate your code blocks even if your circuit is not ready.

MakeCode is not just an editor but also a maker environment where you can publish your projects. By publishing your prepared projects on your MakeCode account, you can share them with us through https://community.robotistan.com/.

What is Thonny IDE?

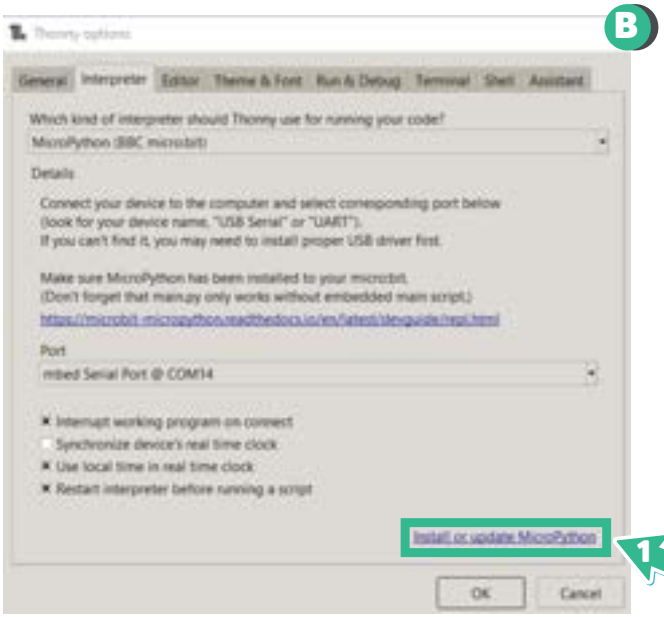
Thonny IDE, Mico:Bit, Raspberry Pi, etc. is an offline programming editor that allows us to program different microcontrollers in the Python (MicroPython) programming language. It can be coded offline with the PicoBricks Thonny IDE editor.

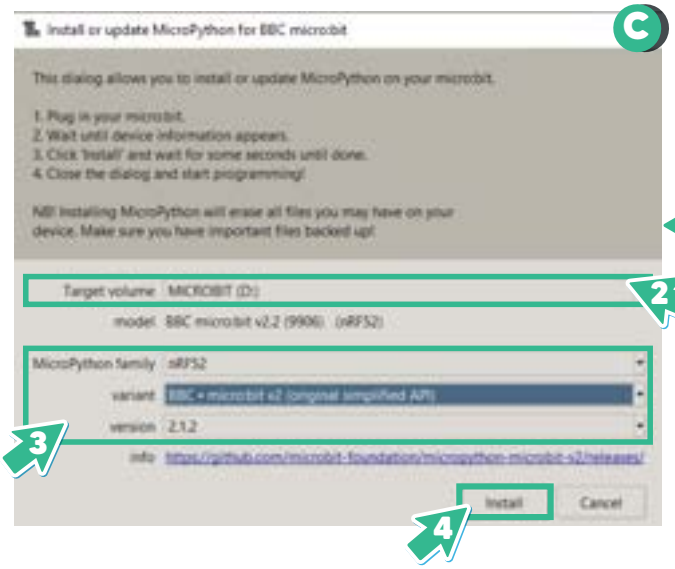
How to Use Thonny IDE

To use PicoBricks with Thonny IDE, please follow the steps below.

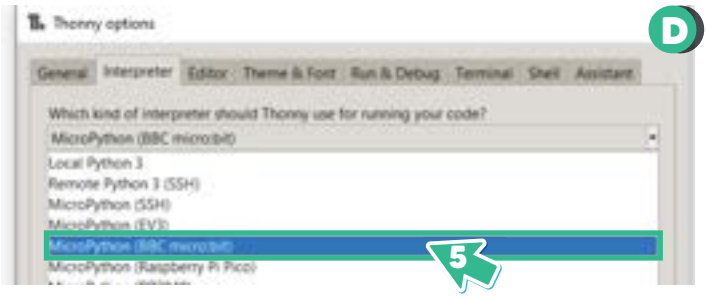
- Go to thonny.org from your browser and download the version of Thonny IDE suitable for your operating system.

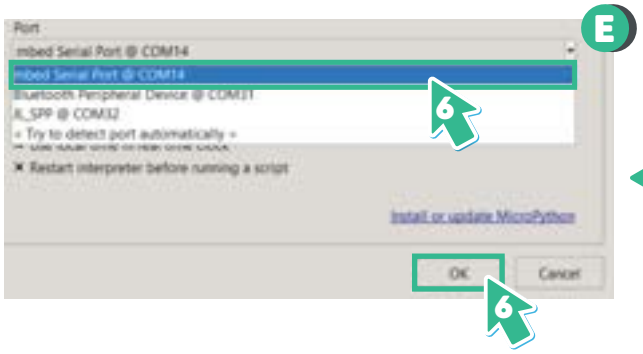
From the window that opens, follow the steps below to install the firmware required for Micro:Bit into our microcontroller.





Let's go back to the previous screen and select our microcontroller and port.





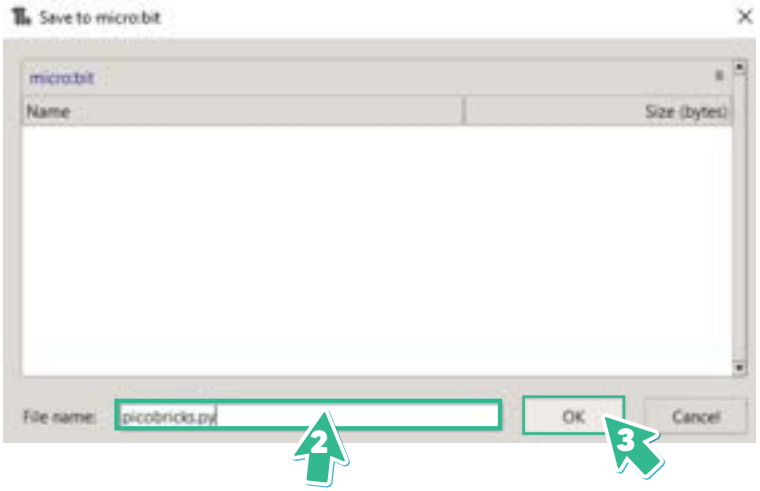
We are connected.

Now let's save the PicoBricks Python library to the microcontroller. To do this, let's go to the PicoBricks for Micro:bit GitHub page (https://github.com/Robotistan/PicoBricks-for-MicroBit) and download the picobricks.py library from the file location below to our computer.

Software>Libraries>picobricks.py

rbt.ist/pbformblib

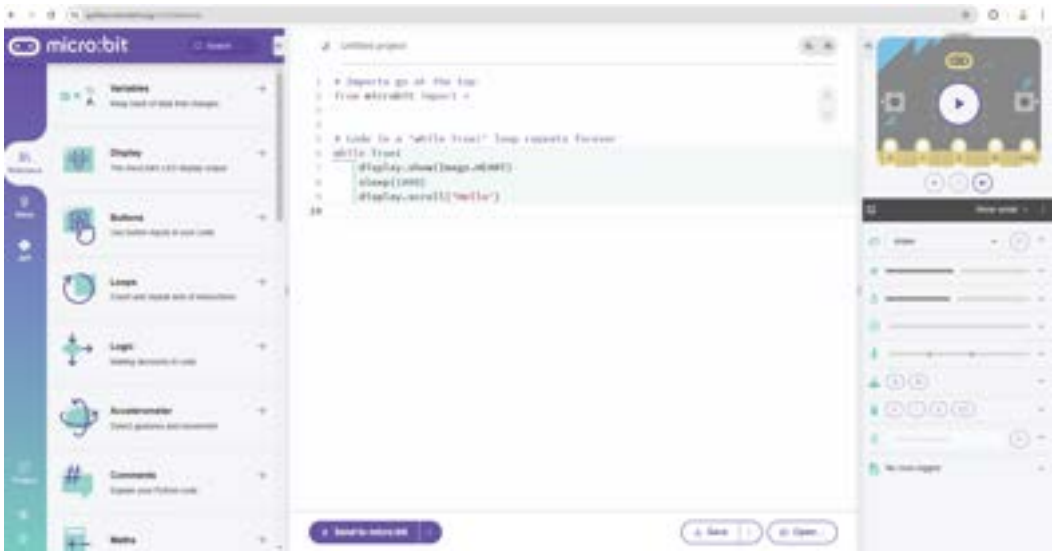
- Open the library file in Thonny IDE. Use the 'Save As' (Ctrl+Shift+S) option to save it to the Micro:bit.



- Once the saving process is successfully completed, you can start writing the project code.

How to Use MakeCode Python

- Open the MakeCode Python editor by going to (https://python.microbit.org/v/3/reference.)



Let's Get to Know MakeCode Python Editor

Reference

The Reference section makes it easy to discover what Python and the micro:bit can do, like browsing blocks in MakeCode or Scratch. Easily discovering the potential of the micro:bit hardware and writing software in Python boosts your students' creativity.

Ideas

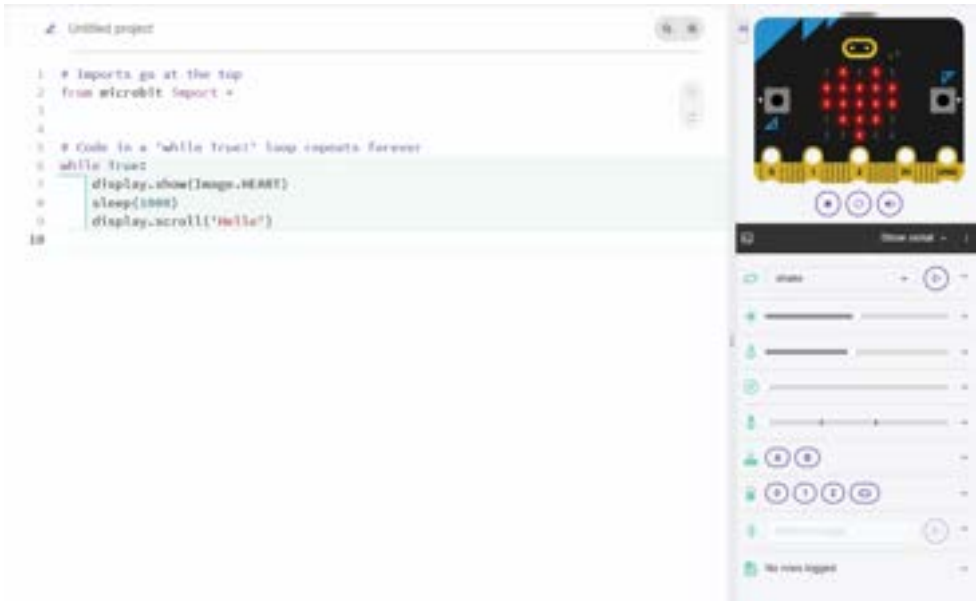
Don't forget to explore the Ideas tab which contains complete working programs your students can use straight away, then modify to make their own. You'll find an emotion badge, step counter, radio, and sound projects.

API

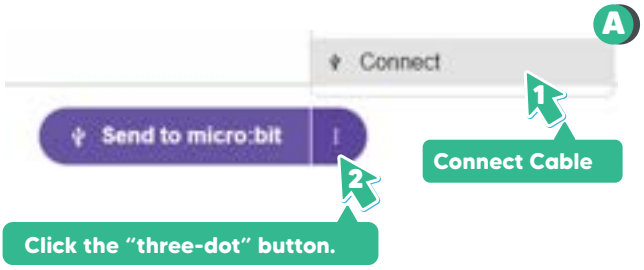
This is the area that allows you to use the functions of Micro:bit sensors, mathematics, Bluetooth, and other libraries.

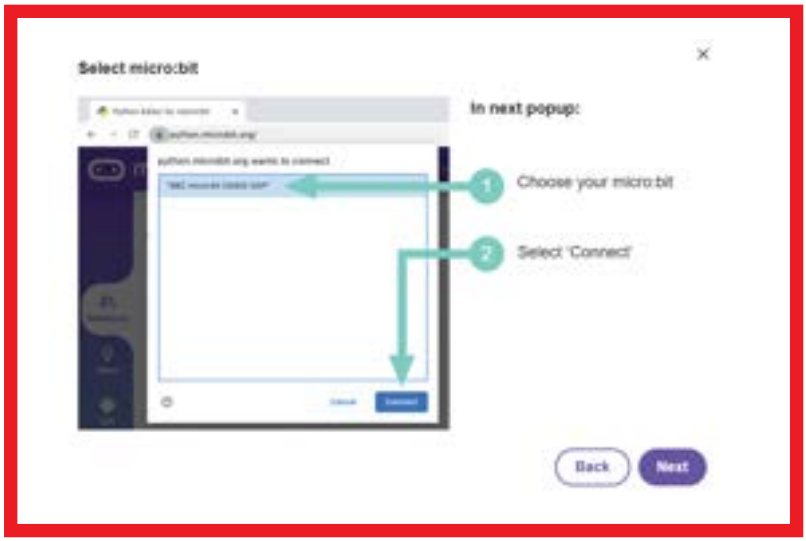
Simulator

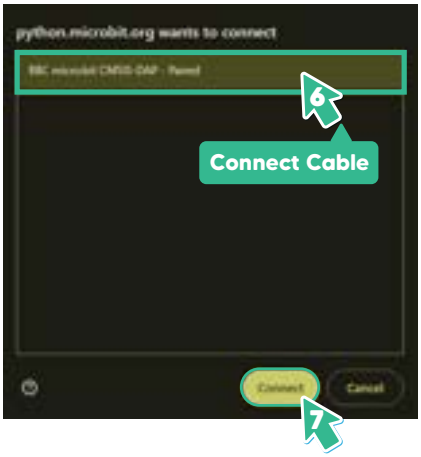
Students can test their code out using the simulator before sending it to a real micro:bit. This helps them develop, test, debug and evaluate their code and means they can work on projects even when they don't have access to a micro:bit device.



MakeCode Python Connection Steps





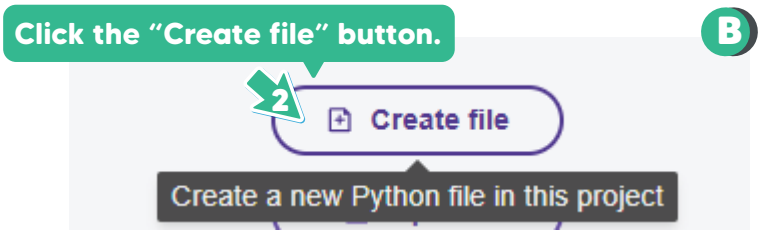


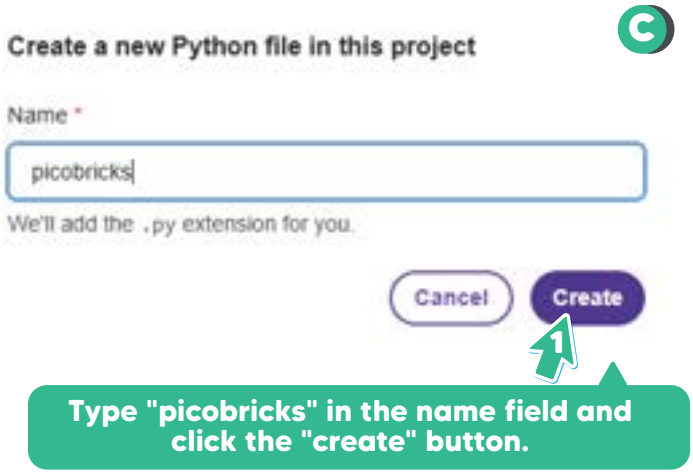
Connection Completed

Now let's save the PicoBricks Python library to the microcontroller. To do this, let's go to the PicoBricks for Micro:bit GitHub page (https://github.com/Robotistan/PicoBricks-for-MicroBit).

Software>Libraries>picobricks.py

rbt.ist/pbformblib

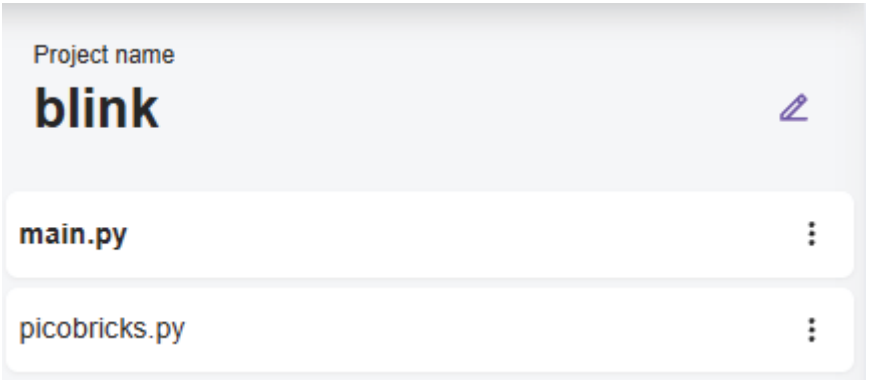


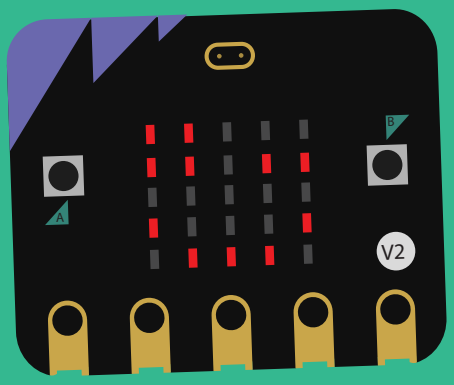


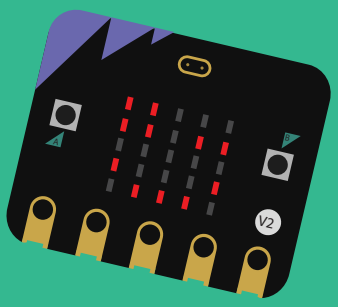
- Paste the codes

```
        if abs(gesture_ud_delta_) > abs(gesture_lr_delta_):
            return "DOWN"
        else:
            return "RIGHT"

##########HCSR04 Library##########
def measure_distance():
    # Send a 10µs pulse to the trigger to initiate measurement
    pin1.write_digital(0)
    time.sleep_us(2)
    pin1.write_digital(1)
    time.sleep_us(10)
    pin1.write_digital(0)

    # Initialize start and end times
    start_time = 0
    end_time = 0

    # Measure the duration of the echo pulse
    while pin2.read_digital() == 0:
        start_time = time.ticks_us()
    while pin2.read_digital() == 1:
        end_time = time.ticks_us()

    # Calculate the distance based on the echo time
    duration = end_time - start_time
    distance_cm = (duration / 2) * 0.0343  # Speed of sound is 343 m/s (0.0343 cm/
    return distance_cm
```

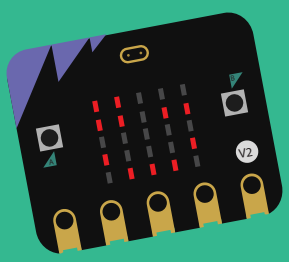
- Once the saving process is successfully completed, you can start writing the project code.

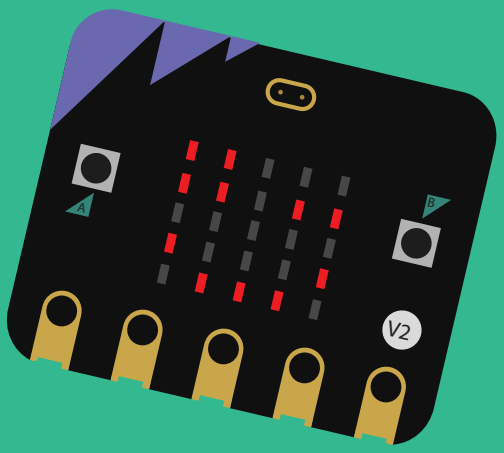






Blink





Blink Project

An employee starting a new job in real life usually takes on the most basic tasks. A janitor learns to use a broom, a chef learns kitchen utensils, and a waiter learns tray carrying. We can multiply these examples. The first code written by those who are new to software development is generally known as "Hello World." The language they use prints "Hello World" to the screen or console window when the program starts, marking the initial step in programming. It's akin to a baby starting to crawl... The first step in robotic coding, also known as physical programming, is the Blink application. In robotic coding, blinking symbolizes a significant moment. Simply by connecting an LED to the circuit board and coding it, the LED can be made to continuously blink. Ask individuals who have developed themselves in the field of robotic coding how they reached this level. The answer they will give you typically starts with: "It all began with lighting up an LED!"

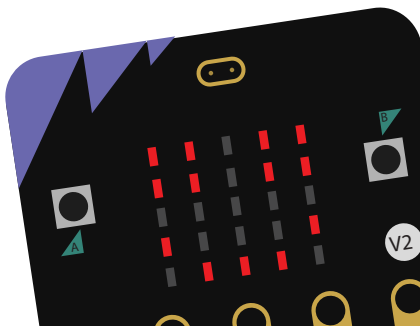
LEDs and screens are electronic circuit components that provide visual output. Thanks to output elements, a programmer can concretely determine at which stage their code is progressing. With PicoBricks, Micro:Bit includes 25 LEDs (5x5 Matrix) and a 128x64 OLED screen. When starting robotic coding with PicoBricks, printing "Hello World" on the OLED screen and winking with matrix LEDs are equally straightforward.

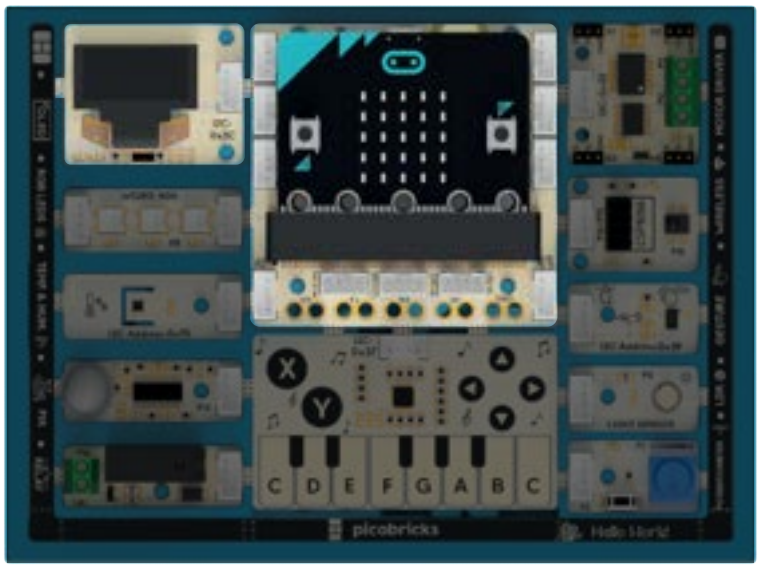
Project Details:

In this project, we will make the emoji we created using the Micro:Bit Matrix LEDs wink while displaying "Hello World" on the OLED screen.

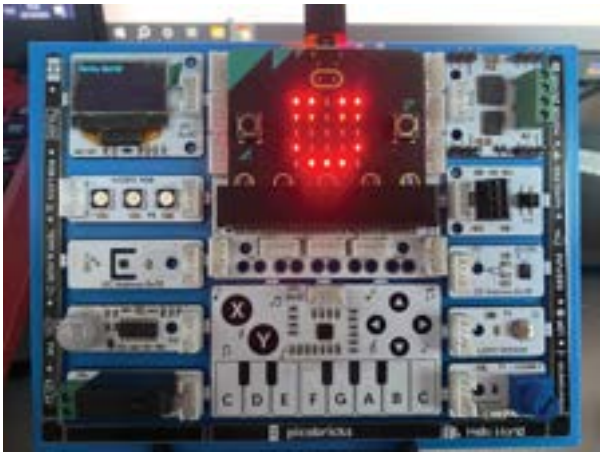
Connection Diagram:

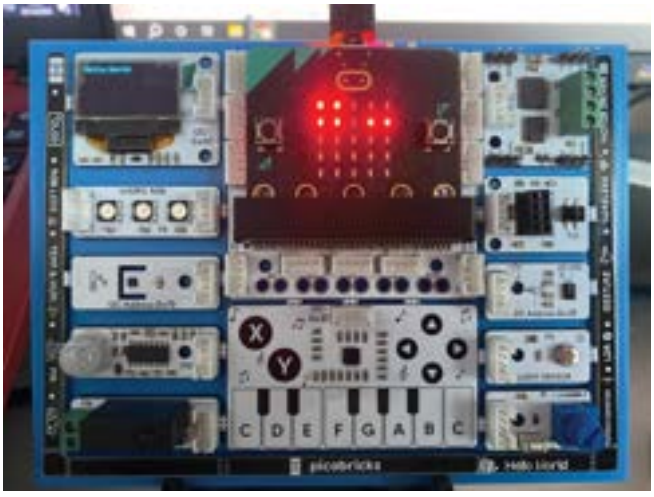
You can prepare this project without making any wiring connections.





Project Images:

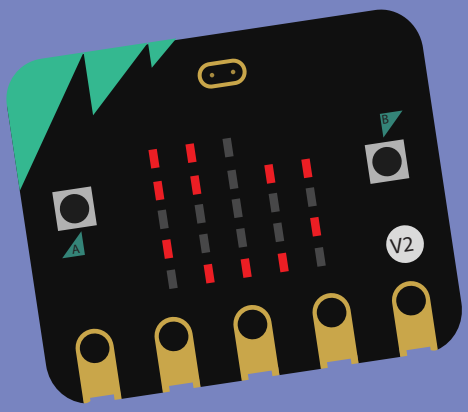




MicroBlocks Code of The Project:

blink

```
#Blink Project
from microbit import *
from picobricks import *

# Function Initialization
oled = SSD1306()
oled.init()
oled.clear()
oled.add_text(0,0,"Hello World")

#smile images
pb_smile = [
    [1, 1, 0, 1, 1],
    [1, 1, 0, 1, 1],
    [0, 0, 0, 0, 0],
    [1, 0, 0, 0, 1],
    [0, 1, 1, 1, 0],
]
#blink images
pb_blink = [
    [1, 1, 0, 0, 0],
    [1, 1, 0, 1, 1],
    [0, 0, 0, 0, 0],
    [1, 0, 0, 0, 1],
    [0, 1, 1, 1, 0],
]

while True:
    for y in range(5):
        for x in range(5):
            if pb_blink[y][x] == 1:
                display.set_pixel(x, y, 9)
            else:
                display.set_pixel(x, y, 0)
    sleep(500)
    for y in range(5):
        for x in range(5):
            if pb_smile[y][x] == 1:
                display.set_pixel(x, y, 9)
            else:
                display.set_pixel(x, y, 0)
    sleep(500)
```



Action Reaction

Action-Reaction Project

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.

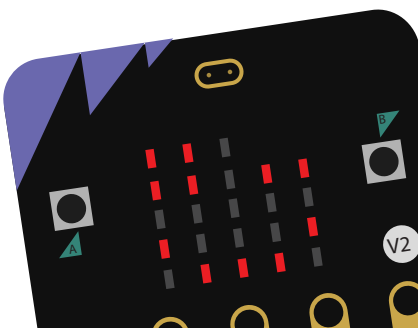
Buttons are circuit components through which we can provide input. Different types of buttons are used in electronic systems: toggle switches, push buttons and more. PicoBricks has a total of 3 push buttons, with 1 on the potentiometer and button module and 2 on the Micro:Bit. Push buttons function similarly to switches; they conduct current when pressed and do not conduct when released. PicoBricks has a total of 3 push buttons, with 1 push button on the potentiometer and button module, and 2 push buttons on the Micro:Bit. Push buttons operate like switches. Push buttons transmit current when pressed and do not transmit when released.

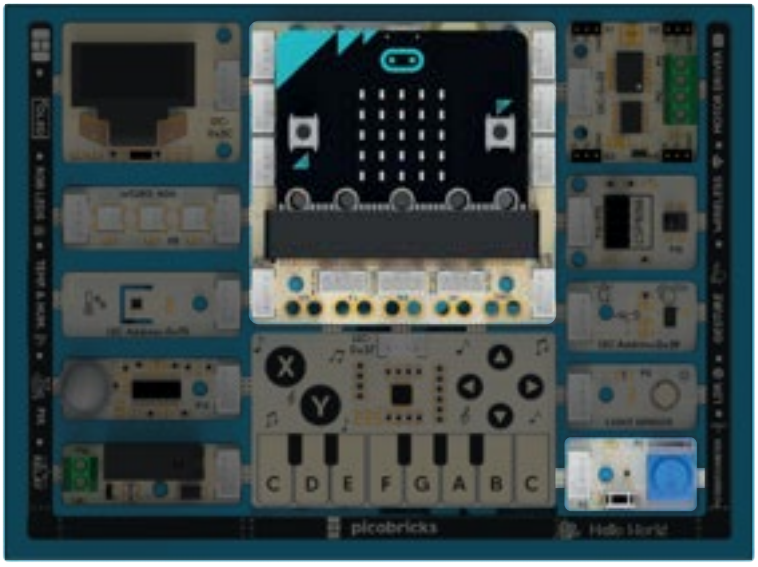
Project Details:

In the project, when the button on the potentiometer & button module is pressed, we will make the smiley face emoji we created on the Micro:Bit LED matrix blink.

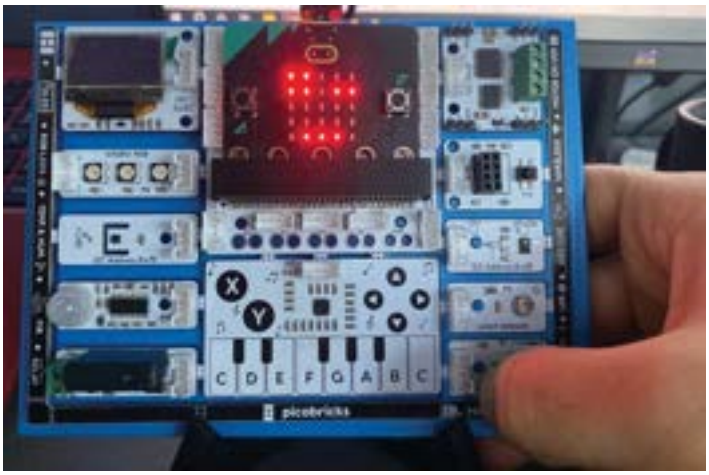
Connection Diagram:

You can prepare this project without making any cable connections.





Project Images:



MicroBlocks Code of The Project:

action-reaction

```
#Action-Reaction
from microbit import *
from picobricks import *

# Pin Initialization
Button_Pin = pin2

# Function Initialization
oled = SSD1306()
oled.init()
oled.clear()
oled.add_text(0,0,"Hello World")

#smile images
pb_smile = [
    [1, 1, 0, 1, 1],
    [1, 1, 0, 1, 1],
    [0, 0, 0, 0, 0],
    [1, 0, 0, 0, 1],
    [0, 1, 1, 1, 0],
]
#blink images
pb_blink = [
    [1, 1, 0, 0, 0],
    [1, 1, 0, 1, 1],
    [0, 0, 0, 0, 0],
    [1, 0, 0, 0, 1],
    [0, 1, 1, 1, 0],
]

while True:
    button = Button_Pin.read_digital()
    if button == 1:
        for y in range(5):
            for x in range(5):
                if pb_blink[y][x] == 1:
                    display.set_pixel(x, y, 9)
                else:
                    display.set_pixel(x, y, 0)
    else:
        for y in range(5):
            for x in range(5):
                if pb_smile[y][x] == 1:
                    display.set_pixel(x, y, 9)
                else:
                    display.set_pixel(x, y, 0)
```


COLOR
CARDS

Color Cards Project

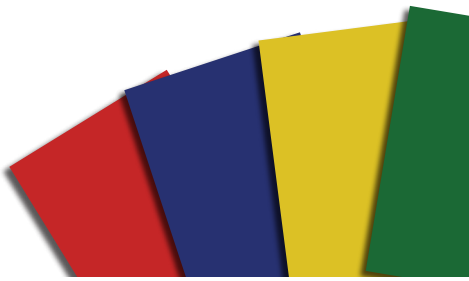
In these days, sensors that perceive the color of passing objects are commonly used in factories to alleviate workforce. For instance, different products moving on a production line can be directed to the correct conveyor belt thanks to color-sensing sensors. Many sectors employ more advanced versions of these sensors in their factories due to this feature. With the gesture module (color and motion sensor) we will use in this project, we can detect the colours of objects around PicoBricks.

The gesture module produces three numerical outputs as R (RED), G (Green), and B (BLUE) while detecting the colors of the object in front of it. When we use these outputs as the values of the RGB LED, a single colour value is formed, and this colour is the colour of the object in front of the gesture sensor.

The environment light level, distance to the object, and the object's opacity can affect the value detected by the gesture module. The recommended distance should be around 5 cm on average.

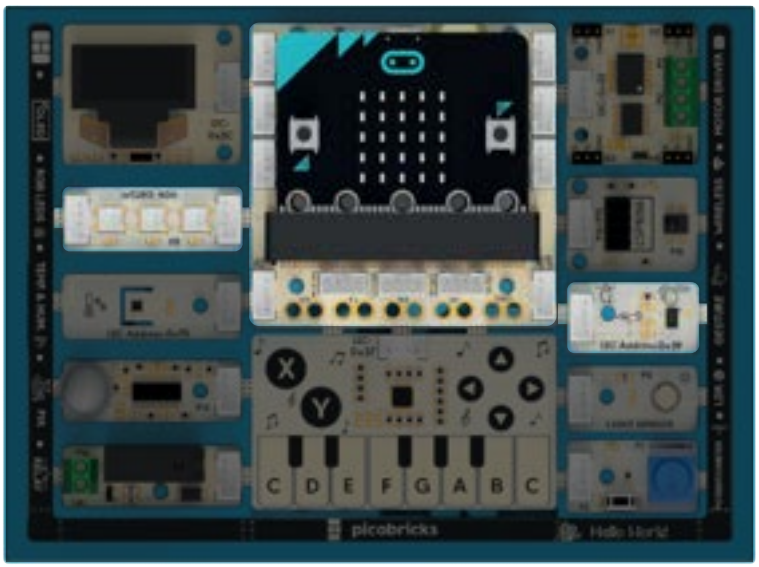
Project Details:

In this project, we will enable the gesture module to detect the colors of color cards we create by cutting colored cardboard, colored A4 paper, etc. This way, we will ensure that all 3 RGB LEDs in the RGB LED module light up in the same color. To do this, let's hold these colored papers in front of the gesture module.

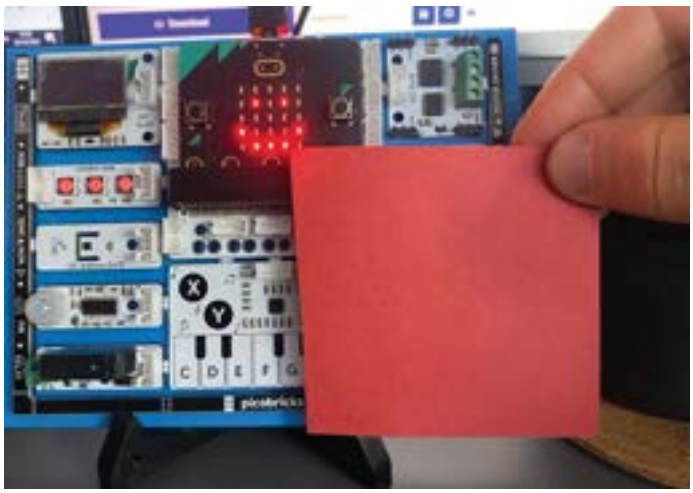


Connection Diagram:

You can prepare this project without making any cable connections.



Project Images:



MicroBlocks Code of The Project:

Color-Cards

```
#Color Cards Project
from microbit import *
from picobricks import *
import neopixel
import gc

RGB_Pin = pin8  # Pin connected to the NeoPixel strip
Num_Leds = 3    # Number of LEDs in the strip

# Initialize the APDS9960 color sensor
apds = APDS9960()
apds.init_color_sensor()
gc.collect()  # Collect garbage to free up memory

# Initialize the NeoPixel strip
np = neopixel.NeoPixel(RGB_Pin, Num_Leds)

while True:
    # Read the RGB values from the color sensor
    r_color = apds.color_value("red") or 0
    sleep(100)
    g_color = apds.color_value("green") or 0
    sleep(100)
    b_color = apds.color_value("blue") or 0
    sleep(100)

    print("red")
    print(r_color)
    print("green")
    print(g_color)
    print("blue")
    print(b_color)

    r=round(round( r_color - 0 ) * ( 255 - 0 ) / ( 1023 - 0 ) + 0)
    g=round(round( g_color - 0 ) * ( 255 - 0 ) / ( 1023 - 0 ) + 0)
    b=round(round( b_color - 0 ) * ( 255 - 0 ) / ( 1023 - 0 ) + 0)

    # Set the color of the NeoPixels
    np[0] = (r, g, b)
    np[1] = (r, g, b)
    np[2] = (r, g, b)
    np.show()  # Update the NeoPixels to show the new colors
    sleep(100)  # Wait for half a second before the next update
```

Piano

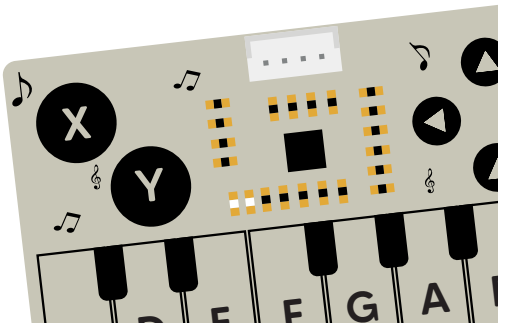
PicoBricks Piano Project

Advancements in electronics have led to the digitization of music instruments that were difficult and expensive to produce. Pianos are at the forefront of these instruments. Each key of digital pianos generates electrical signals at different frequencies, in this way, allowing them to play 88 different notes from their speakers. Factors such as the delay time of the keys on digital instruments, the quality of the speakers, and the resolution of the sound have emerged as quality-affecting elements. In electric guitars, vibrations on the strings are digitized instead of keys. In wind instruments, played notes can be converted into electrical signals and recorded through high-resolution microphones attached to the sound output. These developments in electronics have facilitated access to high-cost musical instruments and diversified music education.

In this project, we will create a touch-sensitive piano by using the PicoBricks Touch & Piano module.

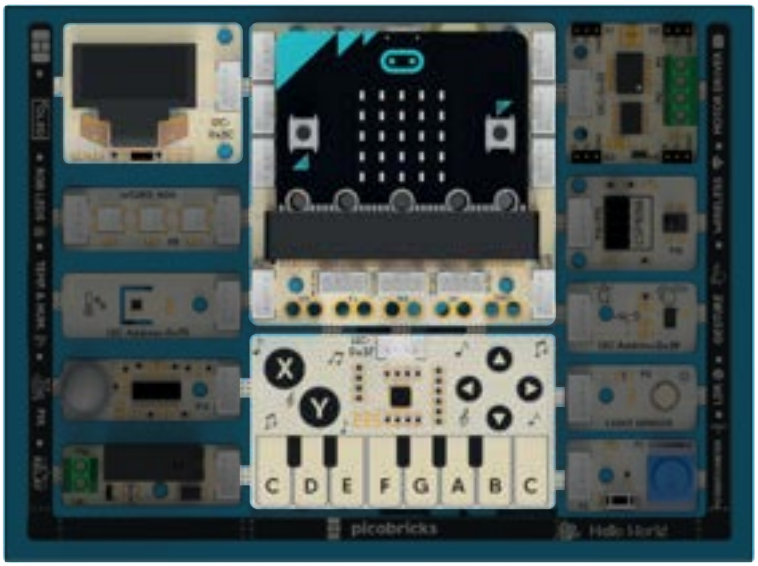
Project Details:

In this project, we will use the PicoBricks Touch & Piano module to play the desired note on the buzzer of the Micro:Bit based on the touch sensor. We will print the value of the pressed note on the Micro:Bit Matrix LEDs, and we will also display the texts "PicoBricks" and "Piano" on the PicoBricks OLED screen.

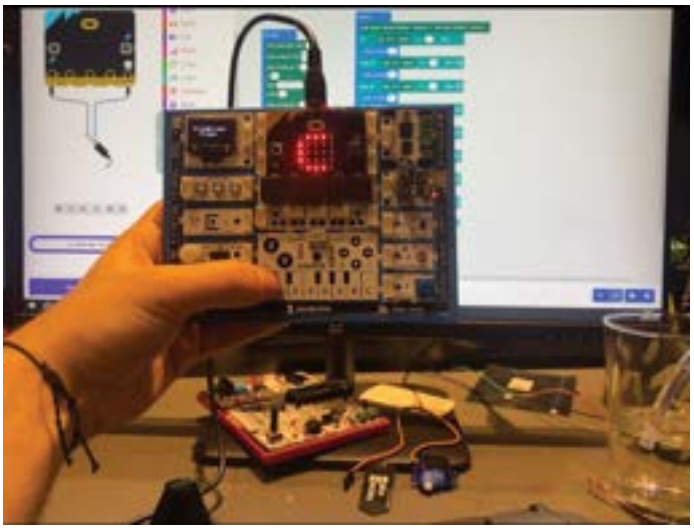


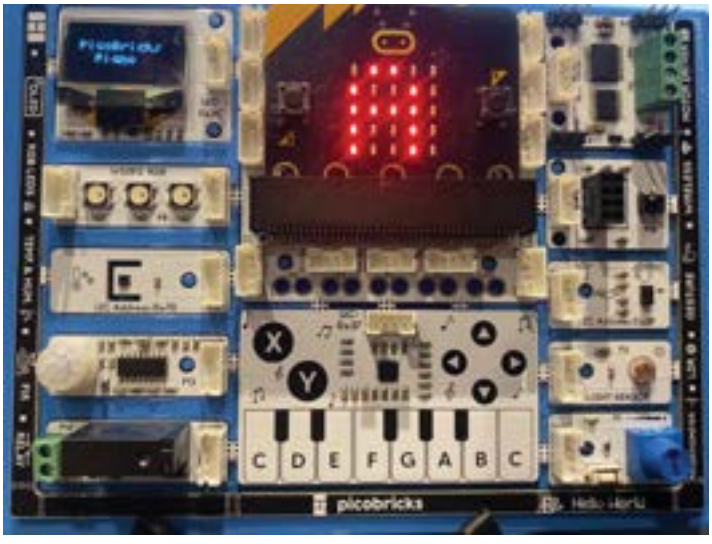
Connection Diagram:

You can prepare this project without making any cable connections.



Project Images:





MicroBlocks Code of The Project:

picoBricks-piano

```
#Piano Project
from microbit import *
from touchsensor import *

touchsensor = CY8CMBR3116()
touchsensor.init()

while True:
    touchsensor.PlayPiano()
    data = touchsensor.ReadButton()
    #print(data)
    if data == 7:
        display.show('C')
    elif data == 8:
        display.show('D')
    elif data == 9:
        display.show('E')
    elif data == 10:
        display.show('F')
    elif data == 11:
        display.show('G')
    elif data == 12:
        display.show('A')
    elif data == 13:
        display.show('B')
    elif data == 14:
        display.show('C')
```

RGB LED Control Panel

RGB LED Control Panel Project

In these days, RGB LEDs, used in various areas such as billboards, traffic lights, warning signs, etc., have a fundamental feature of being able to obtain intermediate colors by taking values between 0-255 for red, green, and blue colours. In fact, with RGB LEDs, we can create animations by changing colors on a panel we create.

There are three RGB LEDs on PicoBricks. By using the MicroBlocks editor, we can obtain various color outputs by setting the desired RGB values for each of these RGB LEDs. In this project, we will examine in detail how RGB LEDs work by changing color values with the potentiometer module and buttons.

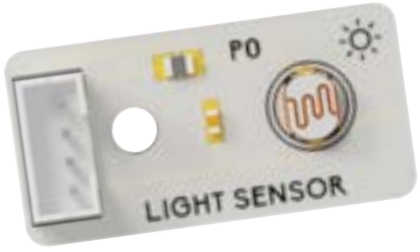
In this project, we will create a touch-sensitive piano by using the PicoBricks Touch & Piano module.

Project Details:

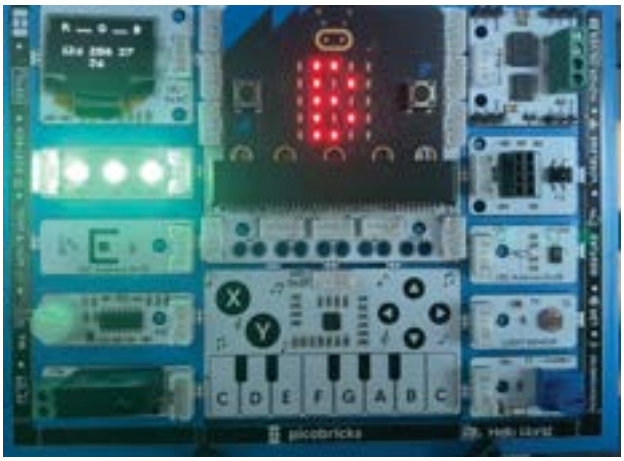
Using the PicoBricks potentiometer module, we will adjust color values between 0-255. By pressing the button on the PicoBricks Potentiometer & Button module, we will set the color value to red; by pressing Micro:Bit A button, we will set it to green, and by pressing Micro:Bit B button, we will set it to blue. This way, we will observe the instant changes in the values of the three RGB LEDs on the PicoBricks RGB LED module. At the same time, the color values will be updated on the PicoBricks OLED screen each time we press a button.

Connection Diagram:

You can prepare this project without making any cable connections.



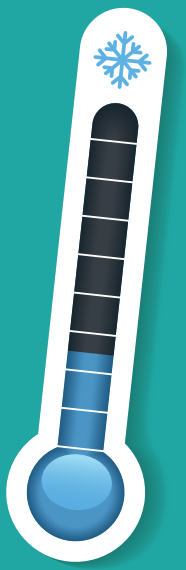
Project Images:



MicroBlocks Code of The Project:

RGB-LED-Control-Panel

```
#RGB LED Control Panel Projects
from microbit import *
from picobricks import *
import neopixel

# Pin Initialization
RGB_Pin = pin8
Num_Leds = 3
Pot_Pin = pin1
Button_Pin = pin2

# Function Initialization
oled = SSD1306()
oled.init()
oled.clear()
np = neopixel.NeoPixel(RGB_Pin, Num_Leds)

#Neopixel
np[0] = (0, 0, 0)
np[1] = (0, 0, 0)
np[2] = (0, 0, 0)
np.show()

pot_value=0
counter=0
r=0
g=0
b=0
while True:
    oldpot=pot_value
    pot_value = round(round( Pot_Pin.read_analog() - 0 ) * ( 255 - 0 ) / ( 1023 - 0 ) + 0)
    if oldpot!=pot_value:
        oled.add_text(5,3,"    ")
    oled.add_text(5,3,str(int(pot_value)))
    oled.add_text(1,0,"R __ G __ B")
    button = Button_Pin.read_digital()
    if button==1:
        display.show('R')
        o_r= r
        r = pot_value
        if o_r != r:
            oled.add_text(1,2,"    ")
        oled.add_text(1,2,str(int(pot_value)))

    if button_a.is_pressed():
        display.show('G')
        o_g=g
        g=pot_value
        if o_g==g:
            oled.add_text(5,2,"    ")
        oled.add_text(5,2,str(int(pot_value)))

    if button_b.is_pressed():
        display.show('B')
        o_b=b
        b=pot_value
        if o_b==b:
            oled.add_text(9,2,"    ")
        oled.add_text(9,2,str(int(pot_value)))

    #Neopixel
    np[0] = (r, g, b)
    np[1] = (r, g, b)
    np[2] = (r, g, b)
    np.show()
```



Thermometer

Thermometer Project

Sensors are the sensory organs of electronic systems. To perceive, we use our skin, eyes for seeing, ears for hearing, tongue for tasting, and nose for smelling. Picobricks already has many sensory organs (sensors), and new ones can also be added. By using sensors such as humidity, temperature, light, and many others, you can interact with the environment. PicoBricks can measure ambient temperature without the need for any other environmental components.

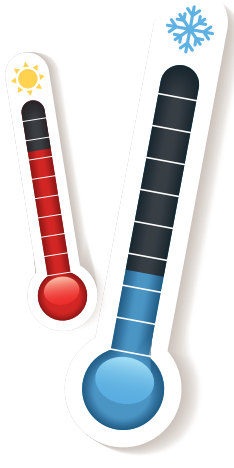
Ambient temperature is used in situations where continuous monitoring of temperature changes is required, such as in greenhouses, incubators, and environments used for transporting medications. If you are going to perform a task related to temperature changes in your projects, you should know how to measure ambient temperature. In this project, you will prepare a thermometer with PicoBricks that displays ambient temperature on the OLED screen. Using the PicoBricks potentiometer module, you can instantly change the displayed temperature value on the OLED screen between Fahrenheit and Celsius.

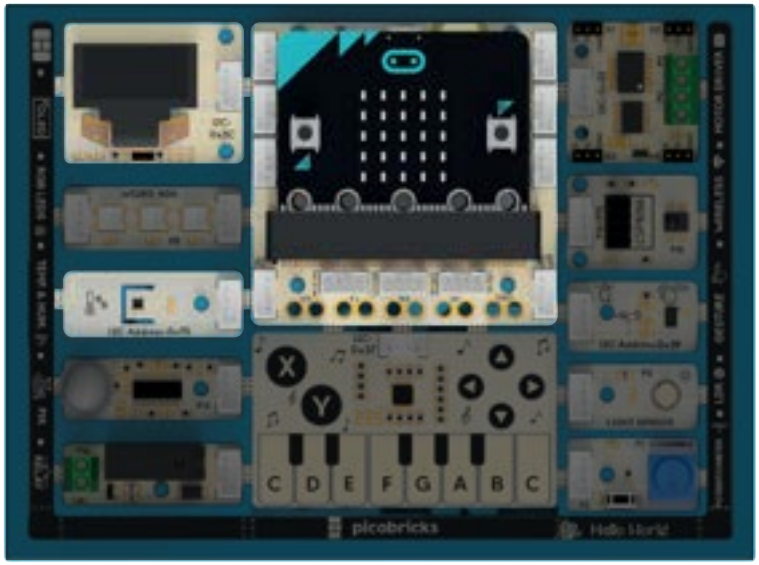
Project Details:

Thanks to the PicoBricks Temperature & Humidity module, we will display the temperature and humidity values detected from the environment on the OLED screen by using the Potentiometer module, either in Celsius or Fahrenheit.

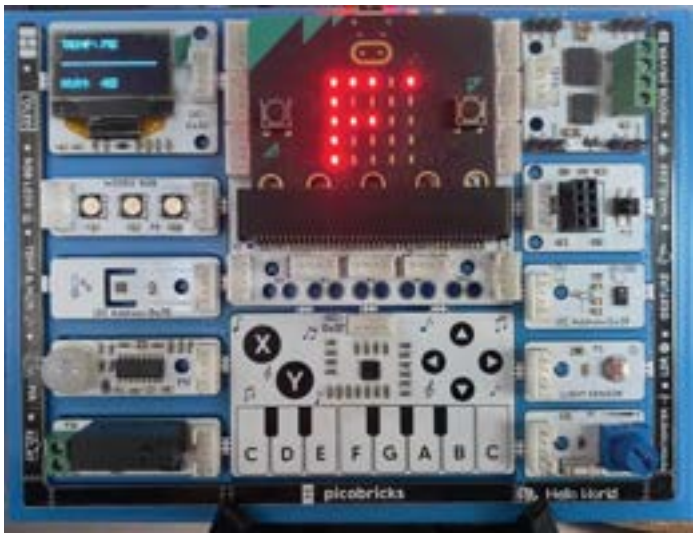
Connection Diagram:

You can prepare this project without making any cable connections.





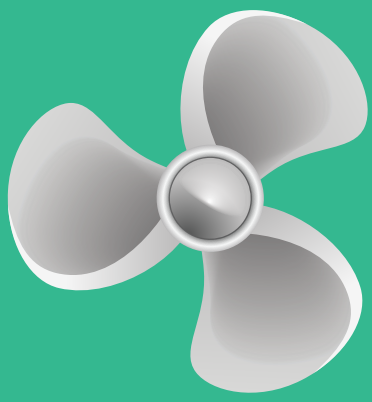
Project Images:

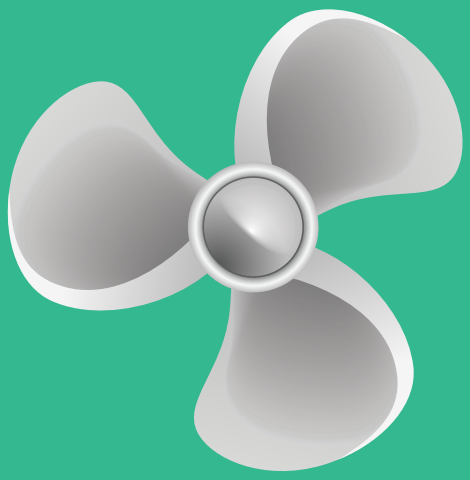


MicroBlocks Code of The Project:

Thermometer

```
#Thermometer Project
from microbit import *
from picobricks import *

# Pin Initialization
Pot_Pin = pin1

# Function Initialization
oled = SSD1306()
oled.init()
oled.clear()
shtc = SHTC3()

oled.add_text(0,0,"TEMP:")
oled.add_text(0,1,"---------------")
oled.add_text(0,3,"HUM:")

def celsius():
    display.show(Image('00009:'
                       '09900:'
                       '90000:'
                       '90000:'
                       '09900'))
def Fahrenheit():
    display.show(Image('99909:'
                       '90000:'
                       '99900:'
                       '90000:'
                       '90000'))

while True:

    temp = shtc.temperature()
    hum=shtc.humidity()
    pot_value = round(round( Pot_Pin.read_analog() - 0 ) * ( 2 - 1 ) / ( 1023 - 0 ) + 1)
    if pot_value==1:
        celsius()
        temp=round(shtc.temperature())
    else:
        Fahrenheit()
        temp=round((9*shtc.temperature())/5 + 32)
    oled.add_text(5,0,str(temp))
    oled.add_text(5,3,str(round(hum)))
```



Smart Cooler

Smart Cooler Project

To cool off during the summer months and warm up in the winter months, air conditioners are used. Air conditioners adjust the heating and cooling degrees based on the temperature of the environment. Ovens, on the other hand, strive to reach and maintain the temperature value set by the user while cooking. Both of these electronic devices use special temperature sensors to control the temperature. Additionally, in greenhouses, temperature and humidity are measured together. To maintain a balance at the desired level, a fan is used to provide air circulation.

You can measure temperature and humidity separately with PicoBricks and interact with the environment using these measurements. In this project, we will prepare a cooling system with PicoBricks that automatically adjusts fan speed based on temperature. This way, you will learn about the operation of a DC motor system and how to adjust motor speed.

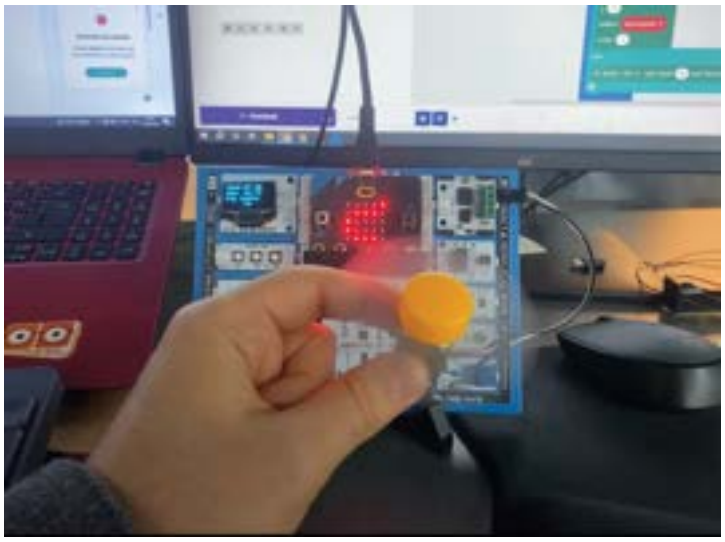
Project Details:

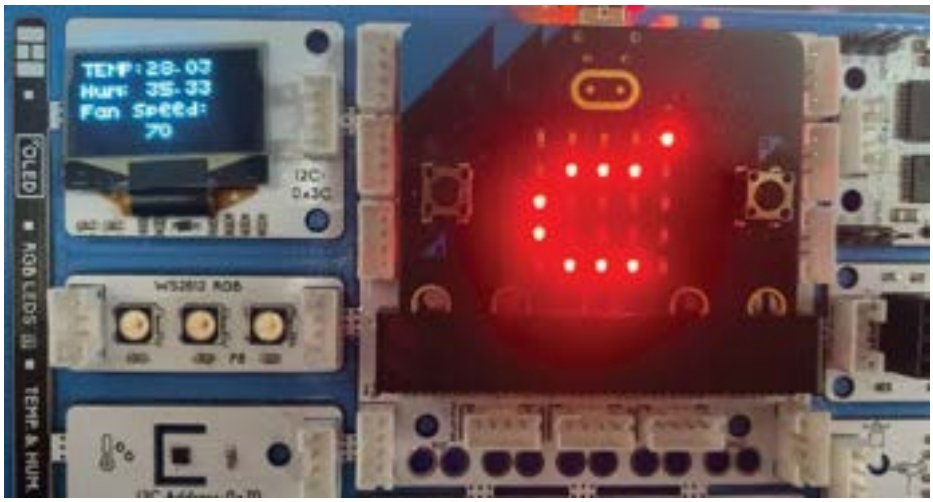
In this project, we will adjust the speed of the fan connected to the motor driver based on the value obtained from the temperature and humidity module. The fan connected to the motor driver will operate when the temperature exceeds a certain value. If the temperature falls below a certain value, the fan will stop.

Connection Diagram:

You can prepare this project without making any cable connections.

Project Images:





MicroBlocks Code of The Project:

Smart-Cooler

```
#Smart Cooler Project
from microbit import *
from picobricks import *

# Function Initialization
oled = SSD1306()
oled.init()
oled.clear()
shtc = SHTC3()
motor = motordriver()

def celsius():
    display.show(Image('00009:'
                       '09900:'
                       '90000:'
                       '90000:'
                       '09900'))
celsius()

while True:
    temp = shtc.temperature()
    hum=shtc.humidity()
    oled.add_text(0,0,"TEMP:")
    oled.add_text(5,0,str(float(temp)))
    oled.add_text(0,1,"HUM:")
    oled.add_text(5,1,str(float(hum)))

    motorSpeed=round( shtc.temperature() - 0 ) * ( 100 - 0 ) / ( 40 - 0 ) + 0
    if temp>25:
        motor.dc(1,round(motorSpeed),1)
        oled.add_text(0,2,"Fan Speed:")
        oled.add_text(5,3,str(round(motorSpeed)))
    else:
        motor.dc(1,0,1)

```

Night and Day

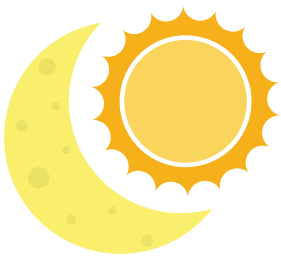
Night and Day Project

How about playing the Day and Night game electronically, a game often played in schools? In this game, when the teacher says "night," we bend our heads, and when the teacher says "day," we raise our heads. It's a game that involves using your attention and reflexes. In this project, we will use a 0.96" 128x64 pixel I2C OLED screen. Since OLED screens can be used as an artificial light source, you can reflect the characters on the screen onto any desired plane using a lens or a mirror. Systems that can project information, road, and traffic data onto smart glasses and car windows can be created using OLED screens.

Light sensors are devices that can measure the light levels in the environment, also known as photodiodes. The electrical conductivity of the sensor changes when exposed to light. By coding, we will control the light sensor and develop electronic systems affected by the amount of light.

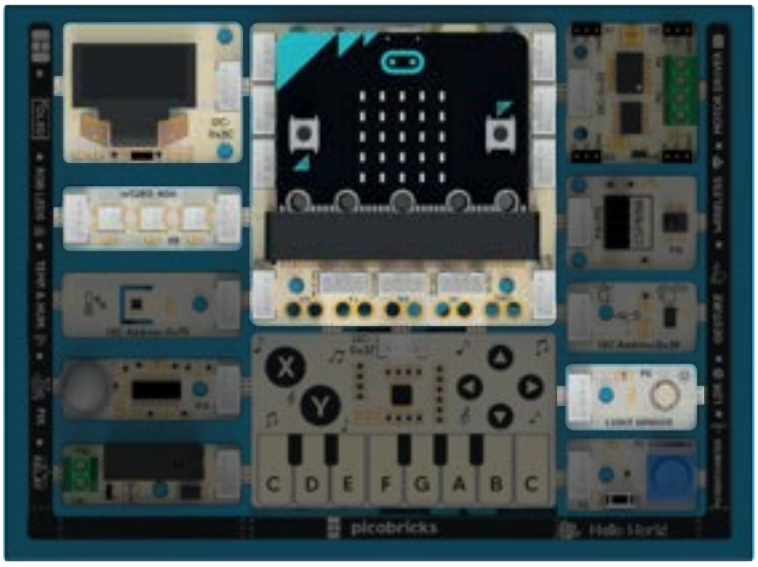
Project Details:

First, we will provide the player to press the button to start the game. Then, on PicoBricks' OLED screen, we will randomly display the expressions NIGHT and DAY for 2 seconds each. If the word NIGHT is displayed on the OLED screen, the player must cover the LDR sensor with their hand within 2 seconds. If the word DAY is displayed on the OLED screen, the player must remove their hand from the LDR sensor within 2 seconds. Each correct response from the player will earn them 10 points and create a checkmark (✓) icon on the Micro:Bit Matrix LEDs. When the player gives an incorrect response, the game will end, and the screen will display a written message indicating the end of the game along with a cross (X) icon on the Matrix LEDs. The buzzer will play a sound in a different tone, and the OLED screen will show the score information. If the player achieves a total of 10 correct responses and earns 100 points, the message "Congrats!!!" will be displayed on the OLED screen at the designated positions.



Connection Diagram:

You can prepare this project without making any cable connections.



Project Images:

MicroBlocks Code of The Project:

Night-and-Day

```
#Night and Day Projects
from microbit import *
from picobricks import *
import neopixel
import random
import music

# Pin Initialization
LDR_Pin = pin0
Button_Pin = pin2
RGB_Pin = pin8
Num_Leds = 3

# Function Initialization
oled = SSD1306()
oled.init()
oled.clear()
np = neopixel.NeoPixel(RGB_Pin, Num_Leds)

counter=0
falseValue=0

button = Button_Pin.read_digital()

while Button_Pin.read_digital()==0:
    oled.add_text(0,1,"Press BUTTON")
    oled.add_text(2,2,"to START!")

oled.clear()
while counter!=100 and falseValue==0:
    light = LDR_Pin.read_analog()
    rand=random.randint(1, 2)
    if rand==1:
        oled.add_text(0,0,"NIGHT")
    else:
        oled.add_text(0,0,"DAY")
    sleep(3000)
    light = LDR_Pin.read_analog()
    if light<60 and rand==1:
        display.show(Image.YES)
        counter=counter+10
    elif light>60 and rand==1:
        display.show(Image.NO)
        falseValue=1
    elif light>60 and rand==2:
        display.show(Image.YES)
        counter=counter+10
    else:
        display.show(Image.NO)
        falseValue=1
    oled.clear()
```


1
2
Fizz
4
Buzz
7
Fizz - Buzz Game
Fizz
11
8
Fizz
Buzz

Fizz - Buzz Game Project

There are some games that every programmer spends time on. Fizz Buzz is one of them. Every programmer who has made some progress in a programming language has created the algorithm for the Fizz-Buzz game, aiming to master that language by writing this game. The Fizz-Buzz game is frequently preferred in programming language education because its algorithm includes both conditional statements and loop structures, helping to grasp the steps of computational thinking. At the same time, while playing this game, we improve our quick decision-making and mathematical thinking skills.

Thanks to PicoBricks, we can code this game by using electronic components and experience it physically.

Project Details:

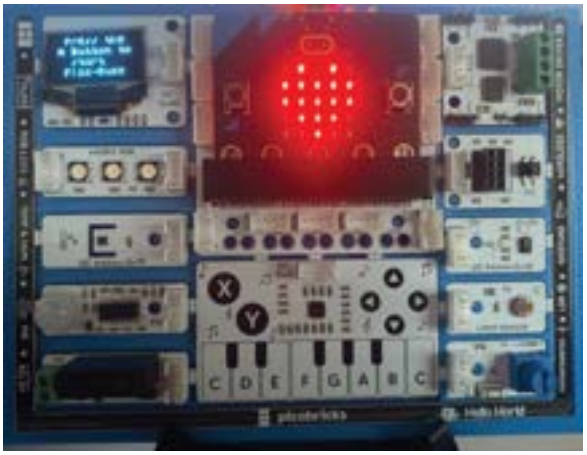
In this project, we will create the Fizz-Buzz game algorithm and code using PicoBricks along with the button, RGB LED, and OLED screen module with Micro:Bit. The Fizz-Buzz game is played by counting numbers from 1 to 100. Starting from 1, when a number that is a multiple of 3 is reached, "Fizz" is said. When a number that is a multiple of 5 is reached, "Buzz" is said. When a number is a multiple of both 3 and 5, "Fizz-Buzz" is said instead of the number.

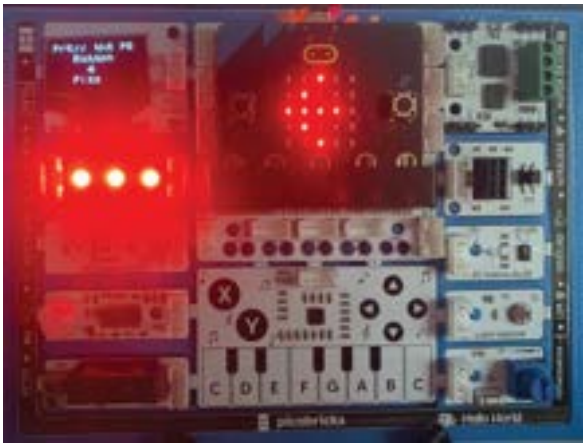
Connection Diagram:

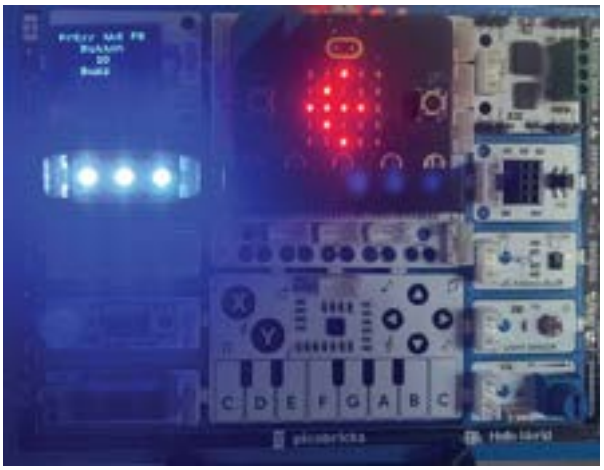
You can prepare this project without making any cable connections.

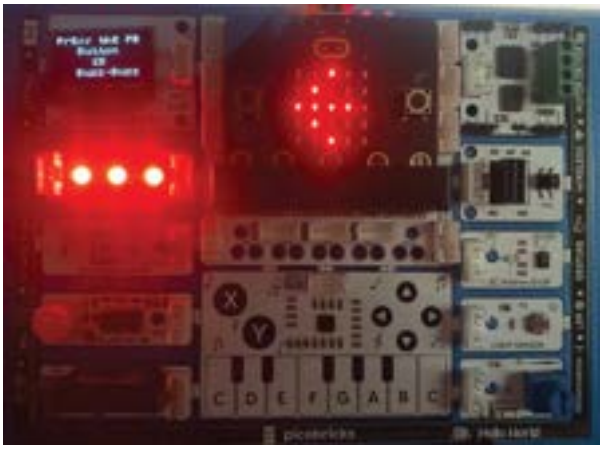
Fizz
Buzz

Project Images:









MicroBlocks Code of The Project:

Fizz-Buzz-Game

```
#Fizz-Buzz Game Projects
from microbit import *
from picobricks import *
import neopixel
import music

# Pin Initialization
Button_Pin = pin2
RGB_Pin = pin8
Num_Leds = 3

# Function Initialization
oled = SSD1306()
oled.init()
oled.clear()
np = neopixel.NeoPixel(RGB_Pin, Num_Leds)

button = Button_Pin.read_digital()
display.show(Image.HEART)

oled.add_text(2,0,"Press the")
oled.add_text(1,1,"A Button to")
oled.add_text(3,2,"start")
oled.add_text(2,3,"Fizz-Buzz")

def left_image():
    display.show(Image('00900:'
                       '09000:'
                       '99999:'
                       '09000:'
                       '00900'))
#Neopixel
np[0] = (0, 0, 0)
np[1] = (0, 0, 0)
np[2] = (0, 0, 0)
np.show()

while True:
    if button_a.is_pressed():
        oled.clear()
        counter=1
        left_image()
        while counter<100:
            oled.add_text(0,0,"Press the PB")
            oled.add_text(3,1,"Button")
            oled.add_text(5,2,str(counter))
            button = Button_Pin.read_digital()
            if button==1:
                counter=counter+1
                music.play(['b'])
                oled.add_text(3,3,"         ")
                np[0] = (0, 0, 0)
                np[1] = (0, 0, 0)
                np[2] = (0, 0, 0)
                np.show()
            if counter % 3 == 0:
                oled.add_text(3,3,"Fizz")
                np[0] = (255, 0, 0)
                np[1] = (255, 0, 0)
                np[2] = (255, 0, 0)
                np.show()
            if counter % 5 == 0:
                oled.add_text(3,3,"Buzz")
                np[0] = (0, 0, 255)
                np[1] = (0, 0, 255)
                np[2] = (0, 0, 255)
                np.show()
            if counter % 15 == 0:
                oled.add_text(3,3,"Fizz-Buzz")
                np[0] = (128, 0, 128)
                np[1] = (128, 0, 128)
                np[2] = (128, 0, 128)
                np.show()
```

Depth Meter

Depth Meter Project

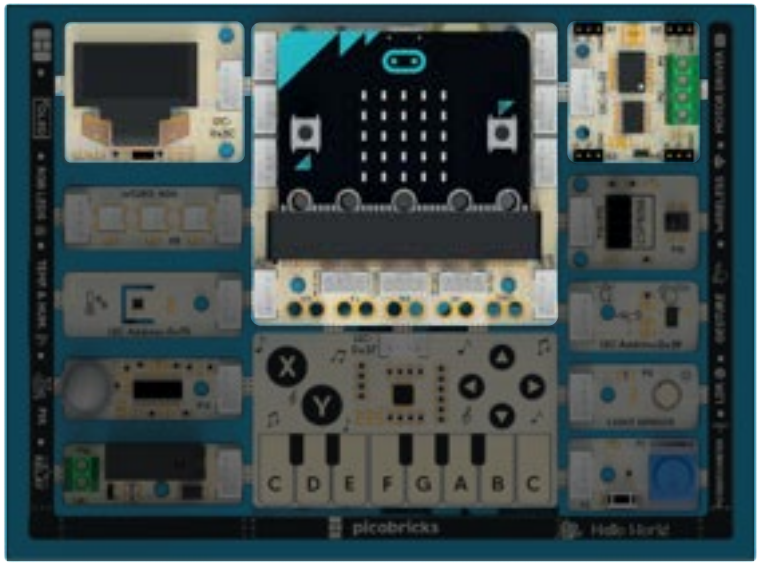
Sometimes, we use depth-measuring machines to measure the quantity of a beverage or mixtures of liquid materials poured into a glass. The fundamental variable that needs to be known for these machines to measure depth is the depth value measured when the container is empty. After defining this information to the measurement devices, the device performs the measurement process by using various sensors such as ultrasonic distance sensor, IR sensor, etc.

Project Details:

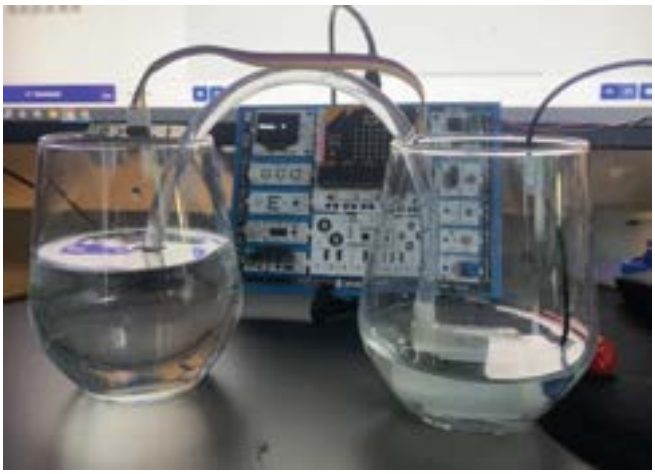
In this project, we will control the water pump connected to the motor driver based on the value measured by the ultrasonic distance sensor we connect to PicoBricks. We will transfer the desired amount of liquid from the container filled with liquid to the empty one. To determine the depth of the glass, we will use the potentiometer & button module.

Connection Diagram:

You can prepare this project without making any cable connections.



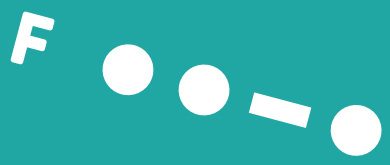
Project Images:



MicroBlocks Code of The Project:

Depth-Meter

```
#Depth Meter Projects
from microbit import *
from picobricks import *

# Pin Initialization
Pot_Pin = pin1
Button_Pin = pin2

# Function Initialization
oled = SSD1306()
oled.init()
oled.clear()
motor = motordriver()
button = Button_Pin.read_digital()

while button_a.is_pressed()==0:
    oled.clear()
    glassDeepValue=10
    oled.add_text(2,0,"Press the")
    oled.add_text(3,1,"A Button")
    oled.add_text(5,2,str(glassDeepValue))
    sleep(100)

while True:
    oled.clear()
    distance = measure_distance()
    print(distance)
    waterDeepValue=glassDeepValue-round(distance)
    oled.add_text(0,2,"Depth:")
    oled.add_text(8,2,str(waterDeepValue))
    if waterDeepValue<(glassDeepValue-4):
        motor.dc(1,150,1)
    else:
        motor.dc(1,0,1)
    sleep(100)
```



Morse Code Cryptography

Morse Code Cryptography Project

People sometimes utilize some passwords to protect their physical belongings or written/visual content. The diversity of symbols used in passwords contributes to the strength of the password. Similarly, not using easily guessable personal data in passwords will enhance the strength of your password.

Cryptography refers to the processes that render readable data incomprehensible to unauthorized individuals.

In Morse Code, there are distinct long and short signals corresponding to each letter. Each character of the text to be encrypted is encoded using the short and long signals in Morse Code. When these signals are combined as a whole and deciphered, the encrypted text is revealed. The long and short signals in Morse Code can be created using sound or light. The most well-known example of this is the SOS distress signal. With a flashlight or similar light source, a call for help can be made by sequentially emitting three long, three short, and three long signals. This is because in Morse Code, the letter "s" is represented by (...) three short signals, and the letter "o" is represented by (---) three long signals

S (●●●) O (▬▬▬) S (●●●)

Morse Code Alphabet:

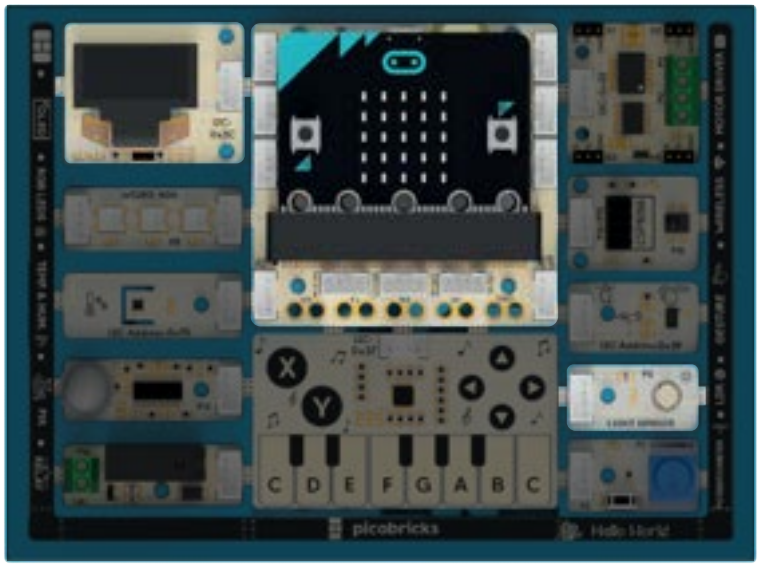
A ●▬	B ▬●●●	C ▬●▬●	D ▬●●	E ●
F ●●▬●	G ▬▬●	H ●●●●	I ●●	J ●▬▬▬
K ▬●▬	L ●▬●●	M ▬▬	N ▬●	O ▬▬▬
P ●▬▬●	Q ▬▬●▬	R ●▬●	S ●●●	T ▬
U ●●▬	V ●●●▬	W●▬▬	X ▬●●▬	Y ▬●▬▬
Z ▬▬●●				

Project Details:

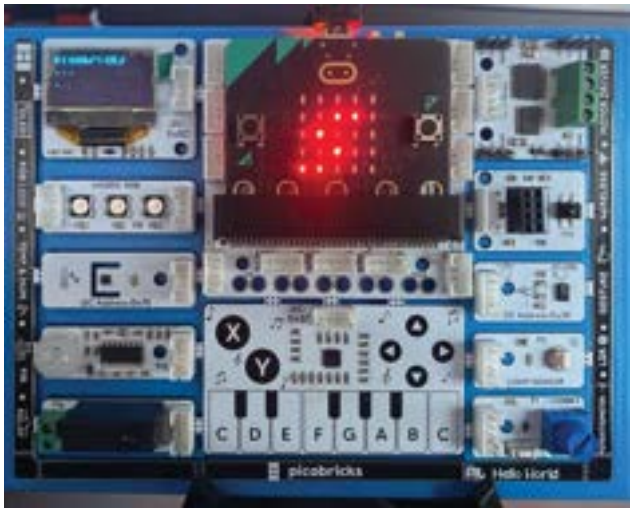
In this project, we will encrypt the specified text by using Morse Code within the code, utilizing the PicoBricks RGB LED module. Each character of the encrypted text will be displayed on the Micro:Bit matrix LED, and its Morse Code equivalent will be shown on the PicoBricks OLED screen.

Connection Diagram:

You can prepare this project without making any cable connections.



Project Images:



MicroBlocks Code of The Project:

Morse-Code-Cryptography

```
#Morse Code Cryptography
from microbit import *
from picobricks import *
import neopixel

# Pin Initialization
RGB_Pin = pin8
Num_Leds = 3

# Function Initialization
oled = SSD1306()
oled.init()
oled.clear()
np = neopixel.NeoPixel(RGB_Pin, Num_Leds)

#Neopixel
np[0] = (0, 0, 0)
np[1] = (0, 0, 0)
np[2] = (0, 0, 0)
np.show()

morse = ['.-','-...','-.-.','-..','.','..-.','--.','....','..',
         '.---', '-.-','.-..','--', '-.','---', '.--.', '--.-',
         '.-.', '...','-','..-', '...-', '.--','-..-', '-.--',
         '--..', '.----','..---','...--','....-','.....', '-....',
         '--...','---..', '----.','-----']

alphabet = ['a','b','c','d','e','f','g','h','i',
         'j', 'k','l','m', 'n','o', 'p', 'q',
         'r', 's','t','u', 'v', 'w','x', 'y',
         'z', '1','2','3','4','5', '6',
         '7','8', '9','0']

while True:
    if button_a.is_pressed():
        passwordText="picobricks"
        for i in range((len(passwordText))):
            oled.clear()
            oled.add_text(0,0,str(passwordText))
            display.show(passwordText[i])
            oled.add_text(0,1,str(morse[alphabet.index(passwordText[i])]))

            j=0
            for j in range(len(morse[alphabet.index(passwordText[i])])):
                oled.add_text(0,2,str(morse[alphabet.index(passwordText[i])][j]))
                if morse[alphabet.index(passwordText[i])][j] == '.' :
                    np[0] = (255, 255, 255)
                    np[1] = (255, 255, 255)
                    np[2] = (255, 255, 255)
                    np.show()
                    sleep(500)
```

Car Parking System

Car Parking System Project

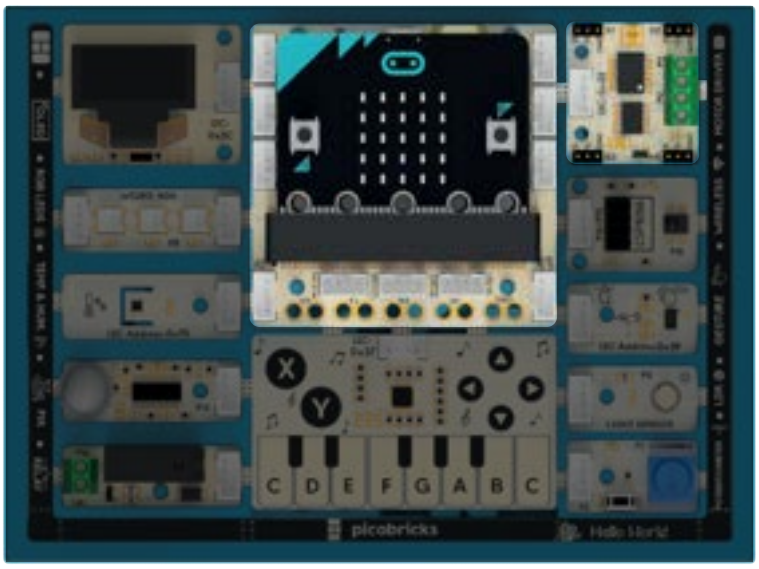
Today, buildings such as hospitals, schools, business centers, etc., often have open or closed parking lots where a large number of people enter and exit. The main reason for the construction of these parking lots is the significant increase in automotive usage in cities. Barrier systems are installed at the entrances of these parking lots to control access. While people were assigned to control these barrier systems in the past, nowadays, with the advancement of sensor technologies, automatic access systems are used. Vehicles are detected using various sensors, the barriers are raised using motor systems, and vehicle passage is allowed.

Project Details:

In this project, we will use the ultrasonic distance sensor connected to PicoBricks to create a barrier system by using waste bins found in our home, depending on the value detected by the sensor. By moving the servo motor connected to the prepared barrier system to the desired angle and we will allow vehicle passage. A checkmark icon (✔) will appear on the Micro:Bit Matrix LED when permission is granted for passage, and a cross (X) icon will appear when permission is denied.

Connection Diagram:

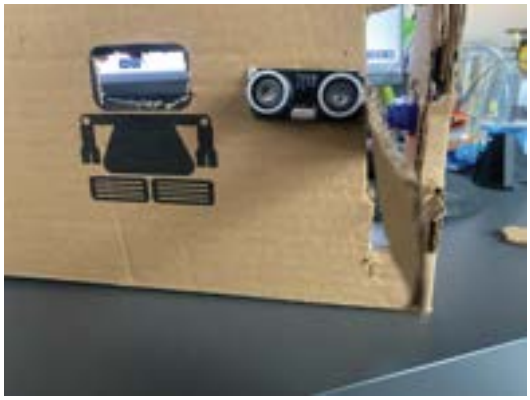
You can prepare this project without making any cable connections.



HC-SR04

Servo Motor

Project Images:



MicroBlocks Code of The Project:

Car-Parking-System

```
#Car Parking System
from microbit import *
from picobricks import *

motor = motordriver()
motor.servo(1,90)

while True:
    distance = measure_distance()
    #print(distance)
    if round(distance)<6:
        motor.servo(1,180)
        display.show(Image.YES)
        sleep(1000)
    else:
        display.show(Image.NO)
        motor.servo(1,180)

```

Table Lamp

Table Lamp Project

Many of us, for reasons such as studying, reading books, preparing reports, etc., prefer to illuminate only our desks instead of turning on all the lights in the room at night. The desk amps we use at home typically use RGB LEDs as light sources. This is because RGB LEDs can emit light in desired color tones. Exposure to certain lights for extended periods can negatively impact our eye health. In such cases, quick transitions between desired colors can be achieved using the color values of RGB LEDs, ranging from 0 to 255. Additionally, RGB LEDs can operate without requiring large power sources. Therefore, desk lamps can be easily illuminated with their own power sources.

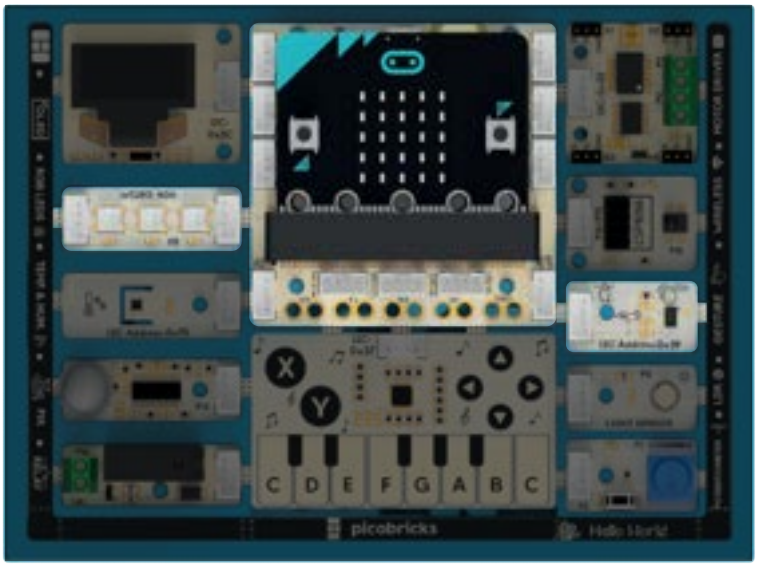
In this project, we will add various features to a table lamp in our home by using PicoBricks modules.
(You can use any table lamp in your home.)

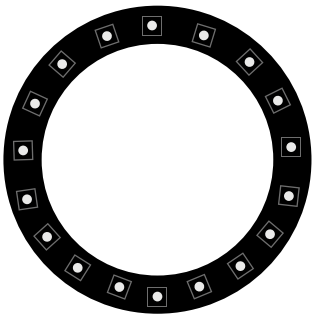
Project Details:

In this project, we will illuminate a desk lamp by using PicoBricks RGB LED and Gesture modules based on the directional movements we make with our hands. After placing the Gesture and RGB LED modules, when we move our hand to the left over the gesture module, the RGB LEDs illuminate according to the color counter. After moving our hand up or down over the gesture module, when we move our hand to the right again, the color of the RGB LED changes. To turn off the desk lamp, you can move your hand to the right over the gesture module.

Connection Diagram:

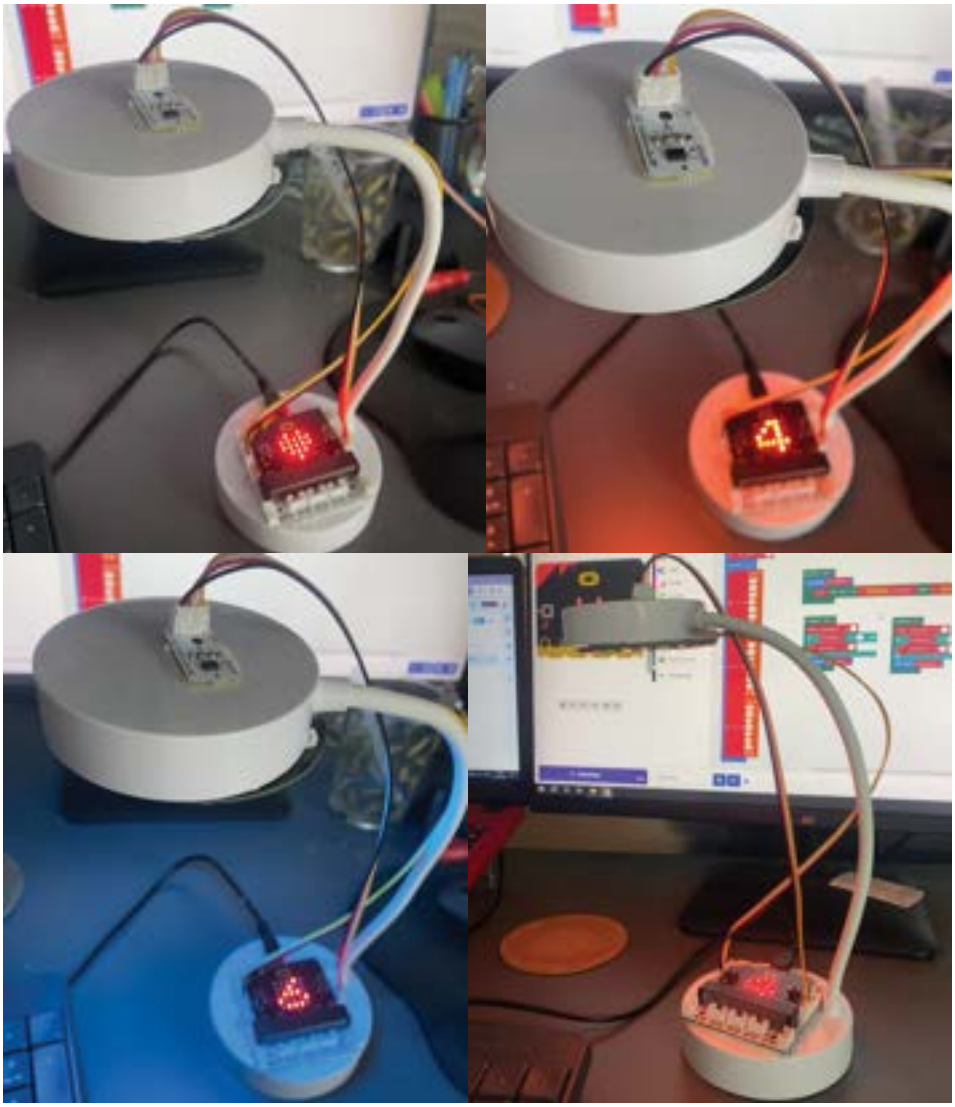
You can prepare this project by breaking PicoBricks modules at proper points.





Addressable Ring
RGB LEDs

Project Images:

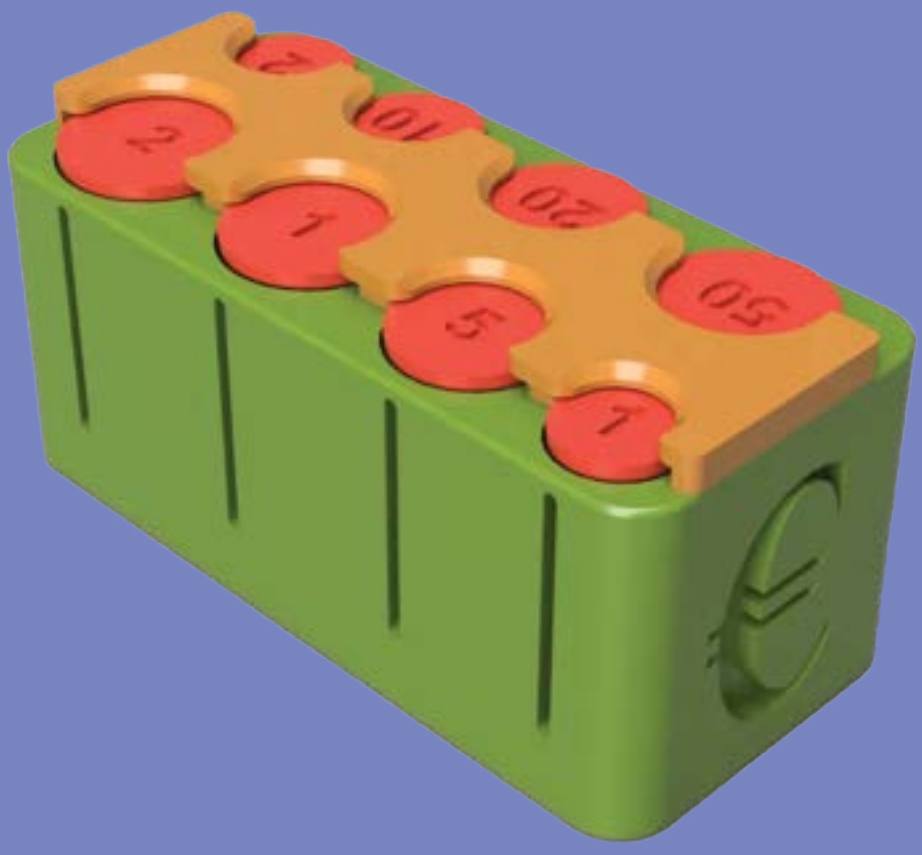


MicroBlocks Code of The Project:

Table-Lamp-Project

```
#Table Lamp Project
from microbit import *
from picobricks import *
import neopixel

# Pin Initialization
RGB_Pin = pin8
Num_Leds = 3 #Enter the number of LEDs

# Function Initialization
apds = APDS9960()
apds.init_gesture_sensor()
np = neopixel.NeoPixel(RGB_Pin, Num_Leds)

#Neopixel
np[0] = (0, 0, 0)
np[1] = (0, 0, 0)
np[2] = (0, 0, 0)
np.show()

colorCounter=0
display.show(Image.HAPPY)
r=[255,128,188,62,139,255,18]
g=[255,135,0,177,50,60,168]
b=[255,193,0,136,0,0,168]

while True:
    gesture = apds.read_gesture()
    if gesture=="RIGHT":
        np[0] = (r[colorCounter], g[colorCounter], b[colorCounter])
        np[1] = (r[colorCounter], g[colorCounter], b[colorCounter])
        np[2] = (r[colorCounter], g[colorCounter], b[colorCounter])
        np.show()
        display.show(Image.HEART)
    elif gesture=="LEFT":
        display.show(colorCounter)
        np[0] = (0, 0, 0)
        np[1] = (0, 0, 0)
        np[2] = (0, 0, 0)
        np.show()
    elif gesture=="UP":
        colorCounter=colorCounter-1
        if colorCounter<0:
            colorCounter=6
        display.show(colorCounter)
    elif gesture=="DOWN":
        colorCounter=colorCounter+1
        if colorCounter>6:
            colorCounter=0
        display.show(colorCounter)
```

Coin Dispenser



Coin Dispenser Project

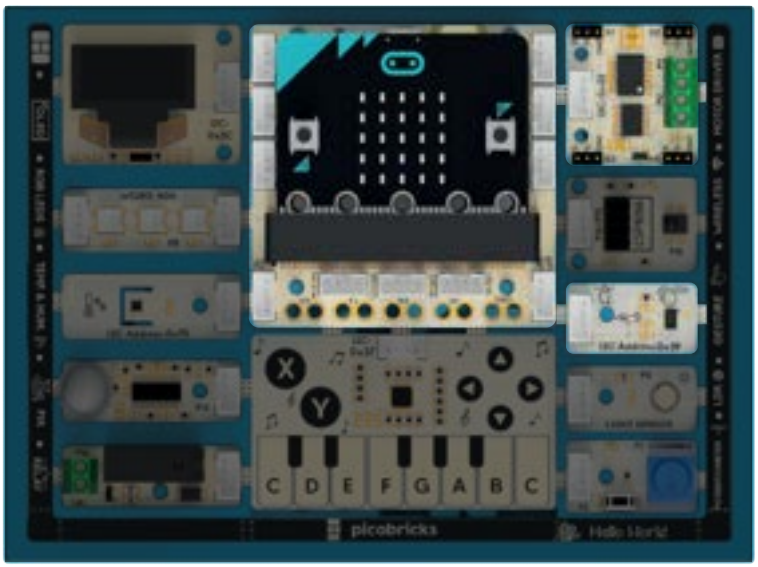
Some people dislike carrying coin in their pockets. This might be due to the extra weight it adds or the noise it makes while walking. For others, collecting coins could be a hobby. However, when collecting coin, we may struggle to separate them. The easiest way to separate coin is by their dimensions. Each coin with different values also has different dimensions. By using the dimensions of the box where we collect the coins, we can quickly separate them. This way, each coin fits into its corresponding box based on its value and can be separated quickly. Moreover, this separation process also makes it easier to count the coins.

Project Details:

In this project, we will use a 3D printer to create a coin dispenser that can be controlled using hand gestures through a gesture module. When we make a rightward gesture with the gesture sensor, the coin dispenser will use a gear system to launch the bottom coin. When we make a leftward gesture, the gear system will pull itself to the left.

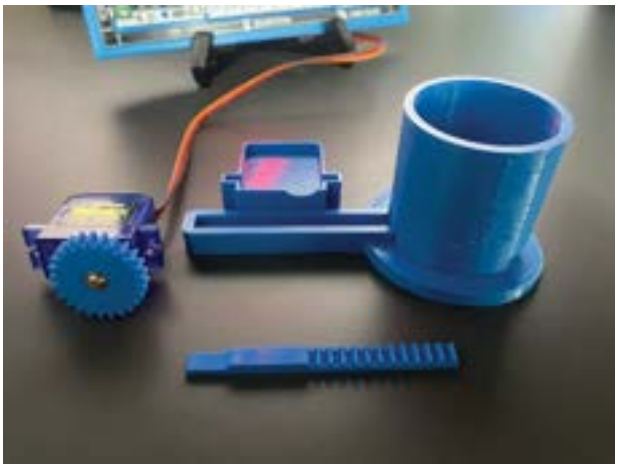
Connection Diagram:

You can prepare this project by breaking down PicoBricks modules at proper points.



Servo Motor

Project Images:



MicroBlocks Code of The Project:

Coin-Dispenser-Project

```
#Coin Dispenser Project
from microbit import *
from picobricks import *

# Function Initialization
apds = APDS9960()
apds.init_gesture_sensor()
motor = motordriver()

display.show(Image.HAPPY)

def left_image():
    display.show(Image('00900:'
                       '09000:'
                       '99999:'
                       '09000:'
                       '00900'))

def right_image():
    display.show(Image('00900:'
                       '00090:'
                       '99999:'
                       '00090:'
                       '00900'))

```

Gesture Controlled ARM Pan Tilt

Gesture Controlled ARM Pan Tilt Project

Robot arms have replaced human labor in the industrial field. They undertake tasks such as carrying and rotating loads that are too heavy or large for a human to handle in factories. Their ability to be positioned with precision up to one-thousandth of a millimeter surpasses the precision achievable by human hands. When you watch production videos of automobile factories, you will see how crucial robot arms are. They are called "robots" because they can perform the same task infinitely by being programmed. The reason for calling them "arms" is because they have an articulated structure similar to our arms. The number of axes a robot arm can rotate and move in determines its degrees of freedom. Robot arms are also used in carving and shaping aluminum and various metals. These devices, known as 7-axis CNC routers, can shape metals similar to how a sculptor shapes clay.

Depending on the purpose of use in robot arms, both stepper motors and servo motors are utilized. PicoBricks enables you to create projects using servo motors.

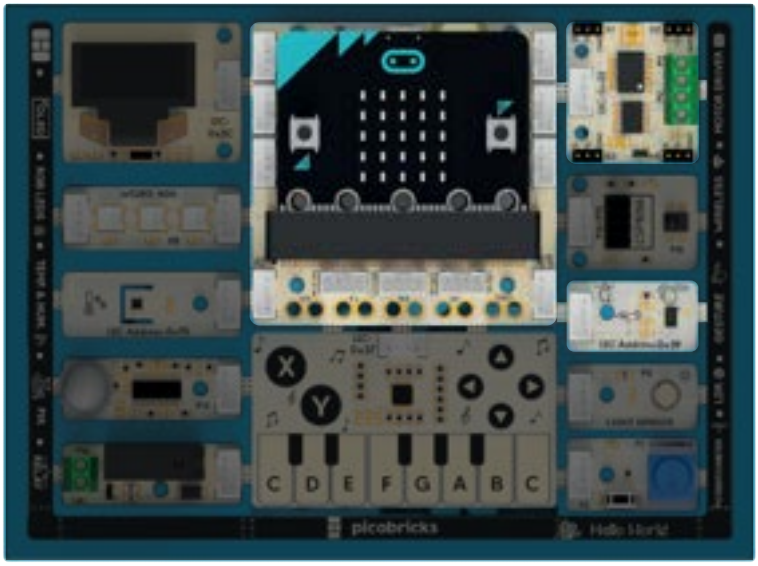
Project Details:

In this project, we will use the "gesture" feature of the PicoBricks gesture module to detect up-down, right, and left hand movements, and move a pan-tilt system accordingly. Additionally, when we press the "A" button on the Micro:Bit, we will reset the servo motors to their initial positions to center the system.

Note: By mounting the RGB LED module on the front surface of this system, we can create a lighting system that can move in two axes.

Connection Diagram:

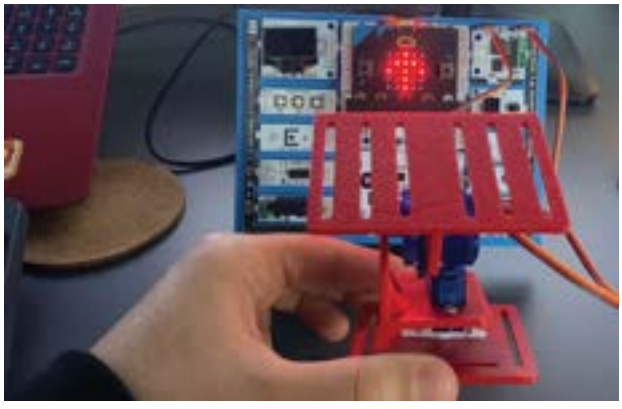
You can prepare this project by breaking down PicoBricks modules at proper points.

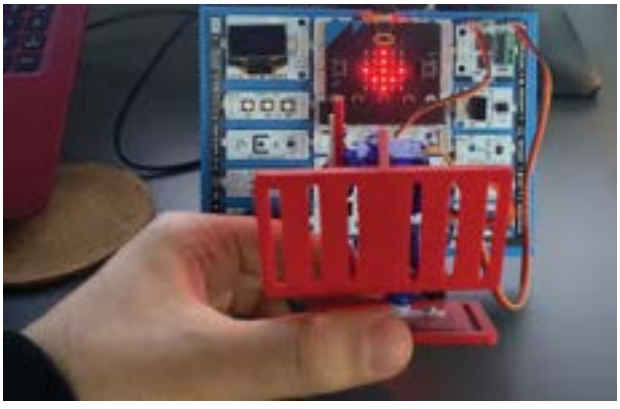


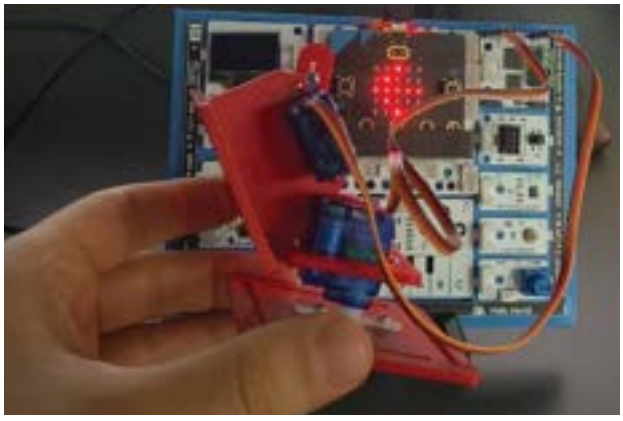
+

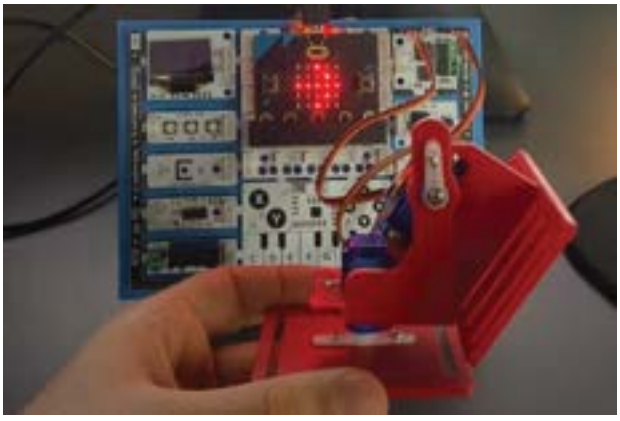
Servo Motor

Project Images:

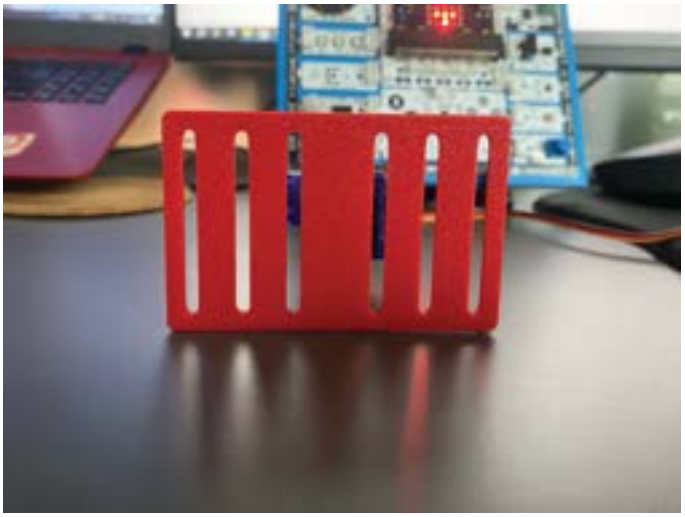


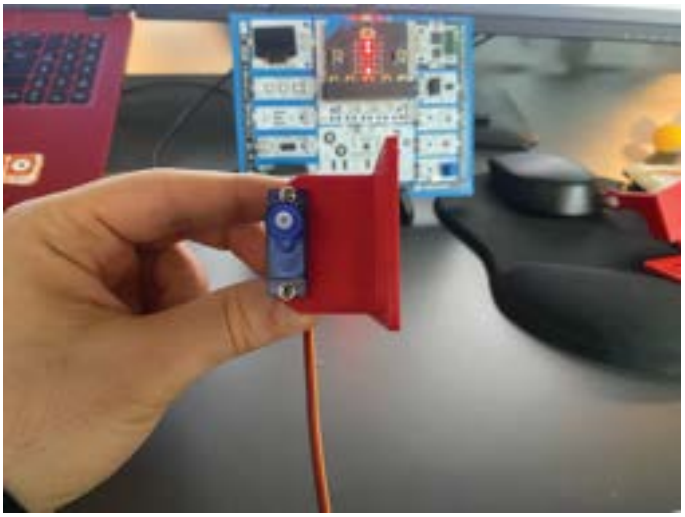


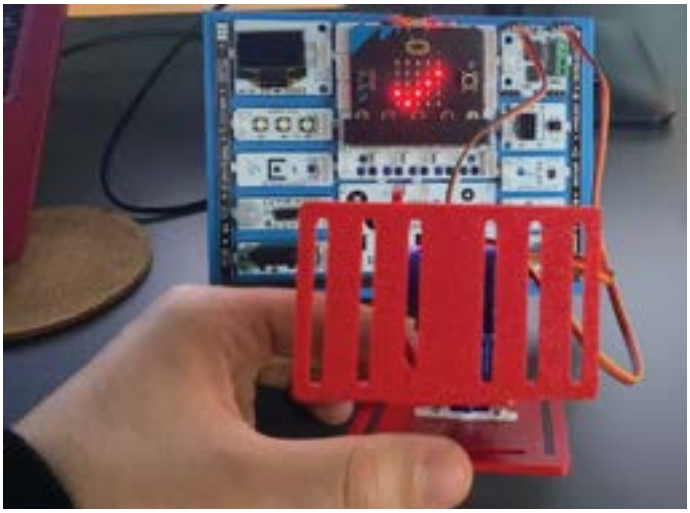


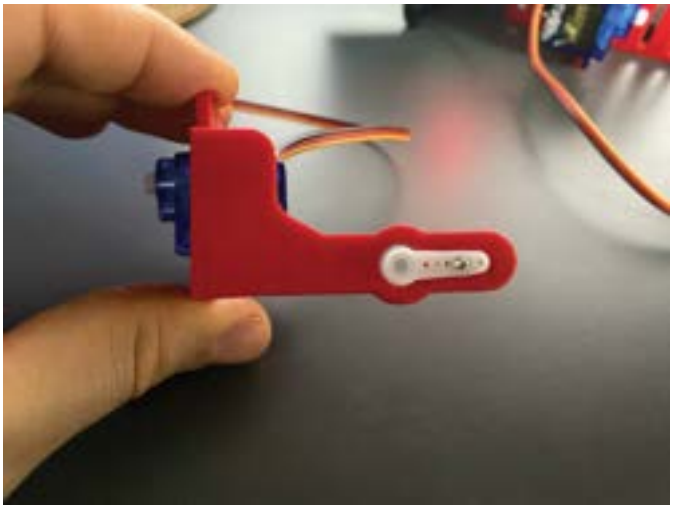


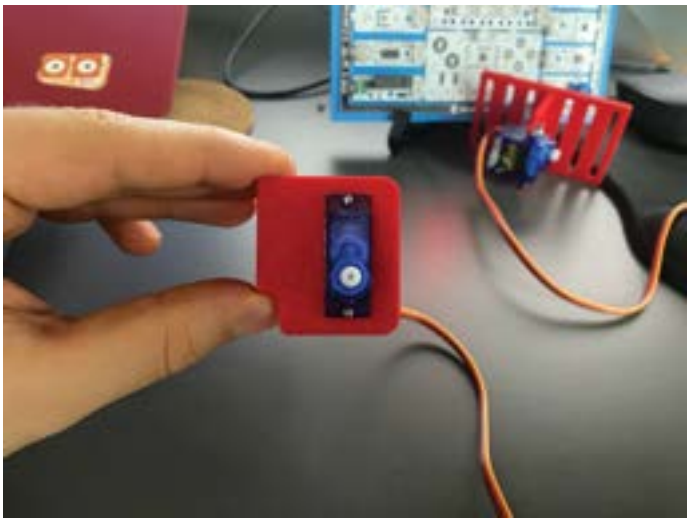
Installation Images:











MicroBlocks Code of The Project:

Gesture-Controlled-Arm

```
#Gesture Controlled Arm Pantilt Project
from microbit import *
from picobricks import *
import music

# Function Initialization
apds = APDS9960()
apds.init_gesture_sensor()
motor = motordriver()

motor.servo(1,0)
motor.servo(2,0)

display.show(Image.YES)

def left_image():
    display.show(Image('00900:'
                       '09000:'
                       '99999:'
                       '09000:'
                       '00900'))

def right_image():
    display.show(Image('00900:'
                       '00090:'
                       '99999:'
                       '00090:'
                       '00900'))

def up_image():
    display.show(Image('00900:'
                       '09990:'
                       '90909:'
                       '00900:'
                       '00900'))

def down_image():
    display.show(Image('00900:'
                       '00900:'
                       '90909:'
                       '09990:'
                       '00900'))

while True:
    gesture = apds.read_gesture()
    if gesture=="RIGHT":
        motor.servo(1,0)
        right_image()
        music.play(['c'])
    elif gesture=="LEFT":
        motor.servo(1,180)
        left_image()
        music.play(['c'])
    elif gesture=="UP":
        motor.servo(2,0)
        up_image()
        music.play(['c'])
    elif gesture=="DOWN":
        motor.servo(2,180)
        down_image()
        music.play(['c'])
    elif button_a.is_pressed():
        motor.servo(1,0)
        motor.servo(2,0)
        display.show(Image.YES)
```

3D Labyrinth

3D Labyrinth Project

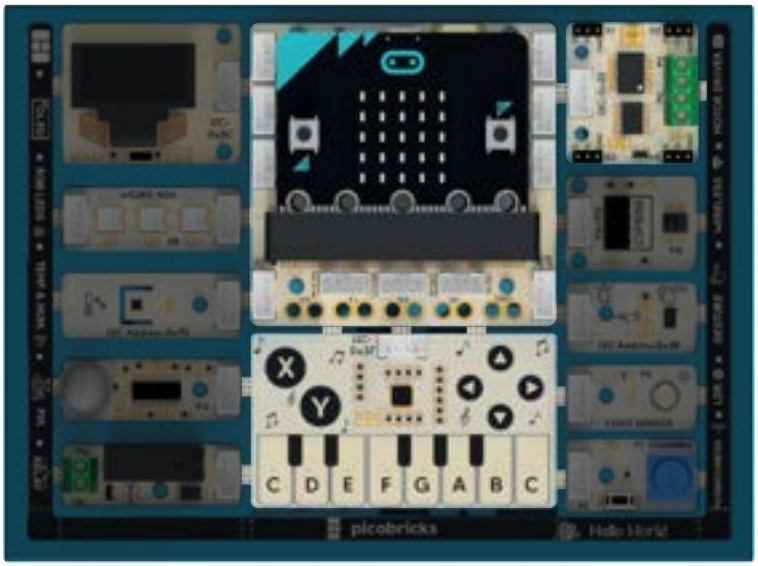
There are multiple ways to exit a maze, but the most well-known method is to follow the wall with your left/right hand. Although this method may take some time, you can definitely get out of the maze by consistently touching the wall. Maze tests enhance problem-solving skills. Someone who frequently solves maze tests can quickly come up with solutions to the problems they encounter. In this project, we will design a maze and the necessary mechanical parts to move the maze by using a 3D printer.

Project Details:

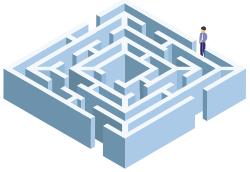
Let's create a maze project that can move in right, left, up, and down directions by assembling 3D printer parts as shown in the visuals. To ensure the movement of the maze in this project, we will utilize two servo motors connected to the PicoBricks motor driver. The direction keys on the PicoBricks Touch & Piano module will be used to move the servo motors in the desired direction. By using the Right, Left, Up, and Down direction keys, we will control the direction and attempt to navigate the ball placed inside the maze to the exit.

Connection Diagram:

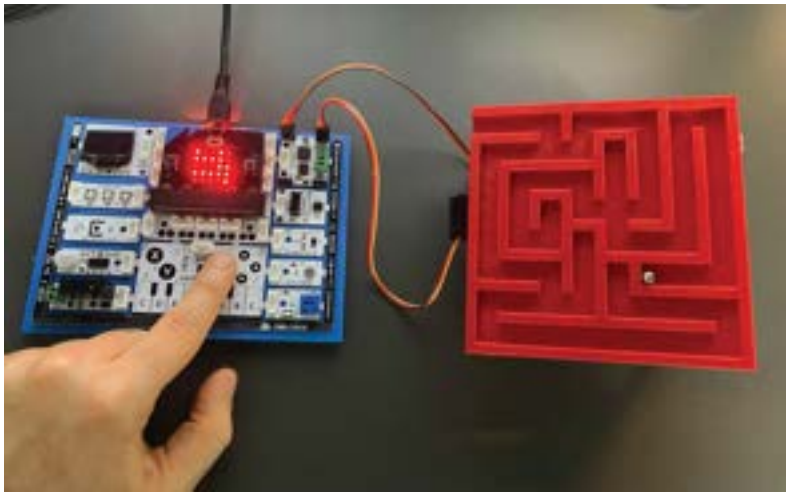
You can prepare this project by breaking down PicoBricks modules at proper points.

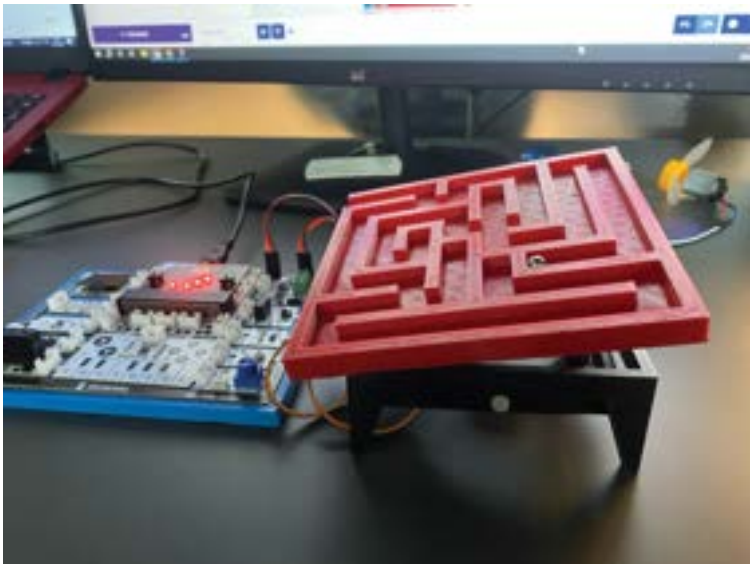


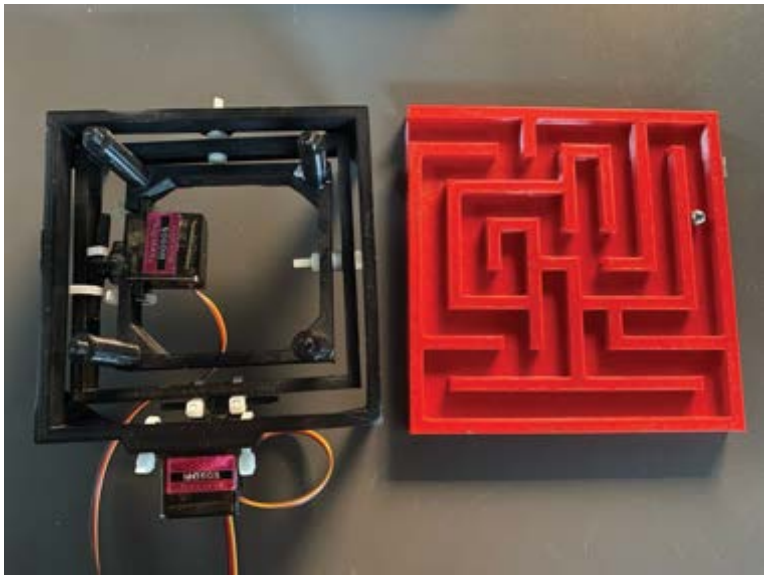
Servo Motor



Project Images:







MicroBlocks Code of The Project:

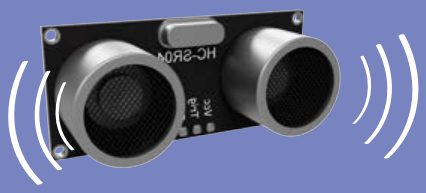
3D-Labyrinth-Project

```
#3D Labyrinth Project
from microbit import *
from picobricks import *

# Function Initialization
oled = SSD1306()
oled.init()
motor = motordriver()
apds = APDS9960()
apds.init_gesture_sensor()
pin15.set_pull(pin15.PULL_UP)
ir = IRM()

def labyrinth_image():
    display.show(Image('90090:'
                       '99990:'
                       '00099:'
                       '99999:'
                       '00000'))
labyrinth_image()
servo1Value=45
servo2Value=45

while True:
    gesture = apds.read_gesture()
    oled.clear()
    motor.servo(1,servo1Value)
    motor.servo(1,servo2Value)
    sleep(100)
    if gesture == "UP":
        servo1Value=servo1Value-1
        if servo1Value==15:
            servo1Value=16
    if gesture == "DOWN":
        servo1Value=servo1Value+1
        if servo1Value==58:
            servo1Value=57
    if gesture == "RIGHT":
        servo2Value=servo2Value+1
        if servo2Value==80:
            servo2Value=79
    if gesture == "LEFT":
        servo2Value=servo2Value-1
        if servo2Value==30:
            servo2Value=31
    oled.add_text(0,0,str(servo1Value))
    oled.add_text(0,1,str(servo2Value))
    if button_a.is_pressed():
        labyrinth_image()
        servo1Value=45
        servo2Value=45
    gesture=0

```

The STL Files of The Project:

You can access the STL files of the project by scanning the QR code or opening the link in your browser.


HC-SR04
Vcc
Trig


HC-SR04
Vcc
Trig

Radar


HC-SR04
Vcc
Trig

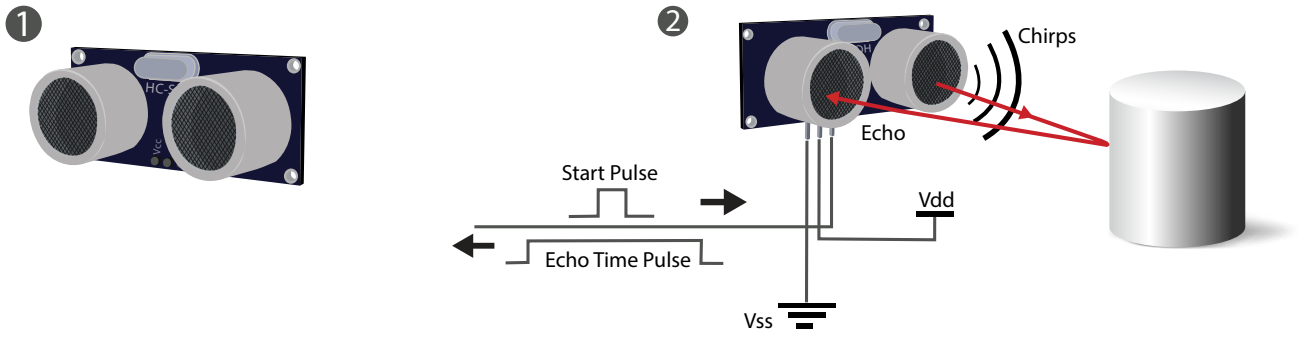

HC-SR04
Vcc
Trig

Radar Project

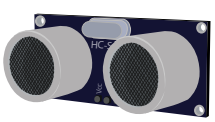
The radar is a device that detects objects in its surroundings, their direction of movement, their speeds, and other values through radio waves. The effective, or ranges, of radars can vary. Depending on this variation, their applications change. Radars are frequently used to ensure security in various vehicles such as ships, airplanes, etc., and in military areas.

Since radar operates with radio waves, which sensors on PicoBricks or in the set can we use to create a sample radar project? Among the PicoBricks modules and sensors in the set, the sensors with distance measuring capability are the Ultrasonic Distance Sensor (HC-SR04) and the gesture module. Due to the limited range of distances that the gesture module can measure, the ultrasonic distance sensor would be more suitable for a project of this kind.

Ultrasonic distance sensors detect objects around them by using sound waves. As explained in the diagram below, the distance to the object in front is determined by calculating the time it takes for the sound wave emitted from the Trig pin to hit the Echo pin.



In this project, we will create a radar project using the PicoBricks, ultrasonic distance sensor, and servo motor.

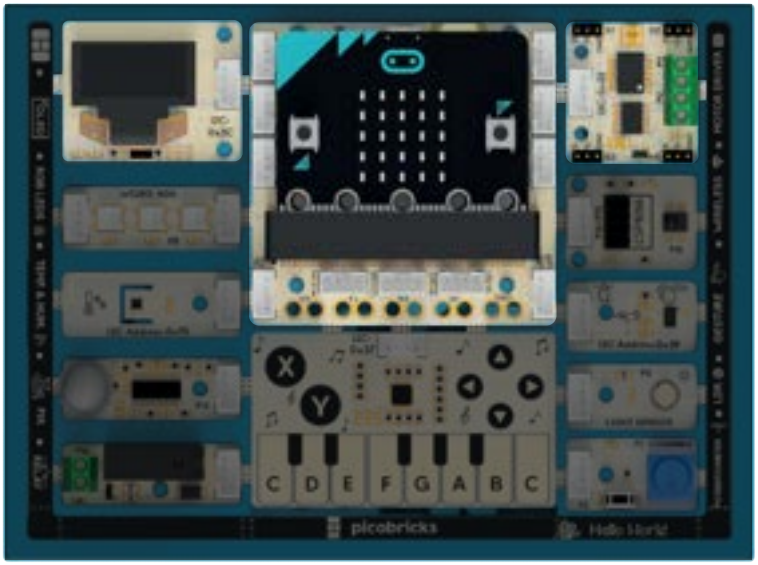


Project Details:

In this project, we will implement a radar project by rotating the servo motor connected to PicoBricks' motor driver based on the value detected by the ultrasonic distance sensor connected to the P1-P2 pins, within a 0-180 degree angle. The radar will move in the range of 0-180 degrees until it detects a value within the range determined by the potentiometer module. If an object is detected within the specified range, it will stop moving, emit a warning sound from the buzzer, and display the distance and angle of the object from the radar on the OLED screen.

Connection Diagram:

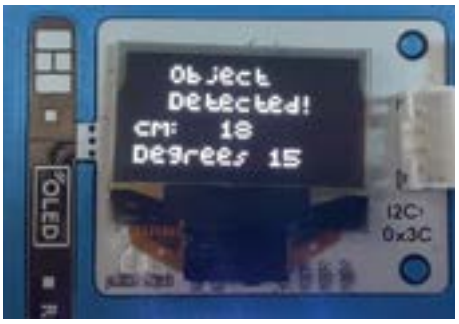
You can prepare this project by breaking down PicoBricks modules at proper points.



Servo Motor

HC-SR04

Project Images:



The STL Files of The Project:

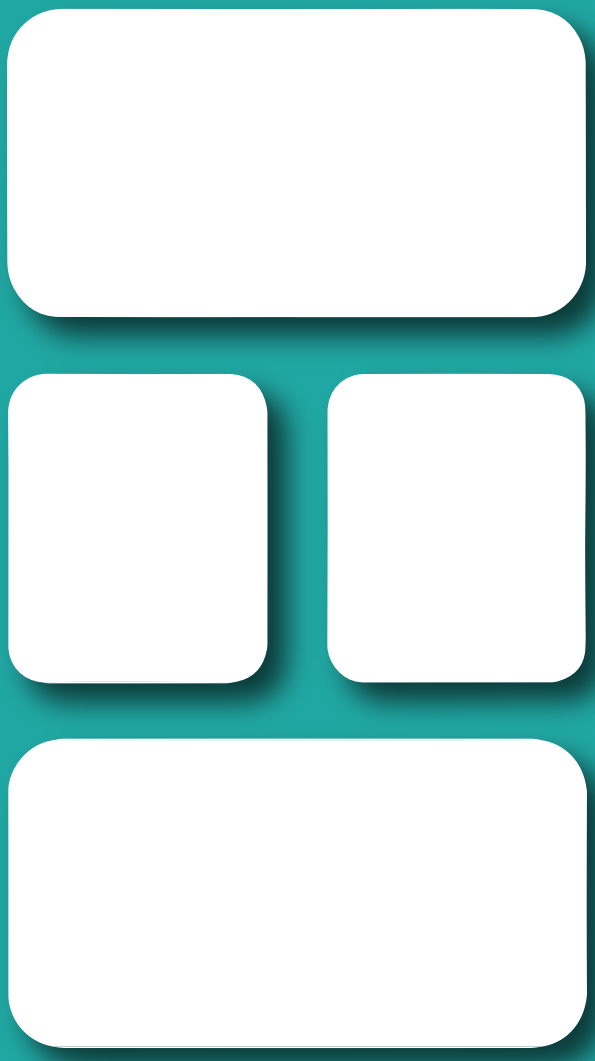
You can access the STL files of the project by scanning the QR code or opening the link in your browser.

MicroBlocks Code of The Project:

Radar-Project

```
#PicoBricks Radar Project
from microbit import *
from picobricks import *
import music

# Pin Initialization
Button_Pin = pin2
Pot_Pin = pin1

# Function Initialization
oled = SSD1306()
oled.init()
motor = motordriver()

motor.servo(1,90)
angleServo=0
radarRange=0
c=1
button = Button_Pin.read_digital()
while Button_Pin.read_digital()==0:
    oled.clear()
    pot = Pot_Pin.read_analog()
    radarRange=round(round( pot - 0 ) * ( 100 - 0 ) / ( 1023 - 0 ) + 0)
    oled.add_text(0,0,"Radar Range:")
    oled.add_text(5,1,str(radarRange))
    sleep(50)
oled.clear()

while True:
    oled.clear()
    distance = measure_distance()
    while round(measure_distance())>= radarRange:
        oled.add_text(2,2,"Scaning...")
        motor.servo(1,angleServo)
        if c==1:
            angleServo=angleServo+5
        if c==0:
            angleServo=angleServo-5
        if angleServo==180:
            c=0
        if angleServo==0:
            c=1
        sleep(10)
    oled.clear()
    objectDistance=round(distance)
    oled.add_text(2,0,"Object")
    oled.add_text(2,1,"Detected!")
    oled.add_text(0,2,"cm:")
    oled.add_text(5,2,str(objectDistance))
    oled.add_text(0,3,"Degreess")
    oled.add_text(8,3,str(angleServo))
    music.play(['c'])

```

PicoBricks Logo Lamp



PicoBricks Logo Lamp Project

In these days, the usage areas of 3D printers have significantly expanded. 3D printers are utilized for various purposes in many sectors such as healthcare, automotive, education, and more. The raw materials used by 3D printers for printing can vary depending on the intended use of the produced part. For instance, with a 3D printer that uses cement as a raw material, we can print a house. In this project, we will prepare a lamp by creating color animations using the 3D-printed PicoBricks Logo and the PicoBricks RGB LED module.

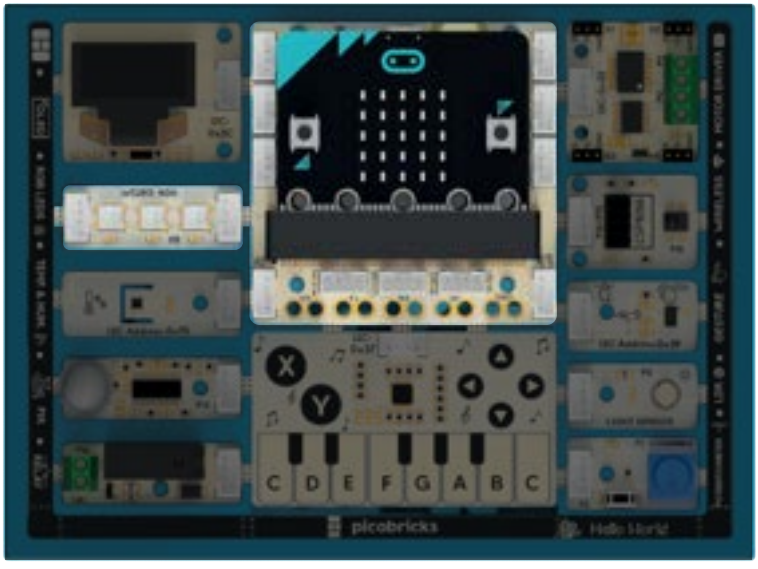
Color animations are used in various areas such as advertising panels, celebration areas, etc., to attract attention. In these systems, which are created by illuminating a LED with different colors at specific time intervals, RGB LEDs are commonly used. The main reason for the use of RGB LEDs in these systems is the ability to easily create desired color tones by utilizing color values ranging from 0 to 255.

Project Details:

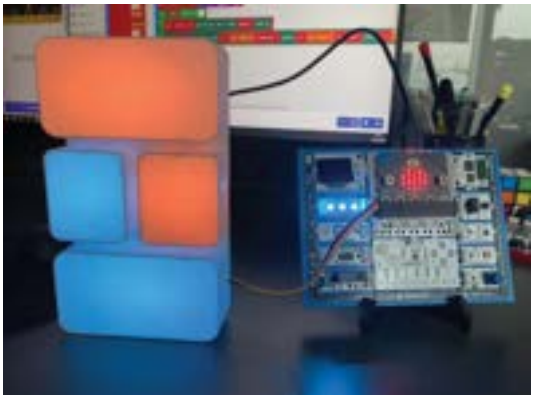
In this project, we will create color animations by placing the addressable RGB LEDs connected to the PicoBricks RGB LED module inside the 3D-printed PicoBricks Logo lamp.

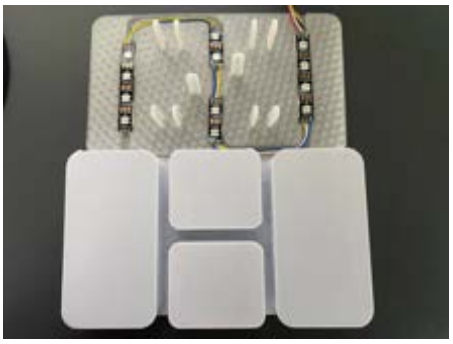
Connection Diagram:

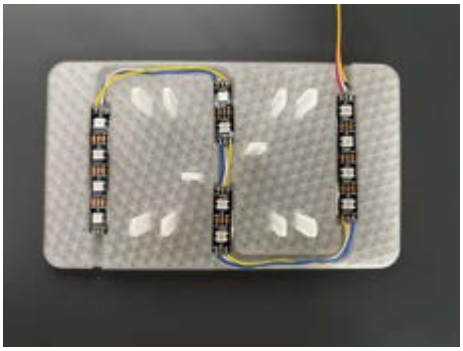
You can prepare this project by breaking down PicoBricks modules at proper points.

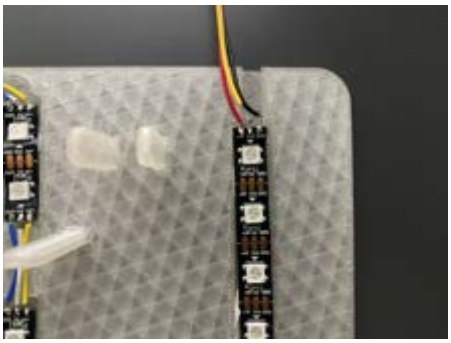


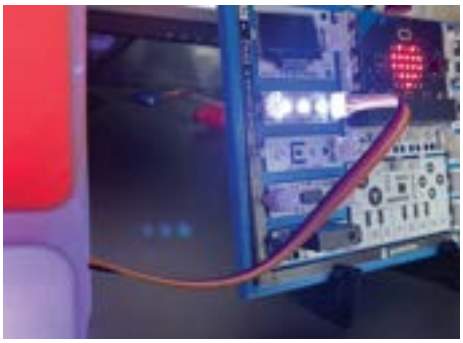
Project Images:











The STL Files of The Project:

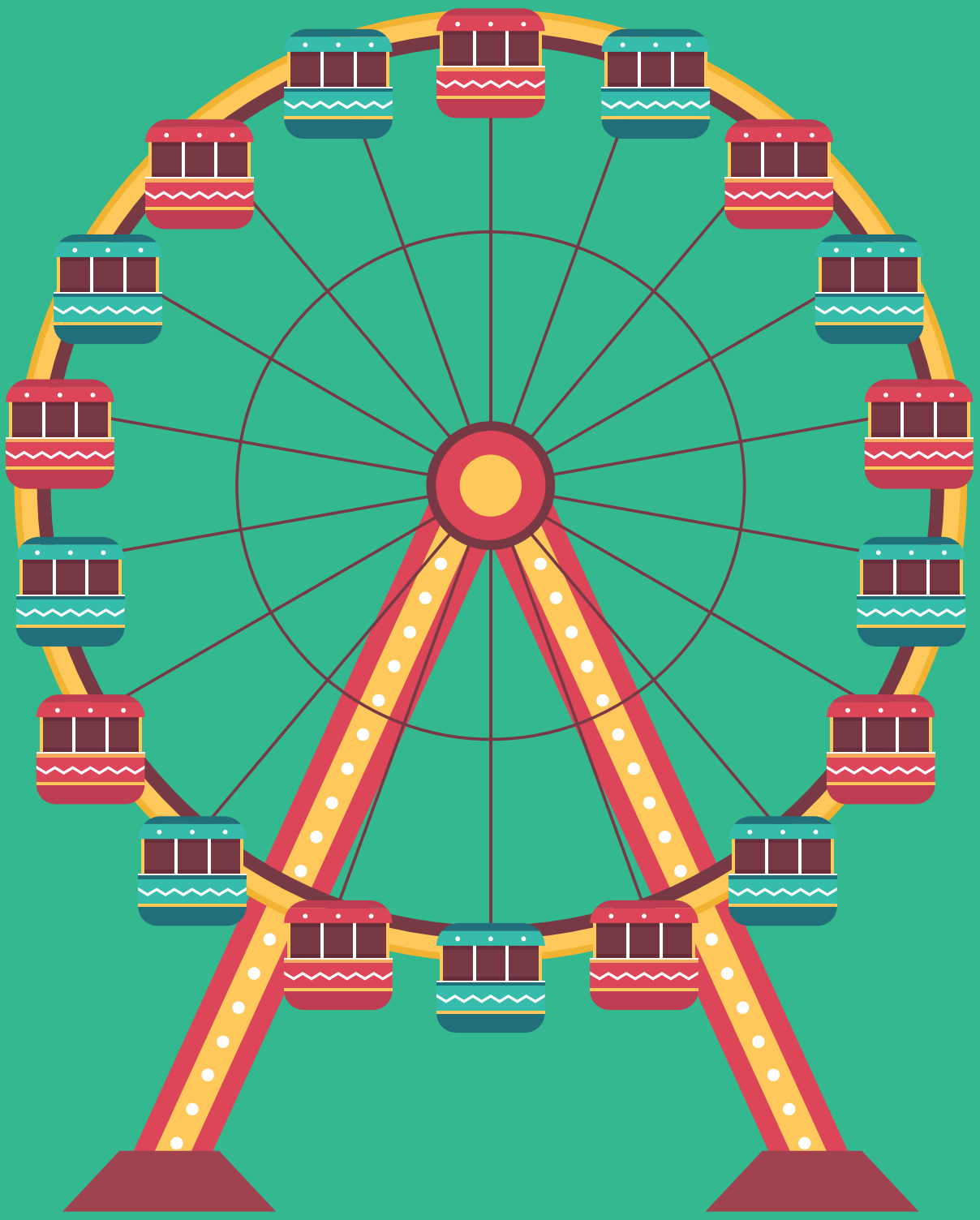
You can access the STL files of the project by scanning the QR code or opening the link in your browser.

MicroBlocks Code of The Project:

PicoBricks-Logo-Lamp-Project

```
#PicoBricks Logo Lamp Project
from microbit import *
from picobricks import *
import neopixel
import random

# Pin Initialization
RGB_Pin = pin8
Num_Leds = 11 #Enter the number of LEDs

# Function Initialization
np = neopixel.NeoPixel(RGB_Pin, Num_Leds)

#Neopixel
np[0] = (0, 0, 0)
np[1] = (0, 0, 0)
np[2] = (0, 0, 0)
np.show()

display.show(Image.HAPPY)

r=[18,128,62,188,139,255]
g=[168,135,177,0,50,60]
b=[168,193,136,0,0,0]

while True:
    randColor=random.randint(0, 5)
    for i in range(11):
        Num_Leds=i
        np[i] = (r[randColor], g[randColor], b[randColor])
        np.show()
        sleep(100)
```

Ferris Wheel



Ferris Wheel Project

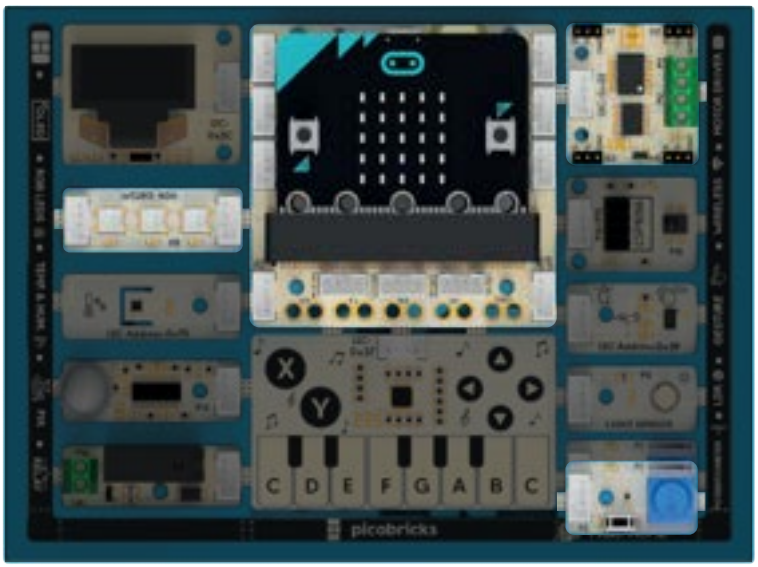
A Ferris wheel is an amusement ride where seats are positioned around a circle that rotates on a central axis. PicoBricks Ferris Wheel is a project kit where you can adjust the speed of the Ferris wheel based on the value of the potentiometer by using the potentiometer, motor driver, and mainboard module on PicoBricks.

Project Details:

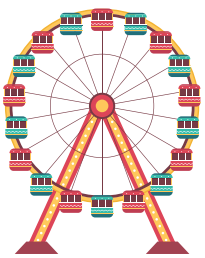
In this project, we will control the rotation speed of the Ferris Wheel based on the speed of the DC motor by using the PicoBricks Potentiometer module.

Connection Diagram:

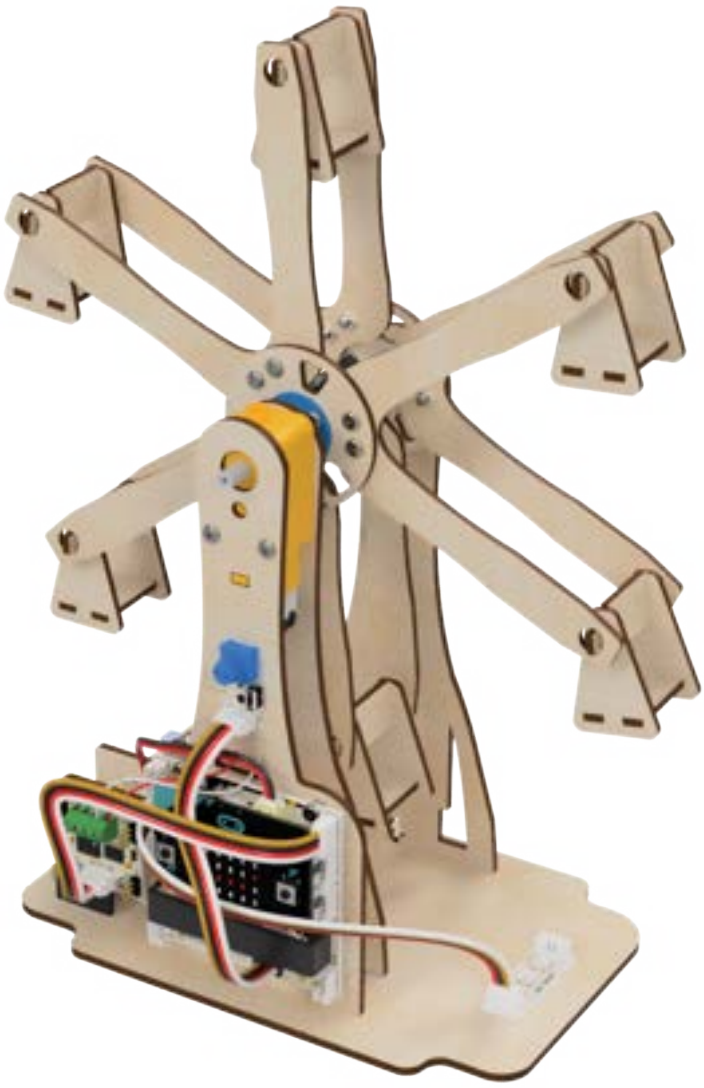
You can prepare this project by breaking down PicoBricks modules at proper points.



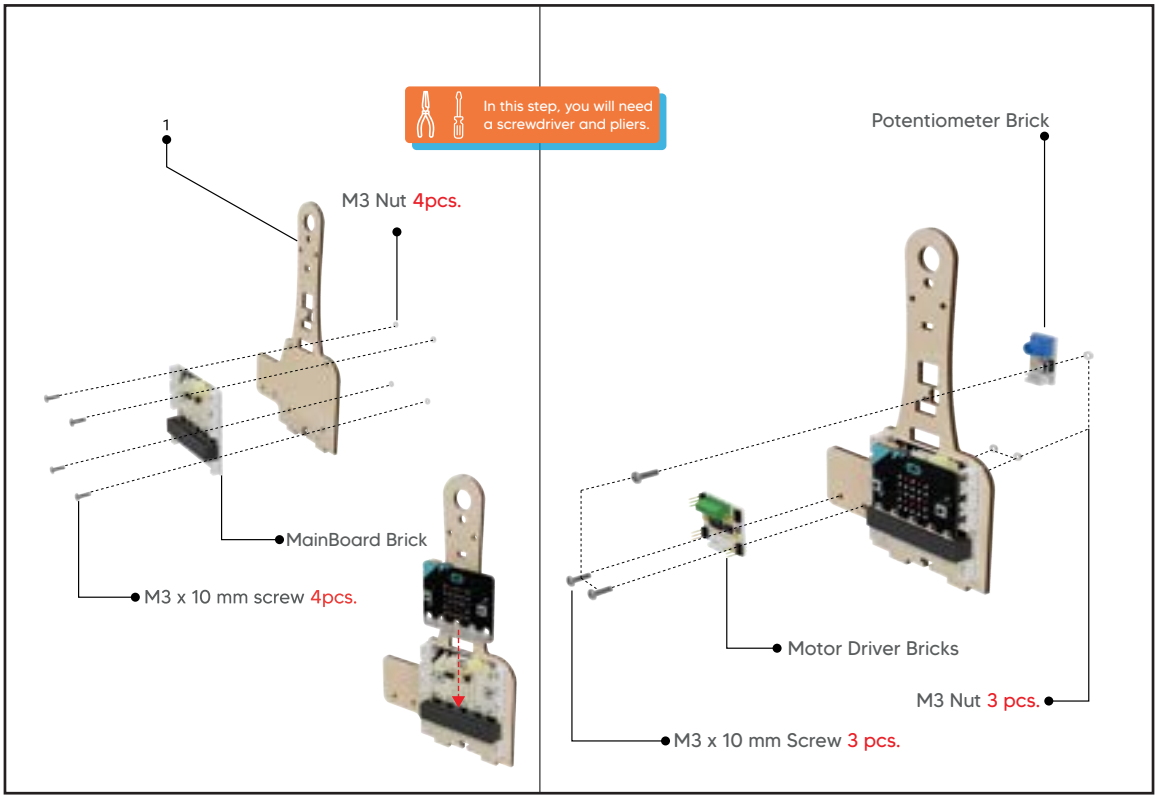
DC Motor

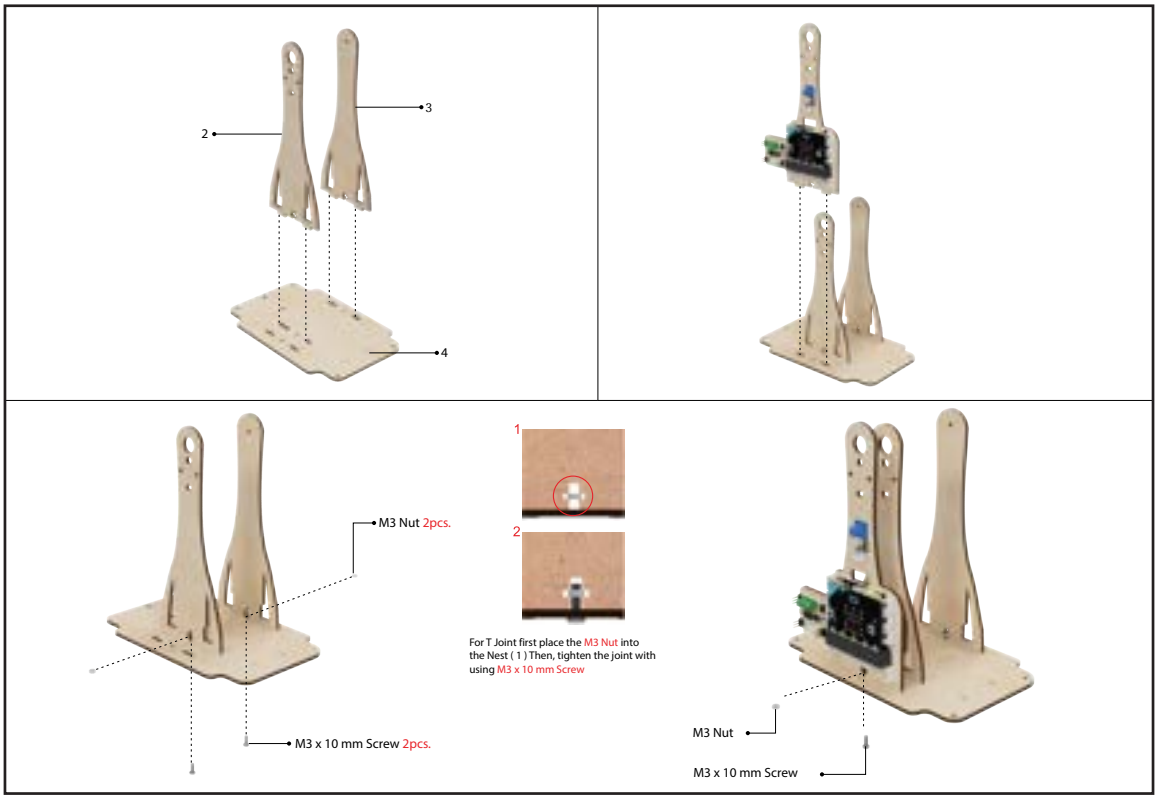


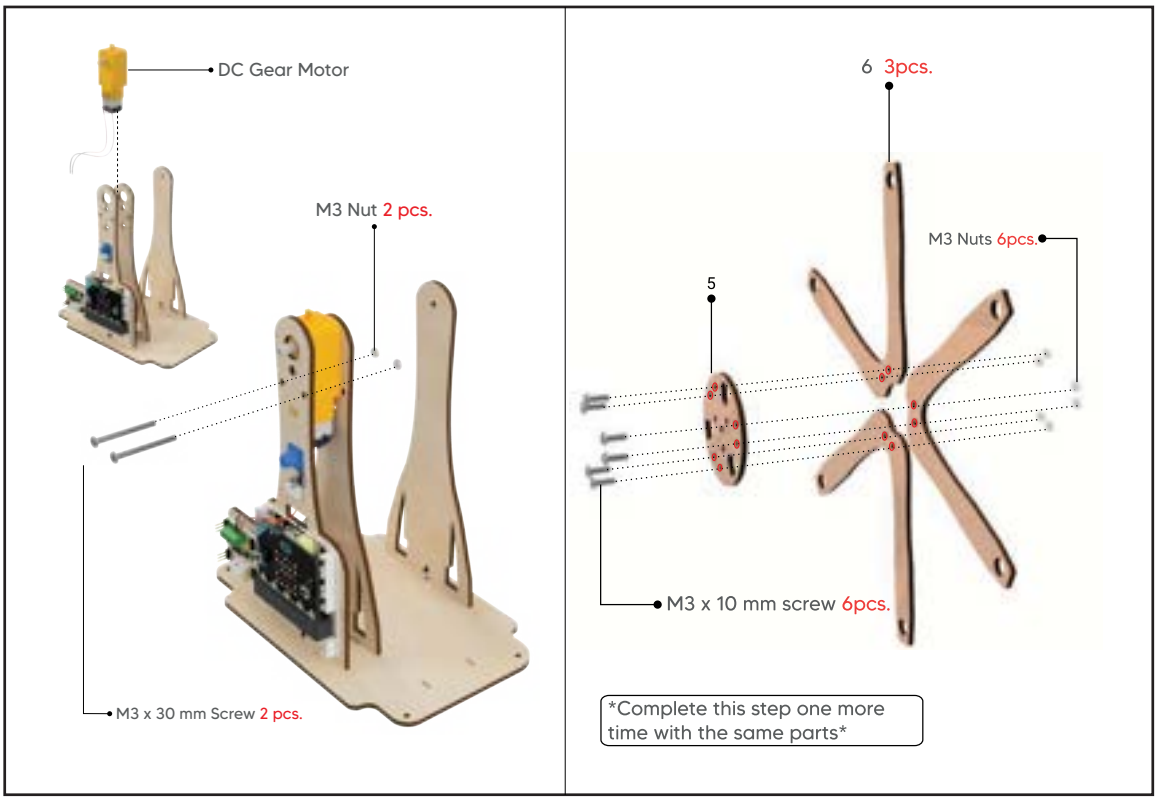
Project Images:

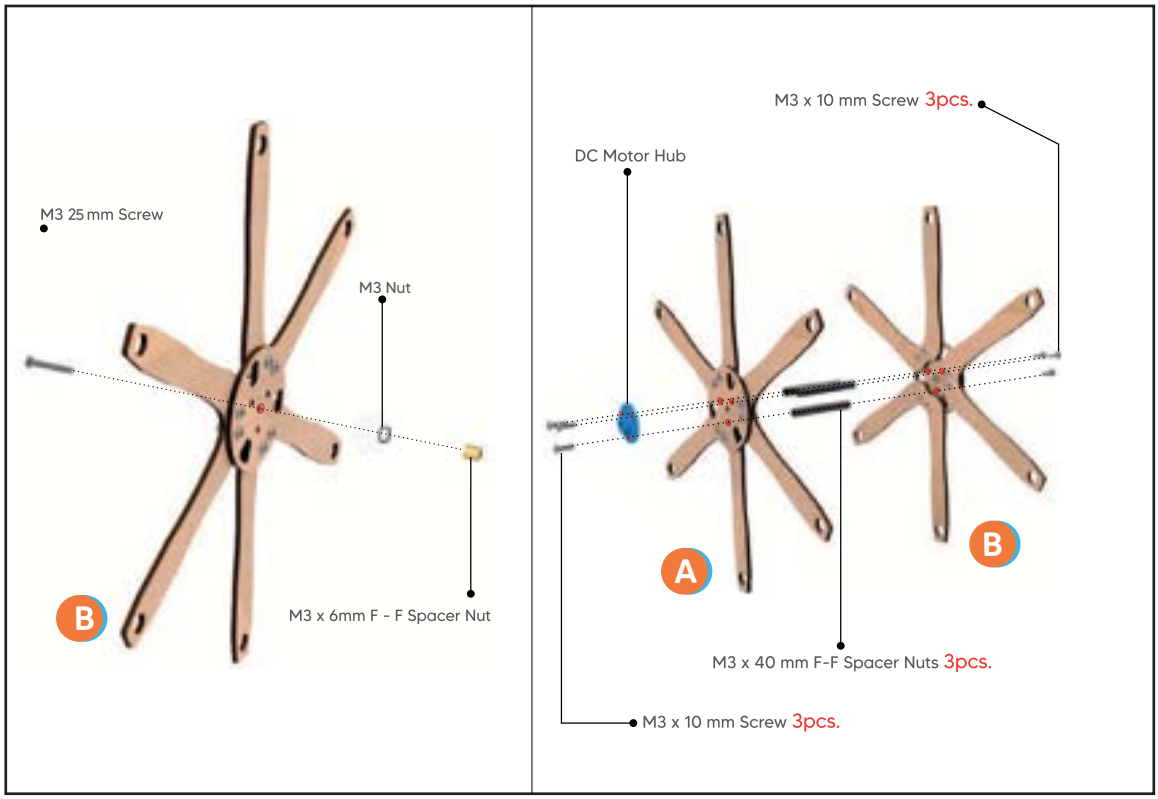


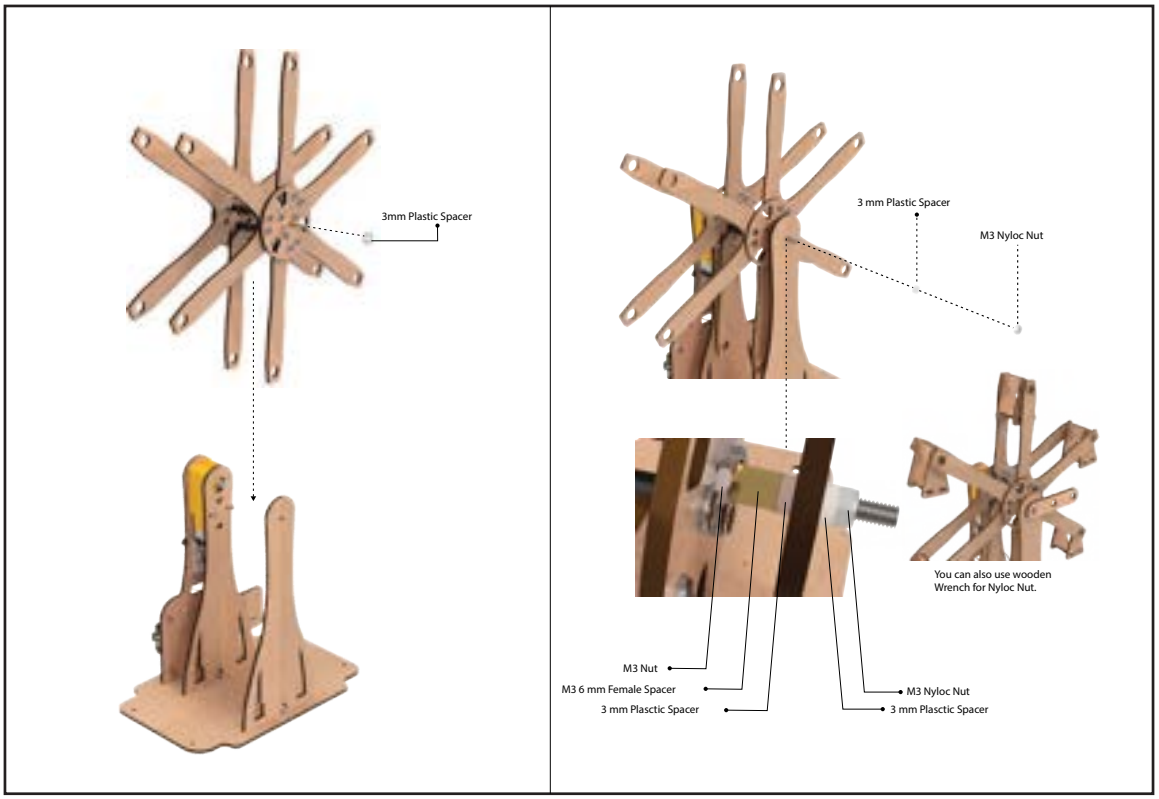
Setup Steps of The Project:

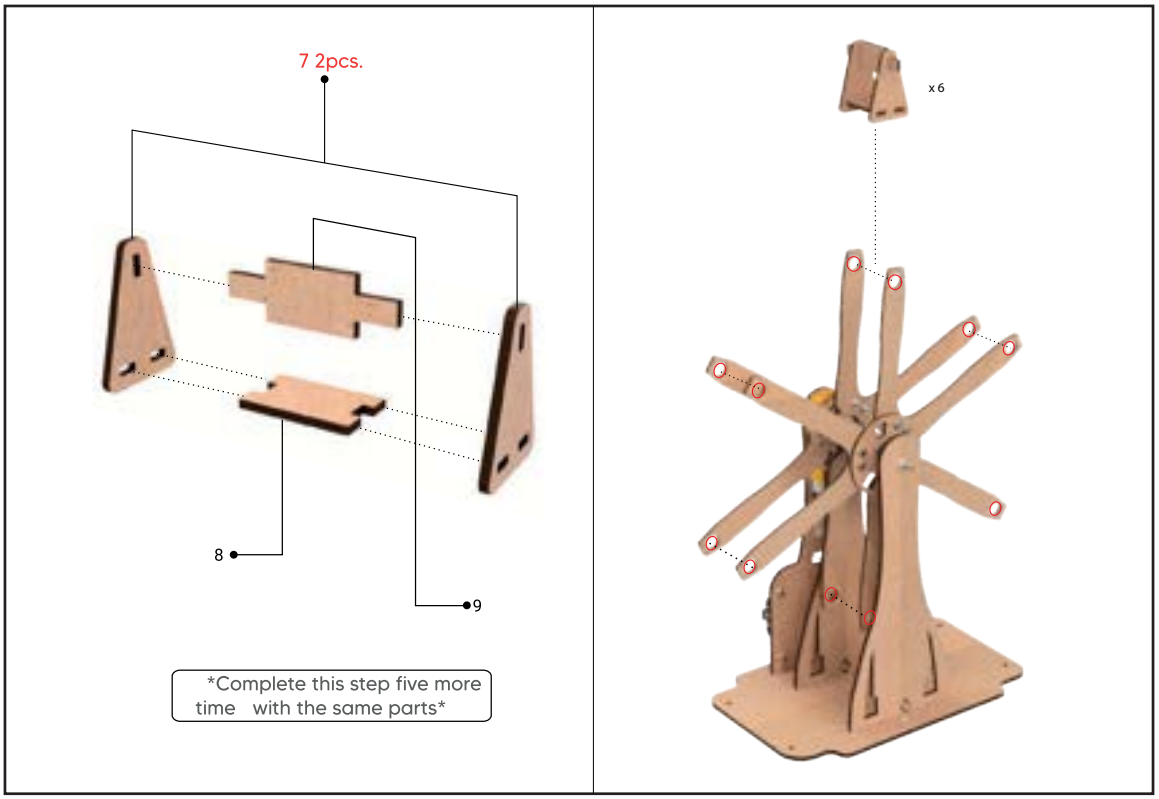


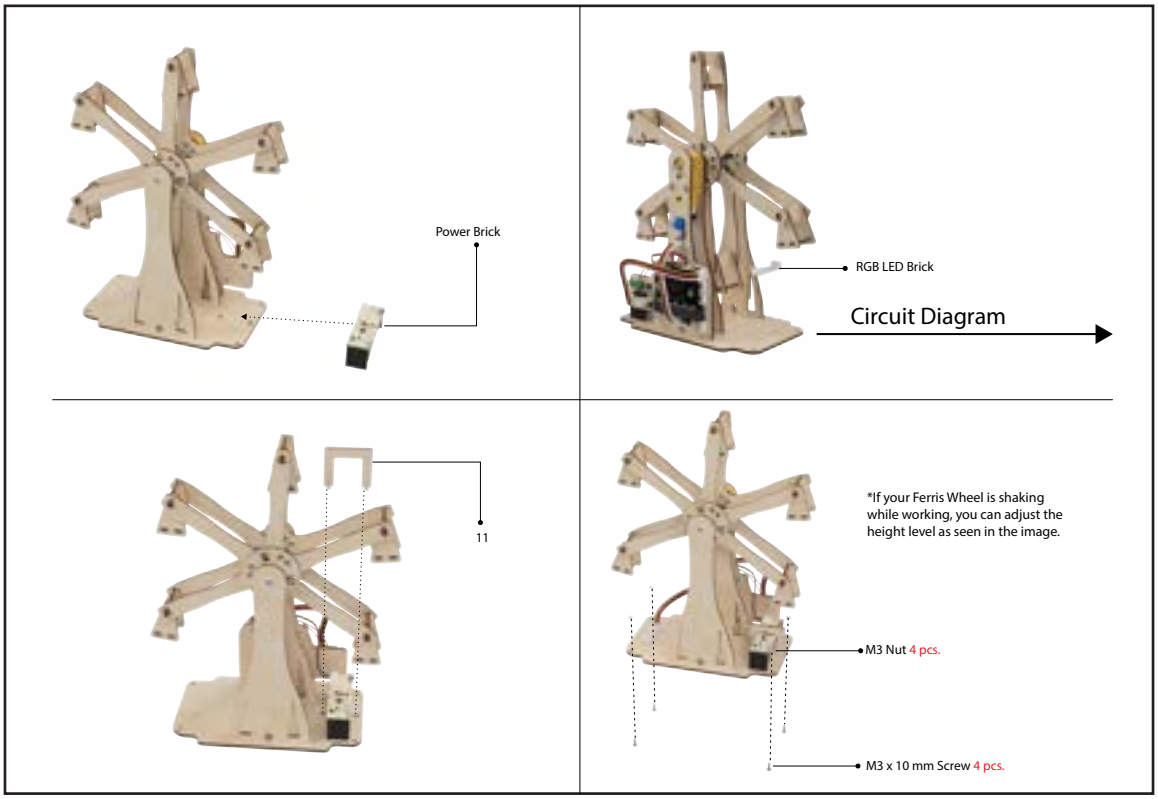



DC Gear Motor
M3 Nut 2 pcs.
M3 x 30 mm Screw 2 pcs.
6 3pcs.
M3 Nuts 6pcs.
5
M3 x 10 mm screw 6pcs.
Complete this step one more time with the same parts


M3 25 mm Screw
M3 Nut
B
M3 x 6mm F - F Spacer Nut
M3 x 10 mm Screw 3pcs.
DC Motor Hub
A
B
M3 x 40 mm F-F Spacer Nuts 3pcs.
M3 x 10 mm Screw 3pcs.

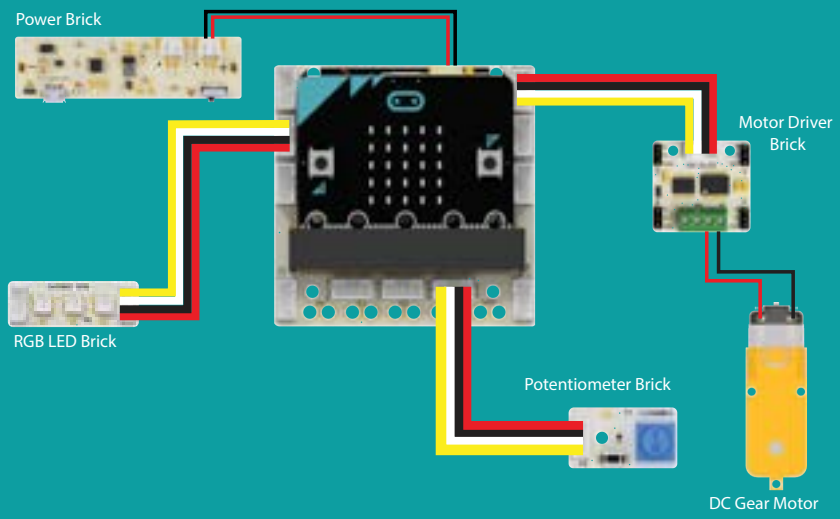

3mm Plastic Spacer
3 mm Plastic Spacer
M3 Nyloc Nut
You can also use wooden
Wrench for Nyloc Nut.
M3 Nut
M3 6 mm Female Spacer
3 mm Plasctic Spacer
M3 Nyloc Nut
3 mm Plasctic Spacer


7 2pcs.
8
9
*Complete this step five more
time with the same parts*
x 6



Setting Up The Circuit

et to know the circuit elements of Mini Tank that we completed the setup and make the circuit setup coBricks modules.

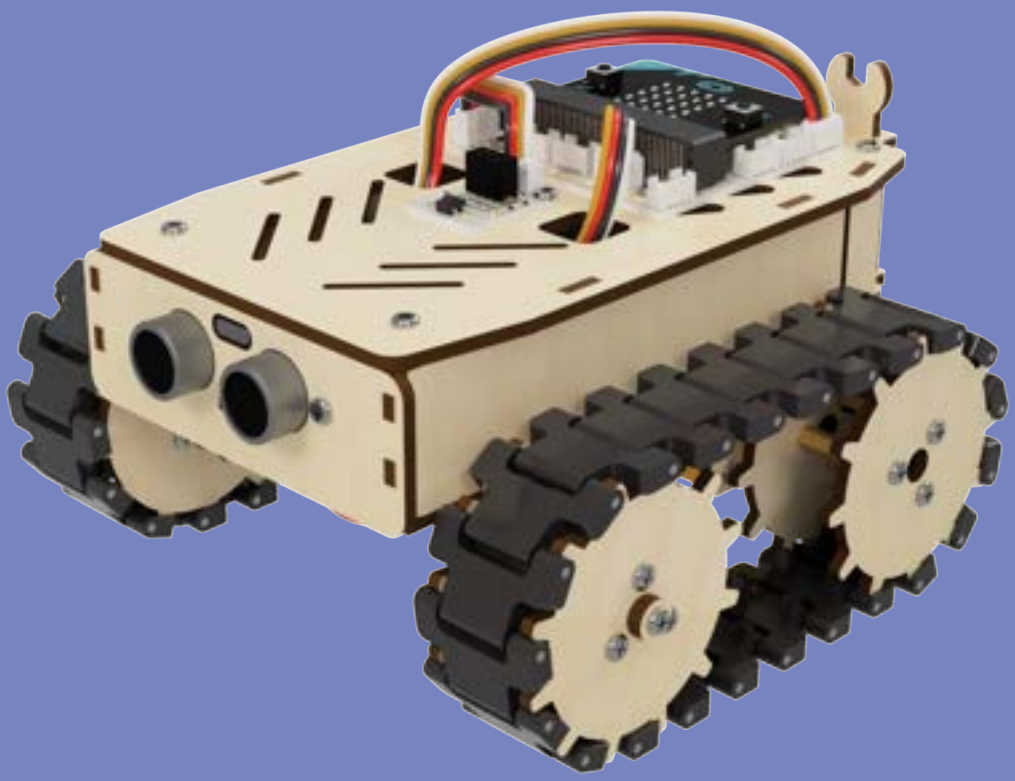


MicroBlocks Code of The Project:

Ferris-Wheel

```
#Ferris Wheel Project
from microbit import *
from picobricks import *

# Pin Initialization
Pot_Pin = pin1

# Function Initialization
motor = motordriver()

display.show(Image.HAPPY)

while True:
    pot = Pot_Pin.read_analog()
    speed=round(round( pot - 0 ) * ( 255 - 0 ) / ( 1023 - 0 ) + 0)
    motor.dc(1,speed,1)
```

Mars Explorer



Mars Explorer Project

Tanks, with their tracked structures, are vehicles that can easily move on rough terrains. Tracks consist of multiple sequential wheels or rollers surrounded by a belt.

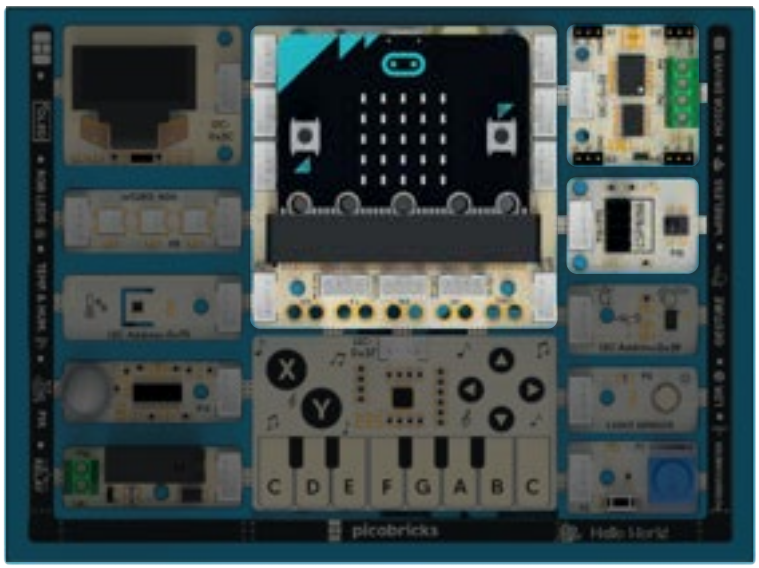
The PicoBricks Mars Explorer Car is a wooden project kit that utilizes two DC motors and a tracked platform. This robot car, controllable remotely with a remote controller thanks to the IR receiver, can decide on its movements by detecting surrounding objects through the front of its distance sensor.

Project Details:

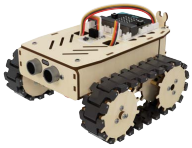
In this project, we will control two DC motors that connected to motor driver by using IR receiver on the PicoBricks wireless module with the remote controller. The robot car moves in the desired direction thanks to the DC motors. Additionally, if the HC-SR04 distance sensor on the robot car kit detects an object within 15 cm, the robot car will stop.

Connection Diagram:

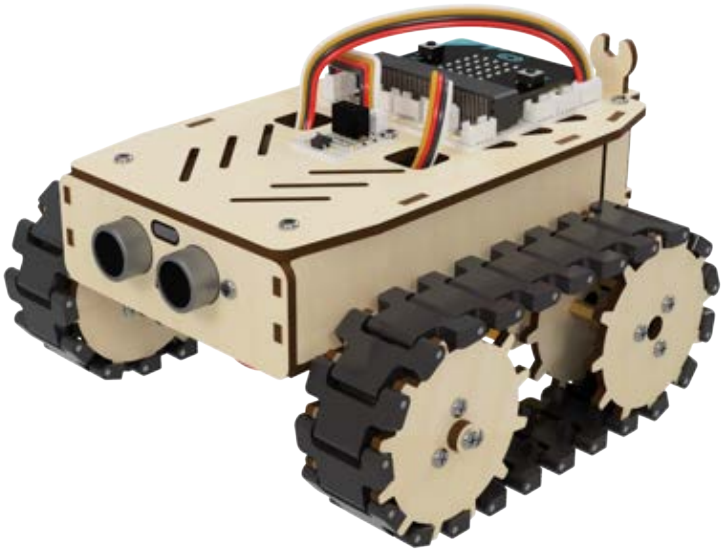
You can prepare this project by breaking PicoBricks modules at proper points.

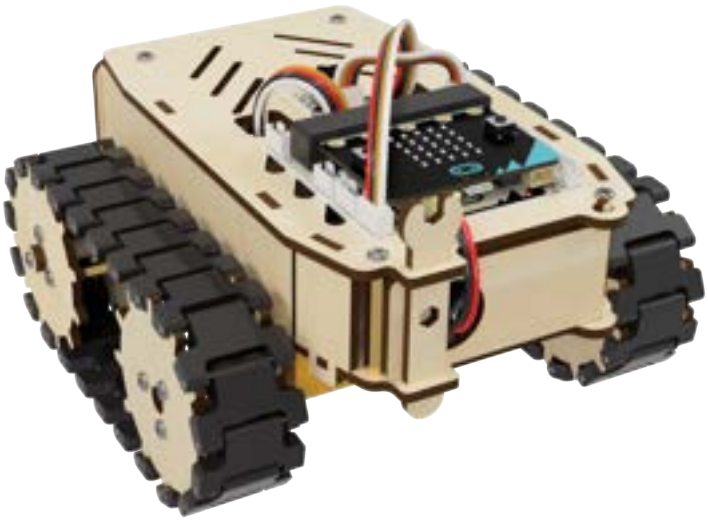


x2 DC Motor

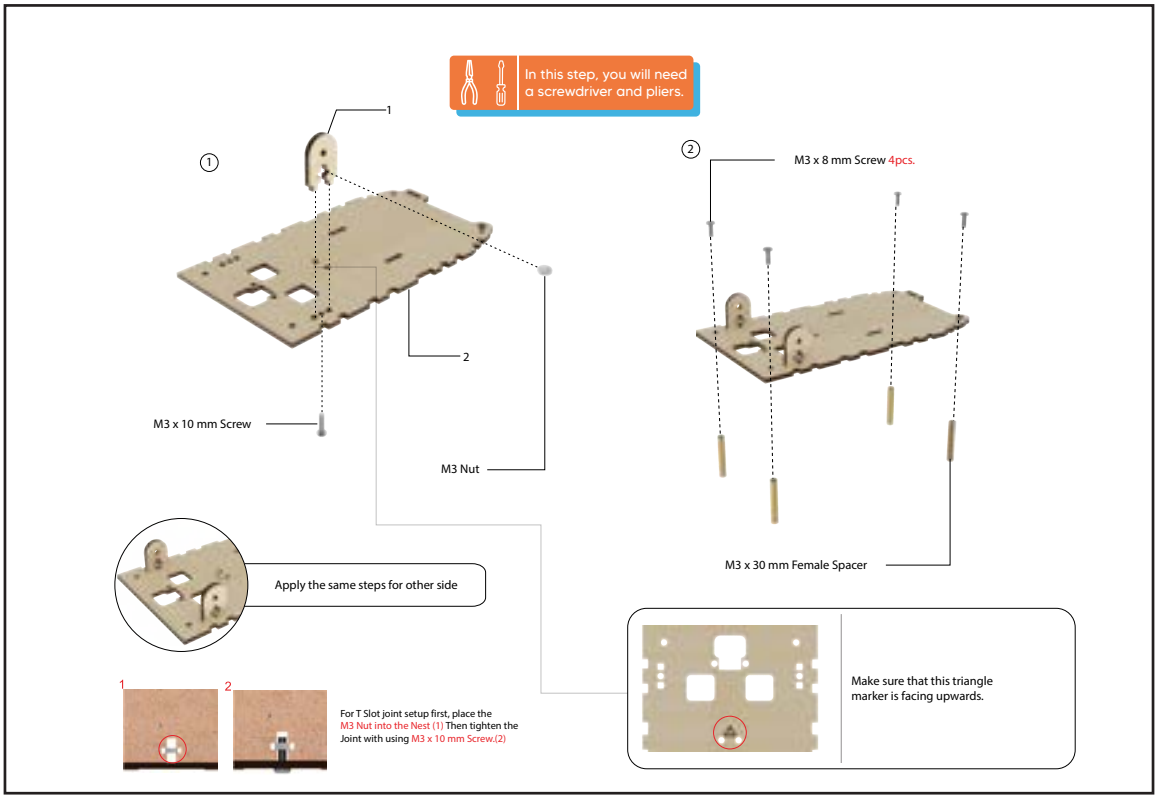


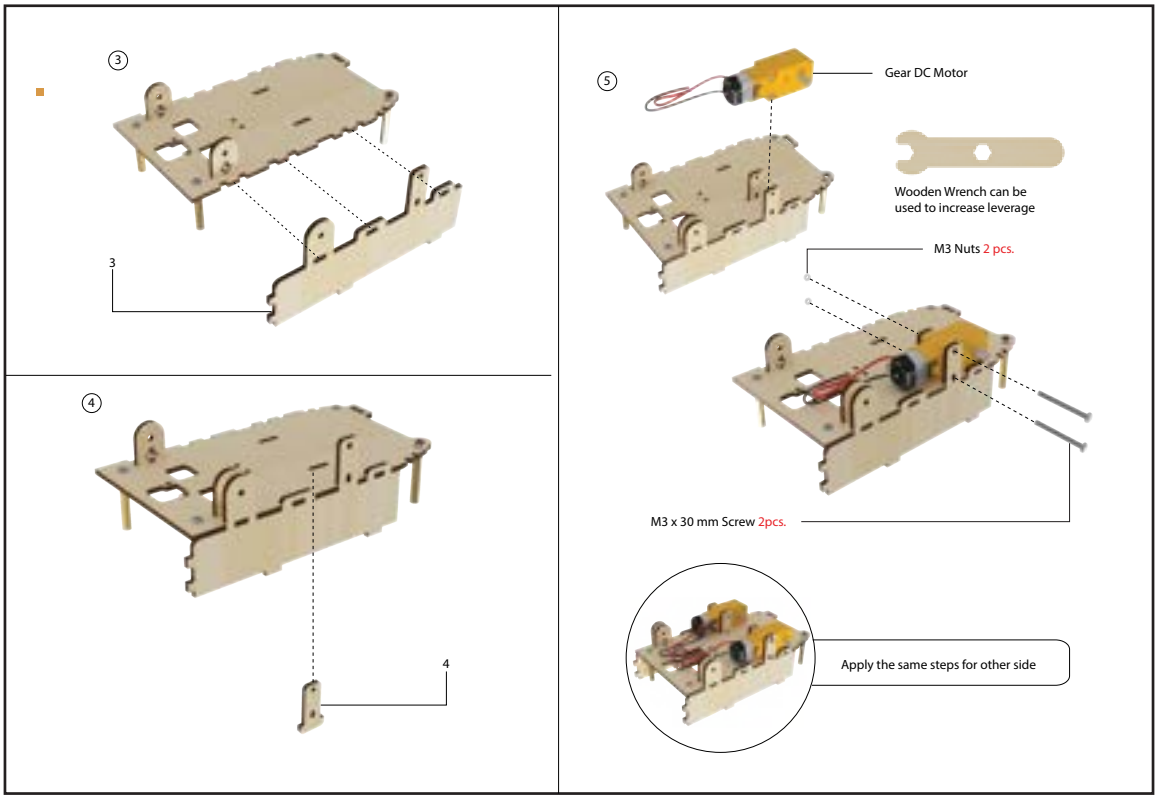
Project Images:

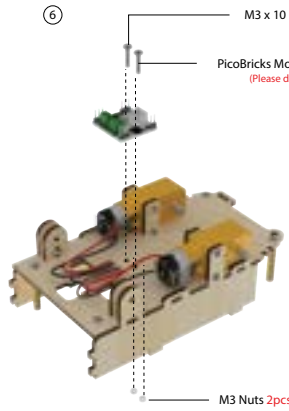


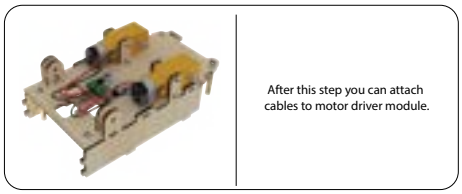


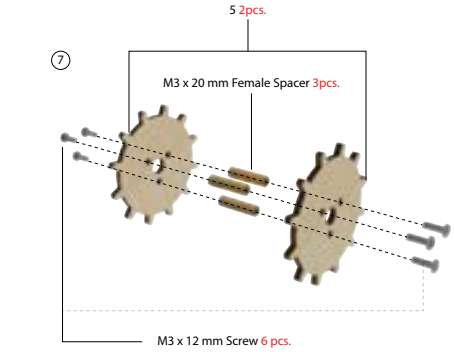
Setup Steps of The Project:

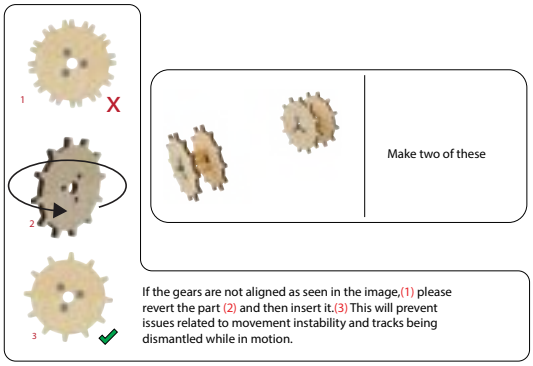


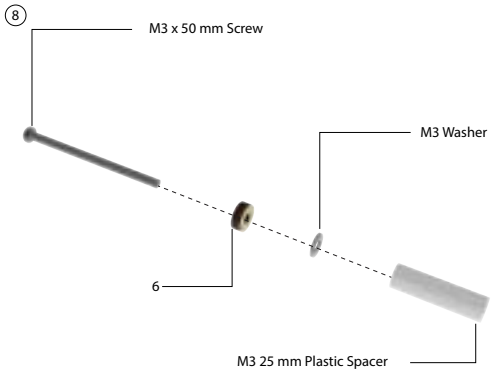


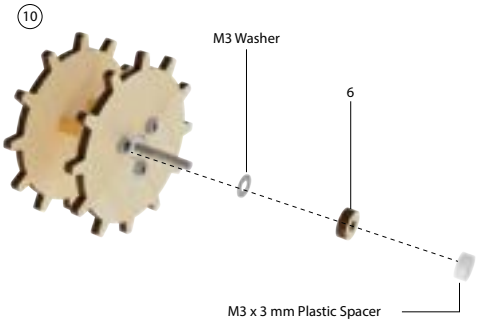

6
M3 x 10 mm Screw 2 pcs.
PicoBricks Motor Driver Module
(Please do not over tight)
M3 Nuts 2pcs.

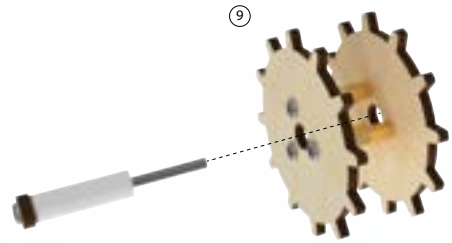

After this step you can attach cables to motor driver module.

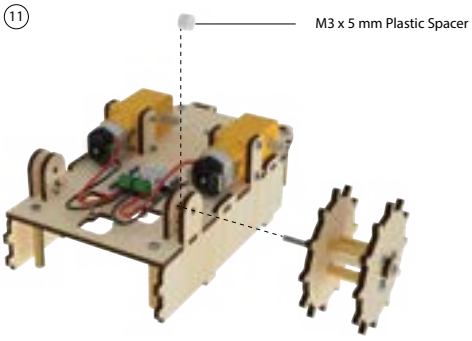

5 2pcs.
7
M3 x 20 mm Female Spacer 3pcs.
M3 x 12 mm Screw 6 pcs.


1
X
2
3
Make two of these
If the gears are not aligned as seen in the image,(1) please revert the part (2) and then insert it.(3) This will prevent issues related to movement instability and tracks being dismantled while in motion.


8
M3 x 50 mm Screw
M3 Washer
6
M3 25 mm Plastic Spacer

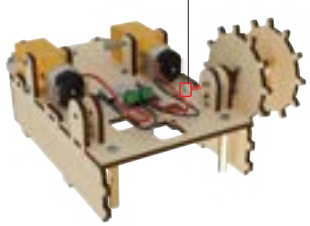

10
M3 Washer
6
M3 x 3 mm Plastic Spacer


9

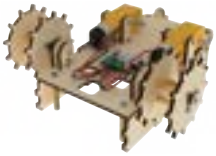

11
M3 x 5 mm Plastic Spacer

12

M3 Nyloc Nut



Wooden Wrench can be used for M3 Nyloc Nut but, pay attention that its not too tight otherwise, wheel won't be turning.

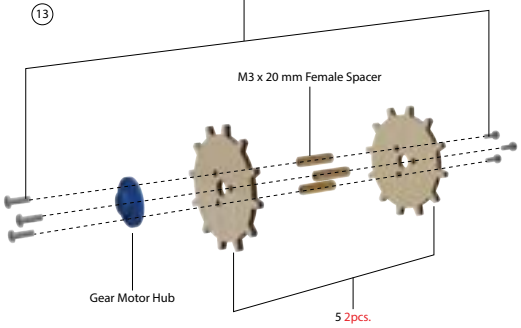


Apply the same steps to other side

13

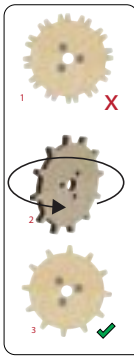
M3 x 12 mm Screw 6pcs.

M3 x 20 mm Female Spacer

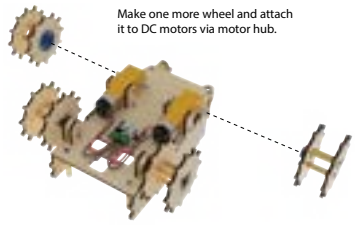


Gear Motor Hub

5 2pcs.



Make one more wheel and attach it to DC motors via motor hub.



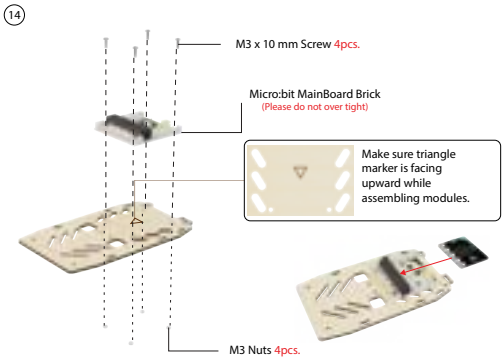
If the gears are not aligned as seen in the image,(1) please revert the part (2) and then insert it.(3) This will prevent issues related to movement instability and tracks being dismantled while in motion.

14

M3 x 10 mm Screw 4pcs.

Micro:bit MainBoard Brick
(Please do not over tight)

Make sure triangle marker is facing upward while assembling modules.

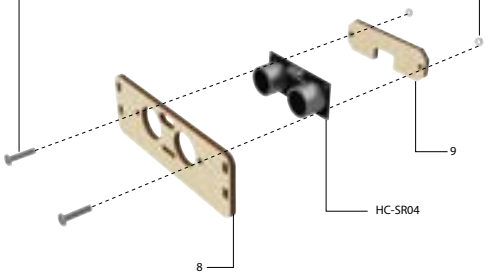


M3 Nuts 4pcs.

16

M3 x 12 mm Screw 2pcs.

M3 Nut 2pcs.



9

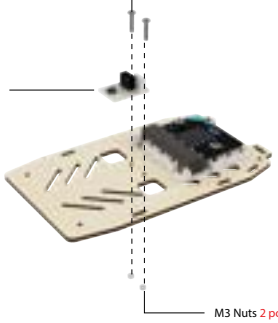
HC-SR04

8

15

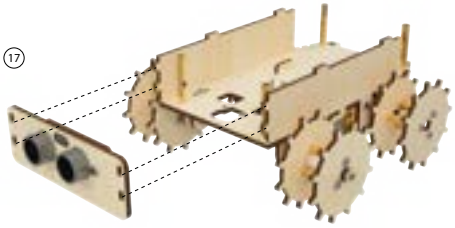
M3 x 10 mm Screw 2 pcs.

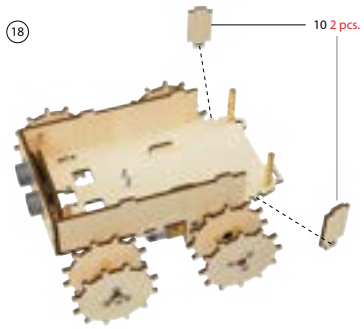
WIFI & IR Brick



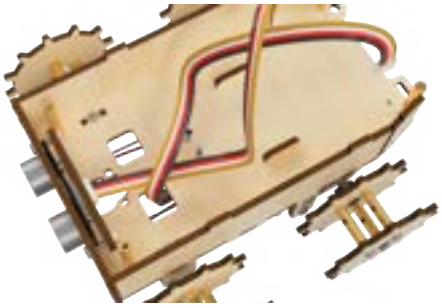
M3 Nuts 2 pcs.

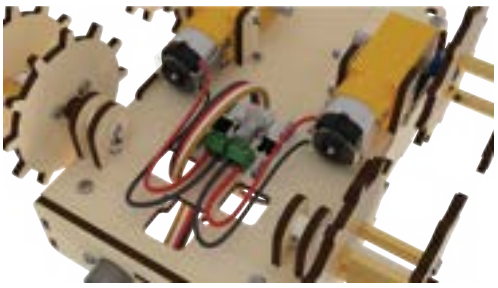
17





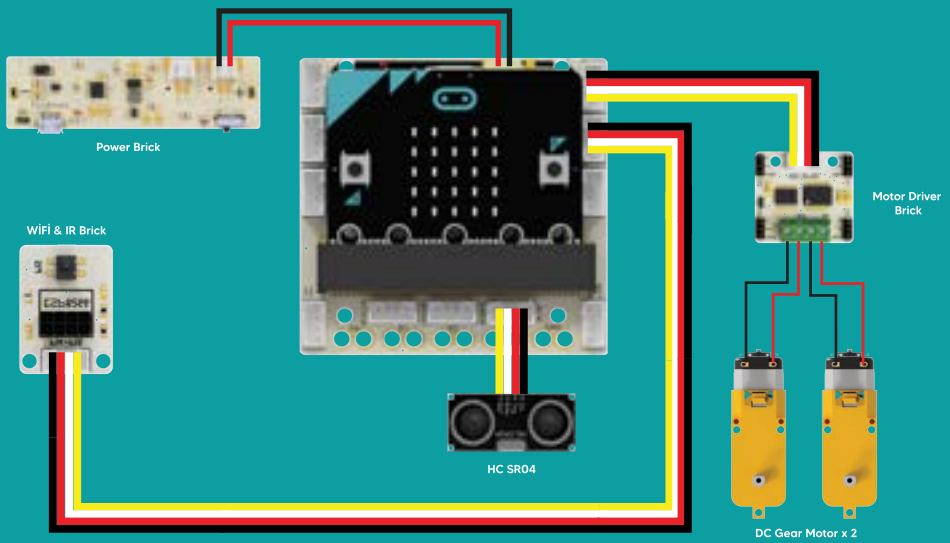
As we approach the process of assembling the body parts together, please ensure that all cables are easily accessible for attachment to the mainboard you can use cable areas on the parts as seen in the image.



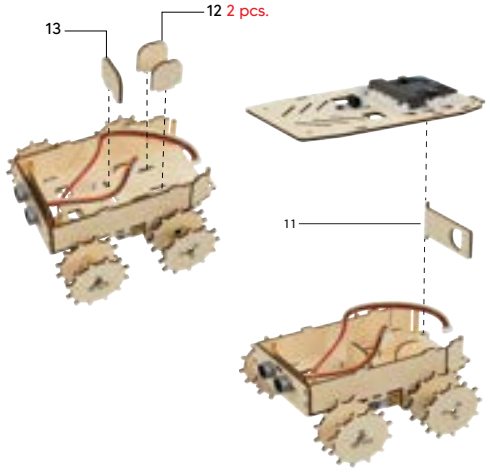


Setting Up The Circuit

Let's get to know the circuit elements of Mini Tank that we completed the setup and make the circuit setup with PicoBricks modules.

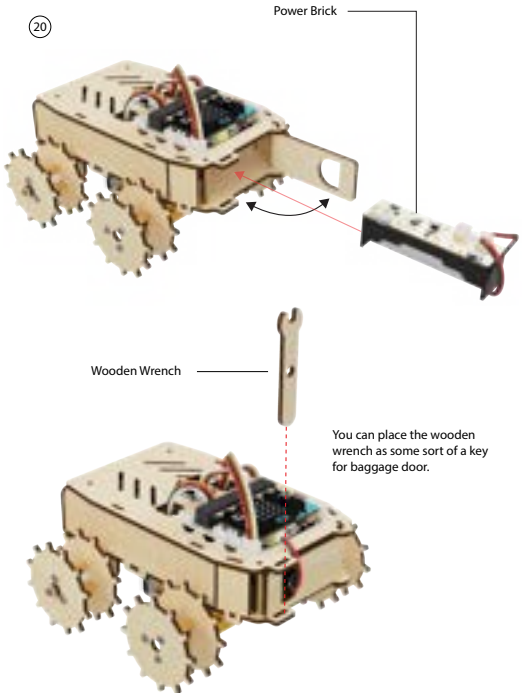


19

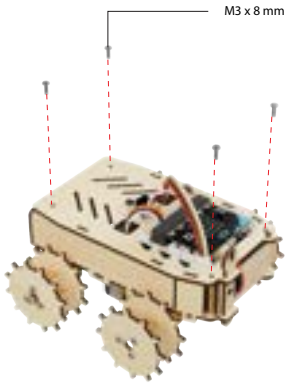


Make sure back side of the upper and body parts are aligned like this in the image. this will prevent dissambling related parts. because hinge holes are needed to be aligned for part 11. to work properly as a bagage door.

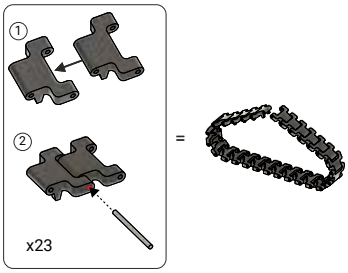
20



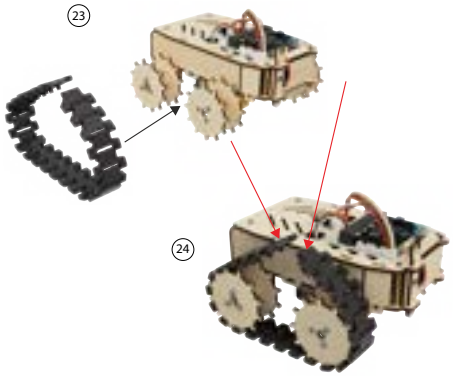
21



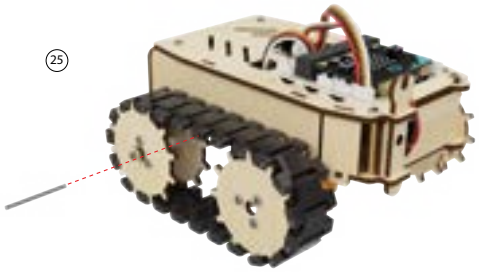
22

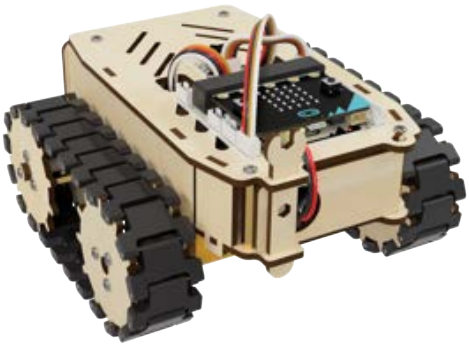


23

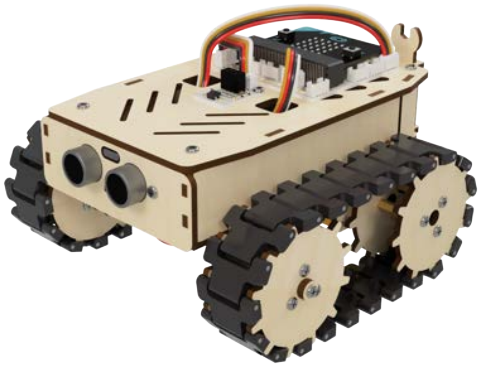


25





Attach the tracks for the other side by following previous assembly steps.



Assembly is finished and you can move on to coding steps.

MicroBlocks Code of The Project:

Mars-Explorer

```
#Mars Explorer Project
from microbit import *
from picobricks import *

# Function Initialization
motor = motordriver()
pin15.set_pull(pin15.PULL_UP)
ir = IRM()

motor.dc(1,0,1)
motor.dc(2,0,1)

def forward():
    motor.dc(1,255,1)
    motor.dc(2,255,1)

def backward():
    motor.dc(1,255,0)
    motor.dc(2,255,0)

def left():
    motor.dc(1,0,1)
    motor.dc(2,255,1)

def right():
    motor.dc(1,255,1)
    motor.dc(2,0,1)

def stop():
    motor.dc(1,0,1)
    motor.dc(2,0,1)

def left_image():
    display.show(Image('00900:'
                       '09000:'
                       '99999:'
                       '09000:'
                       '00900'))

def right_image():
    display.show(Image('00900:'
                       '00090:'
                       '99999:'
                       '00090:'
                       '00900'))

def up_image():
    display.show(Image('00900:'
                       '09990:'
                       '90909:'
                       '00900:'
                       '00900'))

def down_image():
    display.show(Image('09990:'
                       '00900:'
                       '00900:'
                       '09990:'
                       '00900'))

while True:
    distance = measure_distance()
    #print(distance)
    key=ir.get(pin15)
    if(key!=-1):
        print(key)
        if key ==  24:
            if distance<=15:
                stop()
            else:
                forward()
                down_image()
        elif key == 90:
            right()
            right_image()
        elif key == 8:
            left()
            left_image()
        elif key == 82:
            backward()
            up_image()
        #sleep(100)
        else:
            stop()
            print(key)
            display.show(Image.HAPPY)
```

Trash Tech

Trash Tech Project

The Trash Tech Kit is an educational robotics programming kit designed to gain environmental awareness in children.

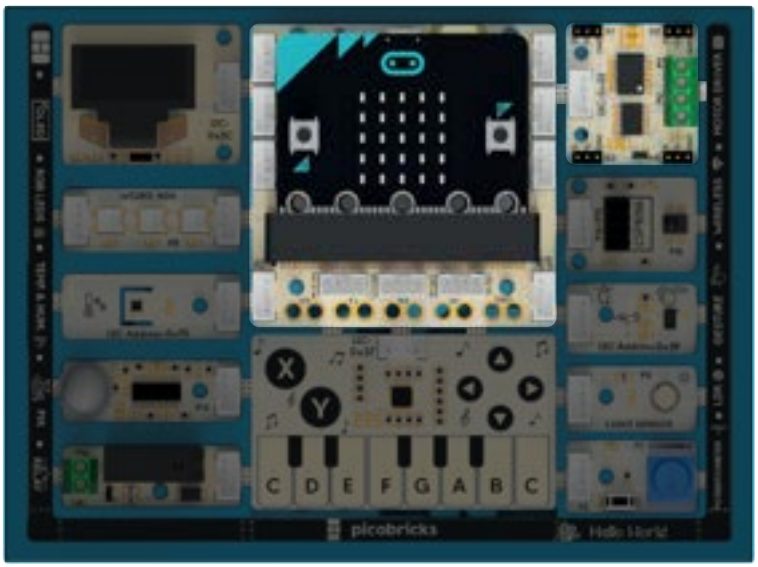
The Trash Tech is a fun kit that allows you to assemble the wooden pieces, sensors, and PicoBricks modules included in the set as specified in the installation guide. The goal of the project is to create an electronic trash bin that opens its lid by detecting objects using the HC-SR04 distance sensor located at its front.

Project Details:

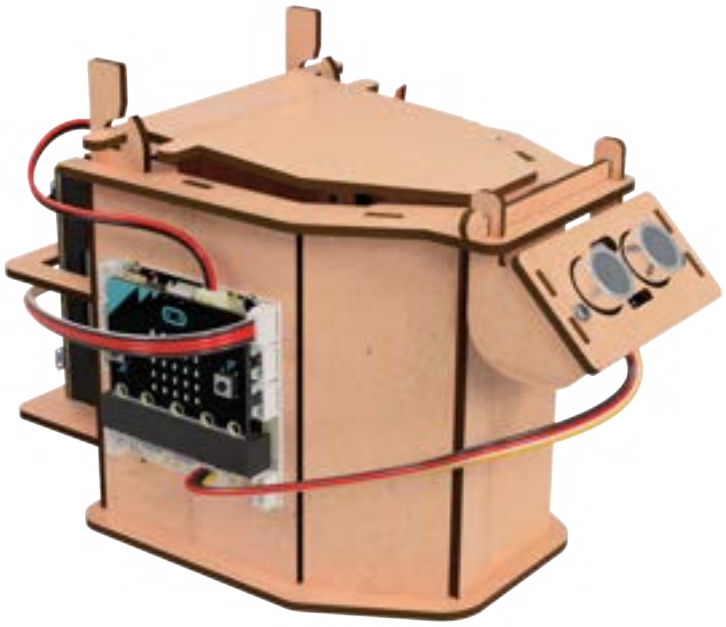
In this project, we detect the distance of our hand using the HCSR04 distance sensor and move the servo motor connected to the motor driver to the desired angle. This way, the lid of the trash bin opens.

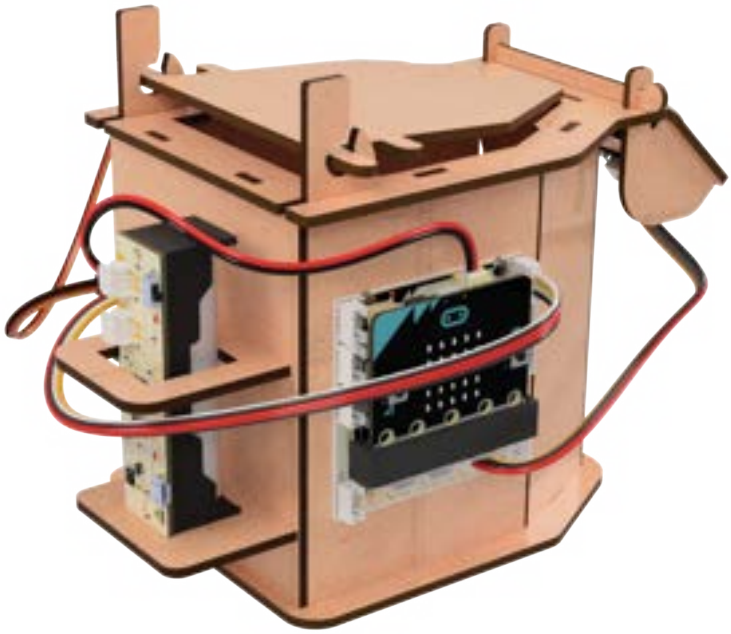
Connection Diagram:

You can prepare this project by breaking the PicoBricks modules at suitable points.



Project Images:

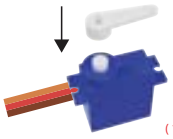




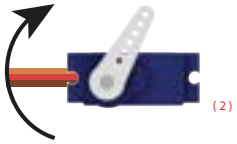
Setup Steps of The Project:

Servo Motor Calibration

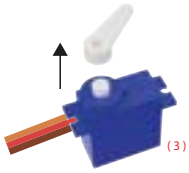
Before starting the assembly, you have to manually calibrate the angles of the servo motors. Otherwise, Servo Motors won't be working properly.



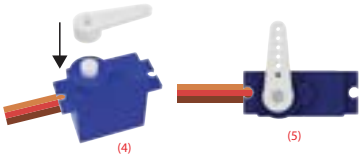
Attach the servo horn to the servo motor (1)



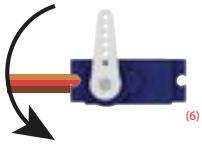
Then slowly turn the servo horn clockwise until it stops. It is not a problem if the servo horn is not the same as the angle shown in the image above. The important thing here is that you have hit the last angle of the servo. (2)



Remove the servo horn from the servo motor (3)



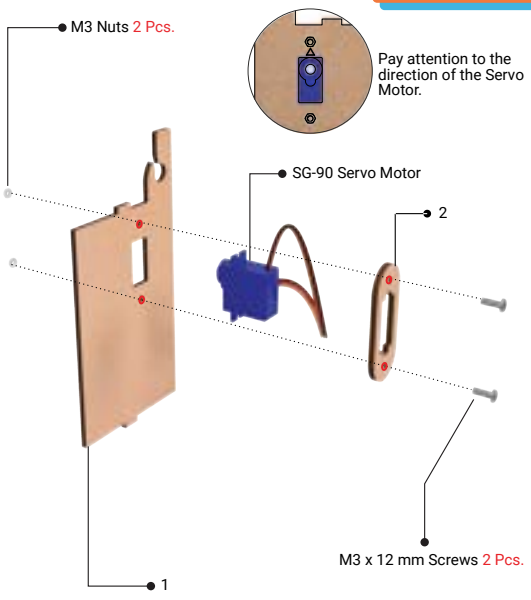
Reattach (4) and reposition the servo horn perpendicular to the servo motor as shown. (5)

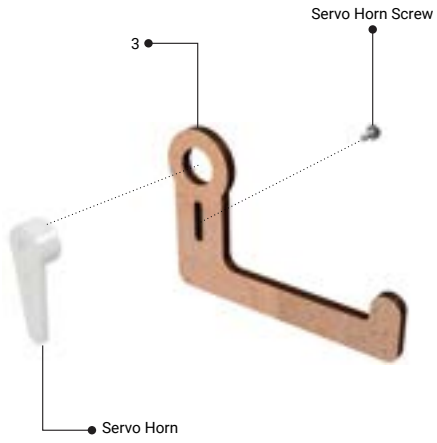


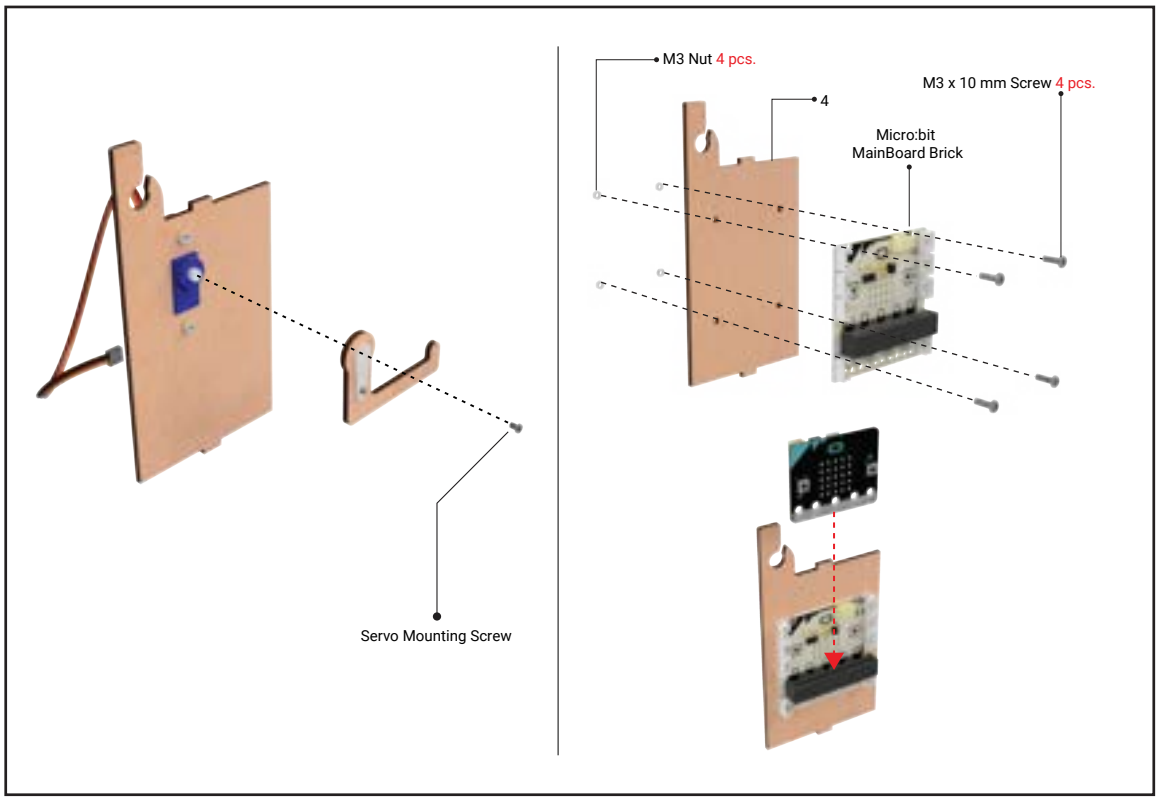
Slowly turn the servo horn counterclockwise (6) until it is parallel with the servo motor, as seen in the image.(7)

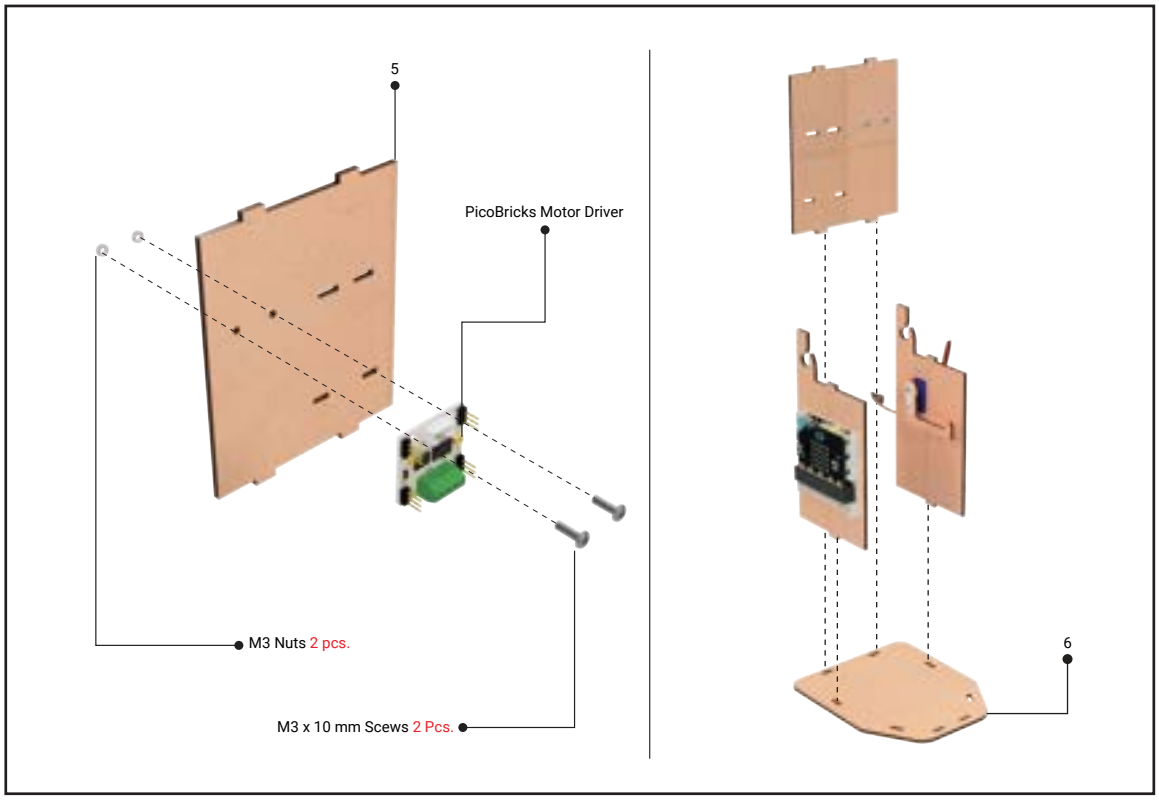
When this step is finished, it means that the servo motor is in the center position. It is important that you apply this process to other servo motors in the set. Afer processing the other motors, remove the servo horn and set aside for assembly.

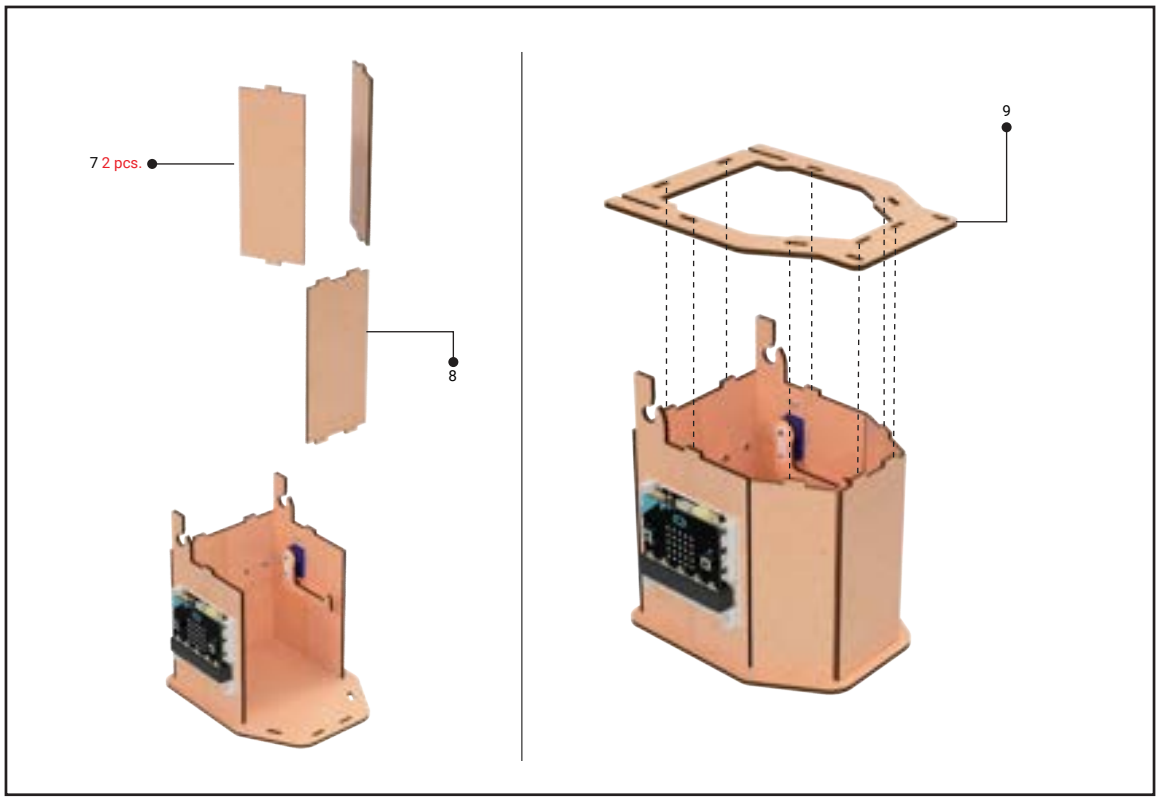
In this step, you will need a screwdriver and pliers.

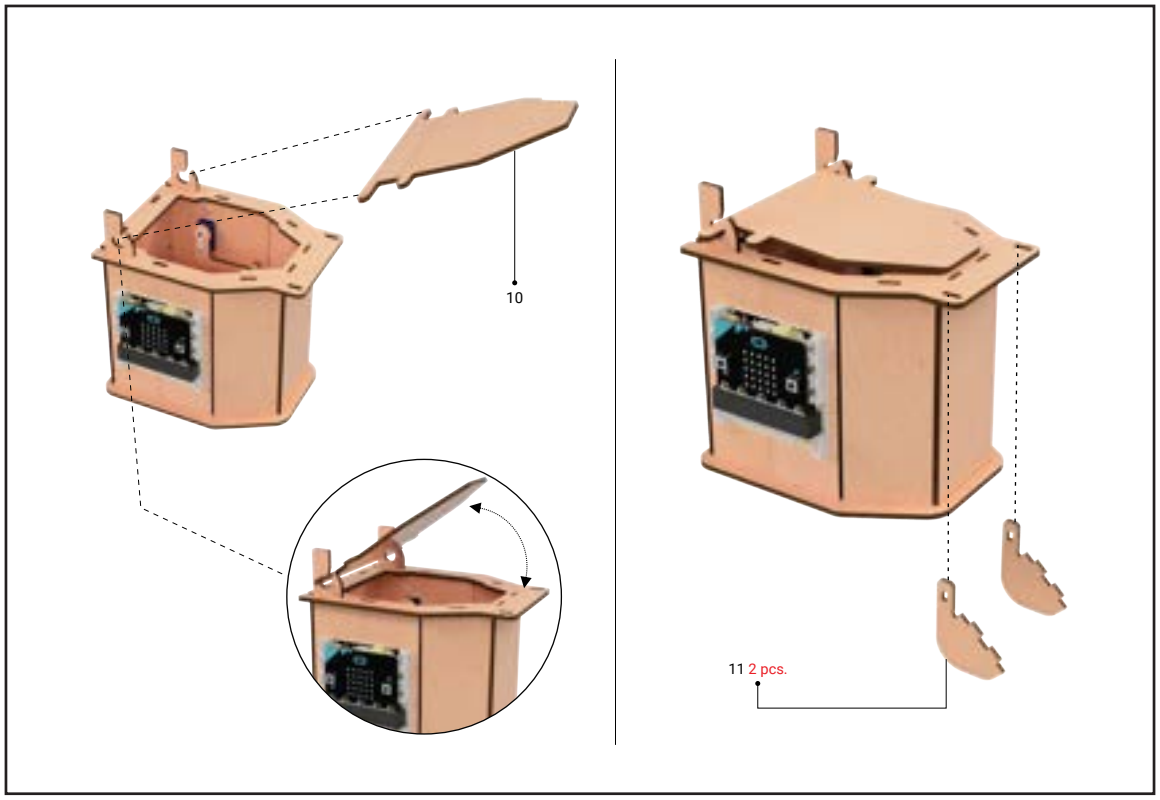


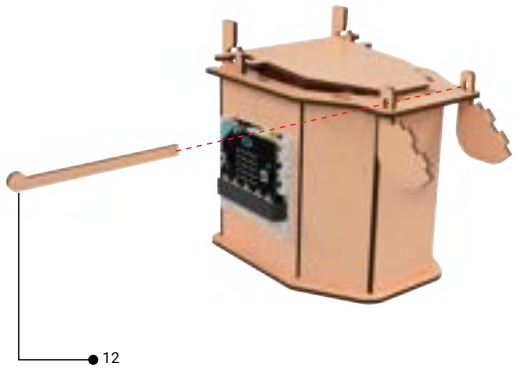


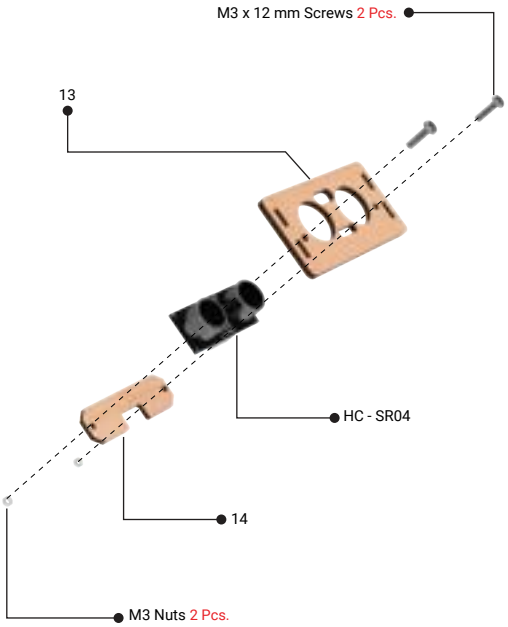

M3 Nut 4 pcs.
M3 x 10 mm Screw 4 pcs.
4
Micro:bit
MainBoard Brick
Servo Mounting Screw

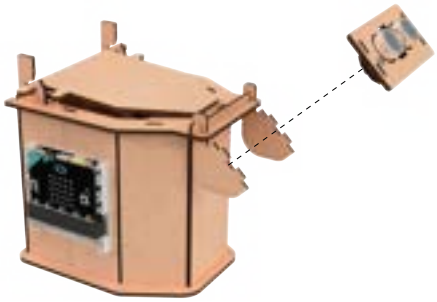

5
PicoBricks Motor Driver
M3 Nuts 2 pcs.
M3 x 10 mm Scews 2 Pcs.
6

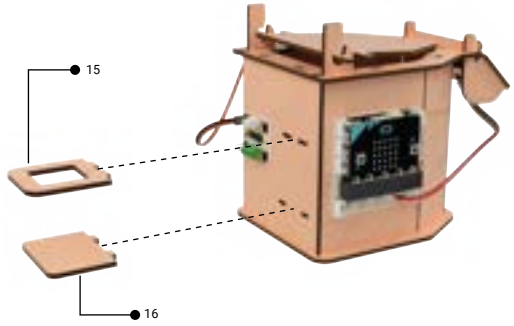

7 2 pcs.
8
9

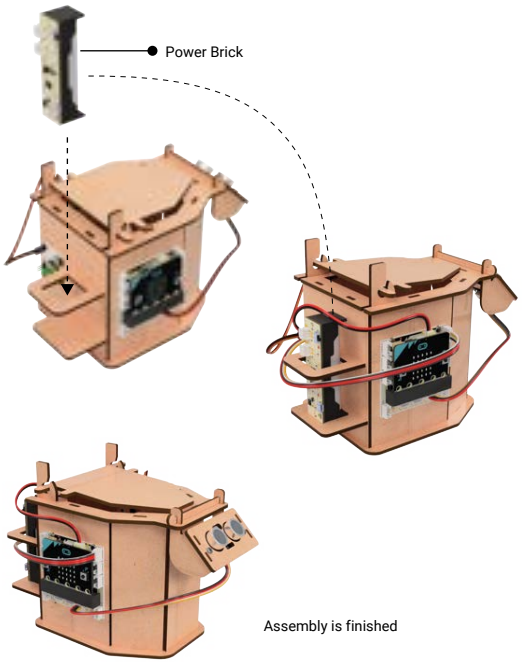

10
11 2 pcs.


12


M3 x 12 mm Screws 2 Pcs.
13
HC - SR04
14
M3 Nuts 2 Pcs.

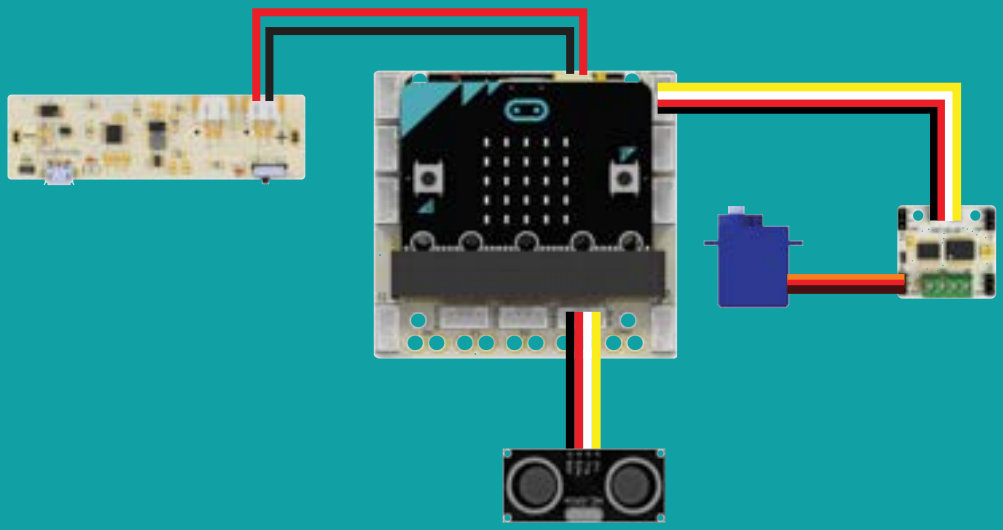



15
16


Power Brick
Assembly is finished

Unplugged 4: Setup of Circut

Let's get to know the circuit elements of the smart trash bin that we have completed and set up the circuit with PicoBricks modules.



MicroBlocks Code of The Project:

Trash-Tech

```
#Trash Tech Project
from microbit import *
from picobricks import *

# Function Initialization
motor = motordriver()

display.show(Image.HAPPY)
while True:
    distance = measure_distance()
    if distance<9 and distance>1:
        display.show(Image.YES)
        motor.servo(1,90)
        sleep(2000)
        motor.servo(1,180)
        sleep(200)
    else:
        motor.servo(1,180)
        display.show(Image.HAPPY)
    sleep(200)

```

Money Box

Money Box Project

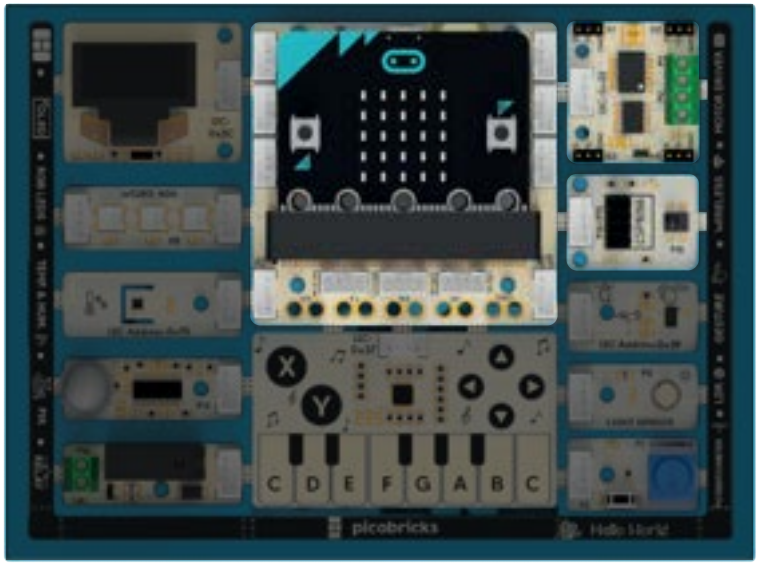
PicoBricks Money Box can detect objects placed in its receptacle through distance sensor in the front of it and automatically lifts its receptacle to take in these objects.

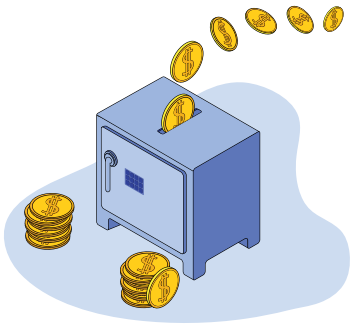
Project Details:

In this project, when the distance sensor detects the object placed in the receptacle , the servo motor connected to the motor driver is adjusted to the angle specified in the code, and then the object inside the receptacle is dropped into the Money Box.

Connection Diagram:

You can assemble this project by breaking apart the PicoBricks modules at the proper points.



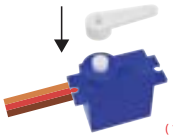


Project Images:

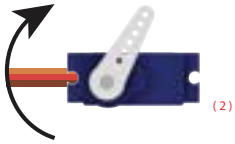
Setup Steps of The Project:

Servo Motor Calibration

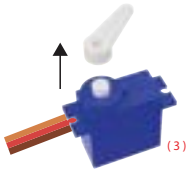
Before starting the assembly, you have to manually calibrate the angles of the servo motors. Otherwise, Servo Motors won't be working properly.



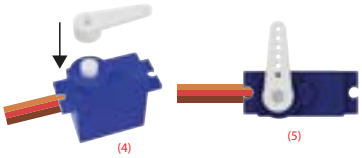
Attach the servo horn to the servo motor (1)



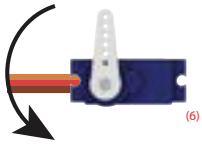
Then slowly turn the servo horn clockwise until it stops. It is not a problem if the servo horn is not the same as the angle shown in the image above. The important thing here is that you have hit the last angle of the servo. (2)



Remove the servo horn from the servo motor (3)

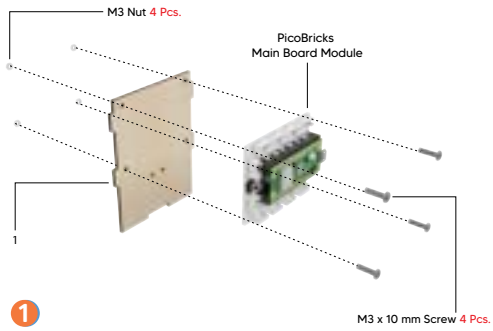


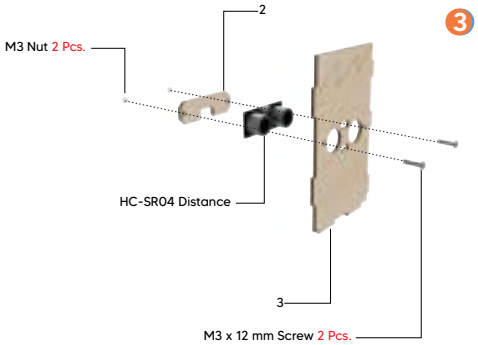
Reattach (4) and reposition the servo horn perpendicular to the servo motor as shown. (5)

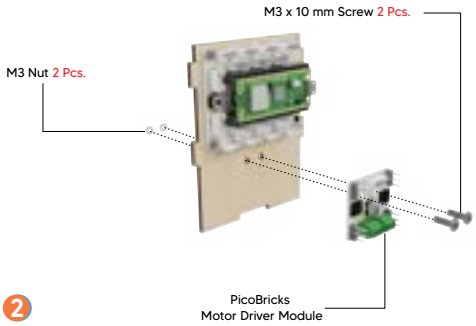


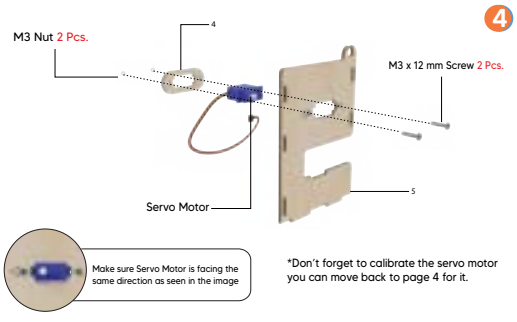
Slowly turn the servo horn counterclockwise (6) until it is parallel with the servo motor, as seen in the image.(7)

When this step is finished, it means that the servo motor is in the center position. It is important that you apply this process to other servo motors in the set. Afer processing the other motors, remove the servo horn and set aside for assembly.

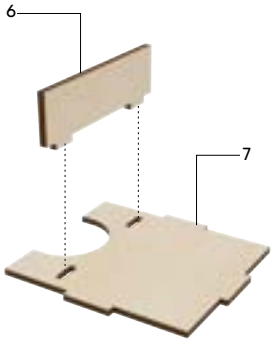




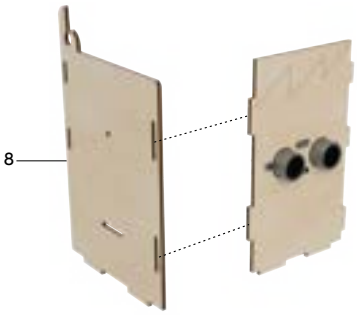




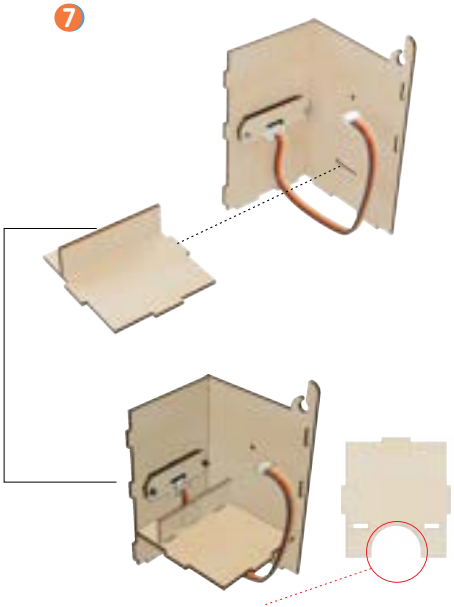
5



6



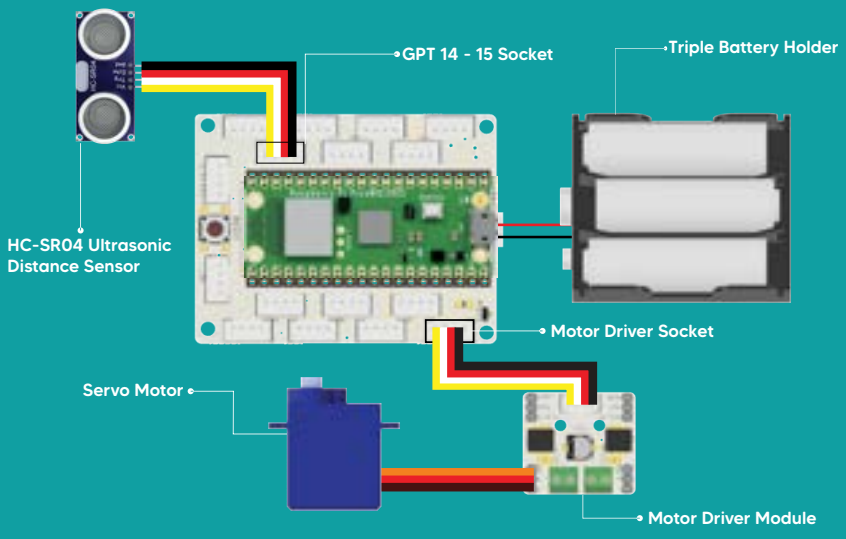
7



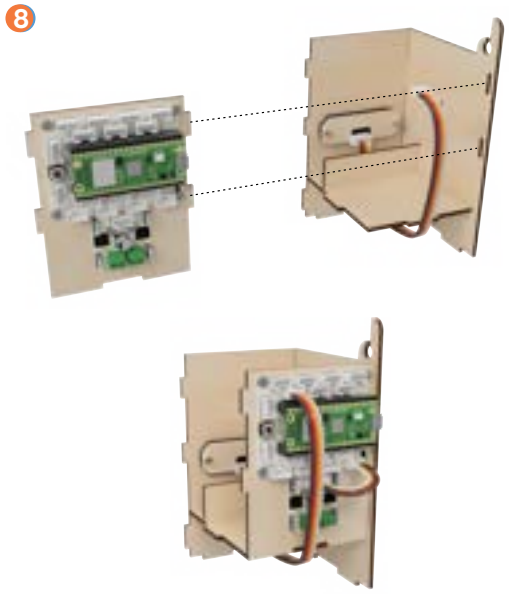
You can thread the cable through the open-ended side of the component.

Setting Up The Circuit

Let's get to know the circuit elements of Money Box that we completed the setup and make the circuit setup with PicoBricks modules.

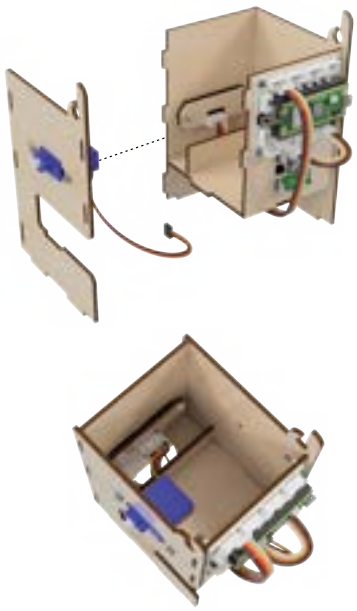


8



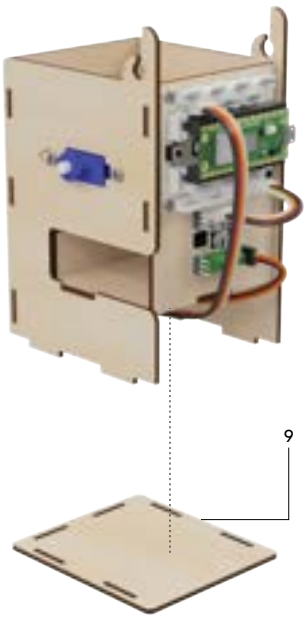
You can start making the cable connections after this step

9

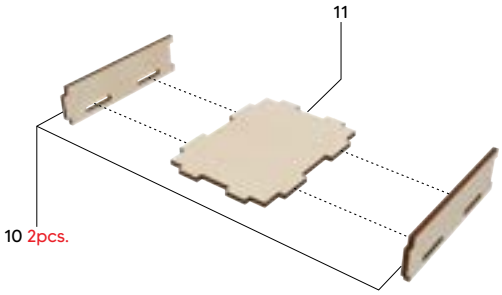


You can thread cables through the same component.

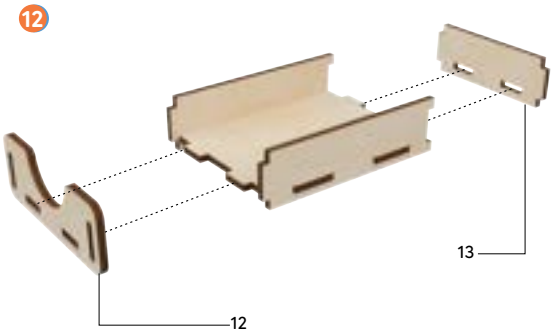
10



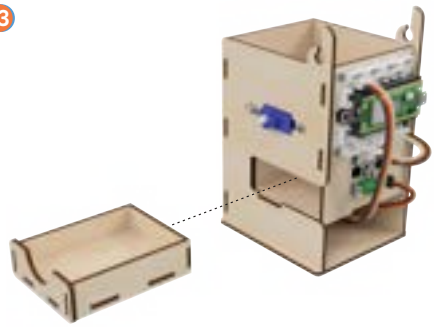
11



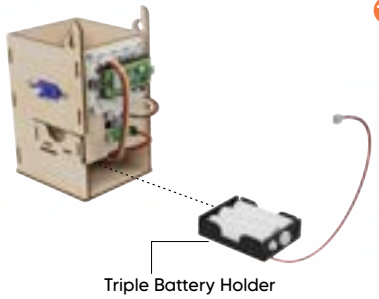
12



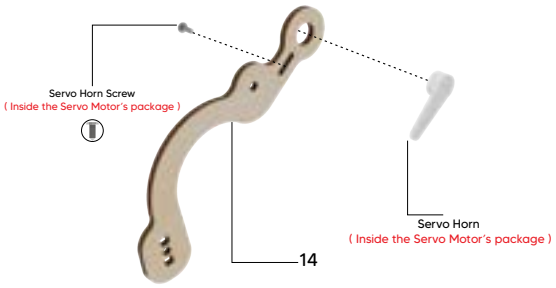
13



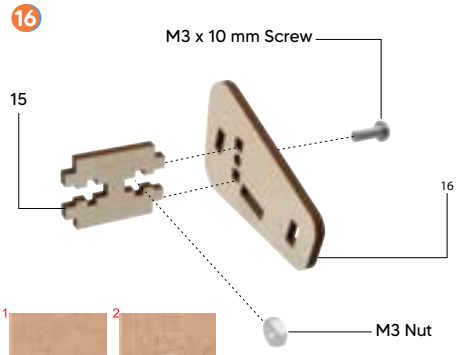
14

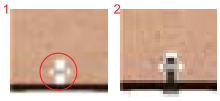


15



16

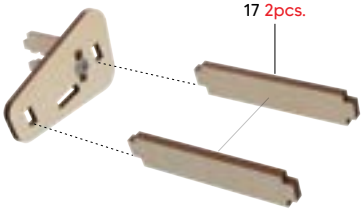




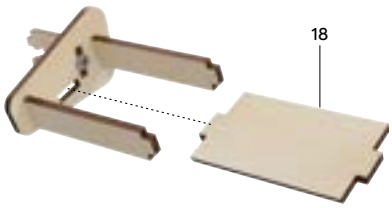
For T Slot joint setup first, place the M3 Nut into the Nest (1) Then tighten the Joint with using M3 x 10 mm Screw.(2)

Repeat this step one more time.

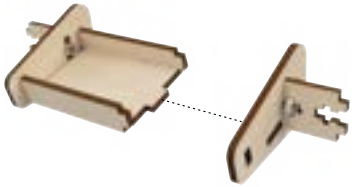
17



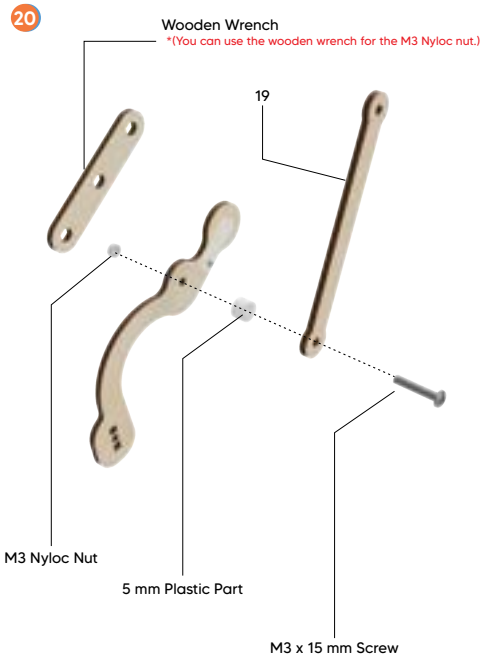
18



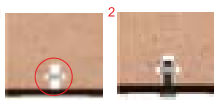
19



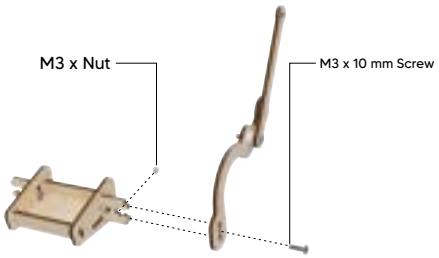
20



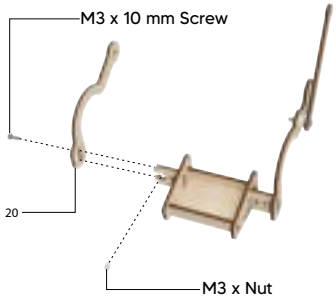
21



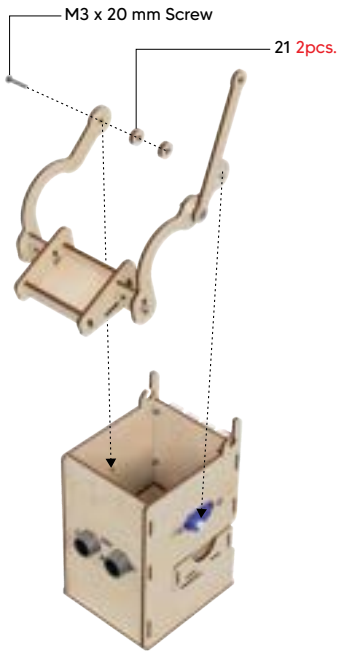
For T Slot joint setup first, place the M3 Nut into the Nest (1) Then tighten the Joint with using M3 x 10 mm Screw.(2)



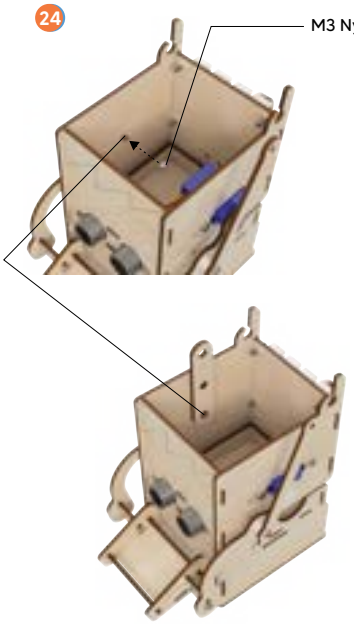
22



23



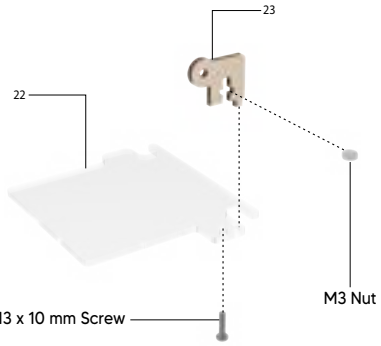
24



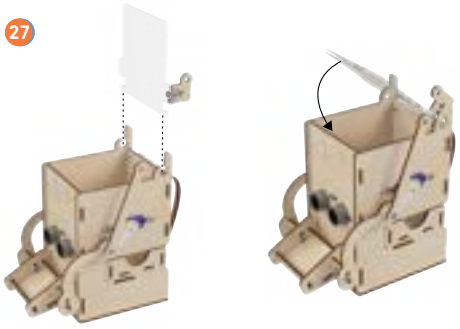
You can use the wooden wrench for the M3 Nyloc nut.

25

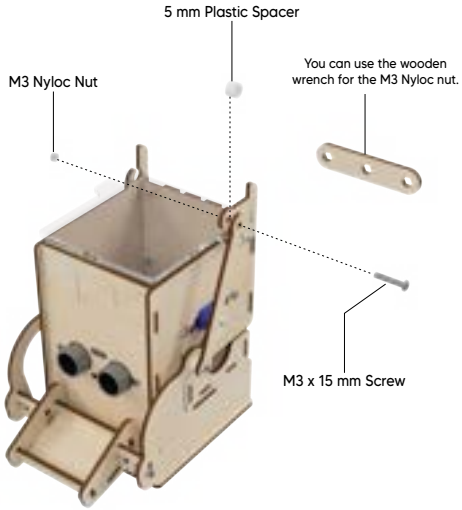
26



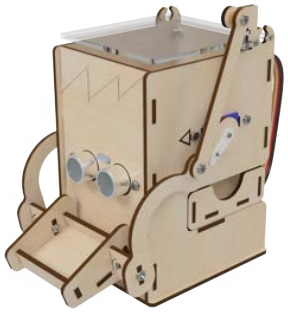
27



28



29



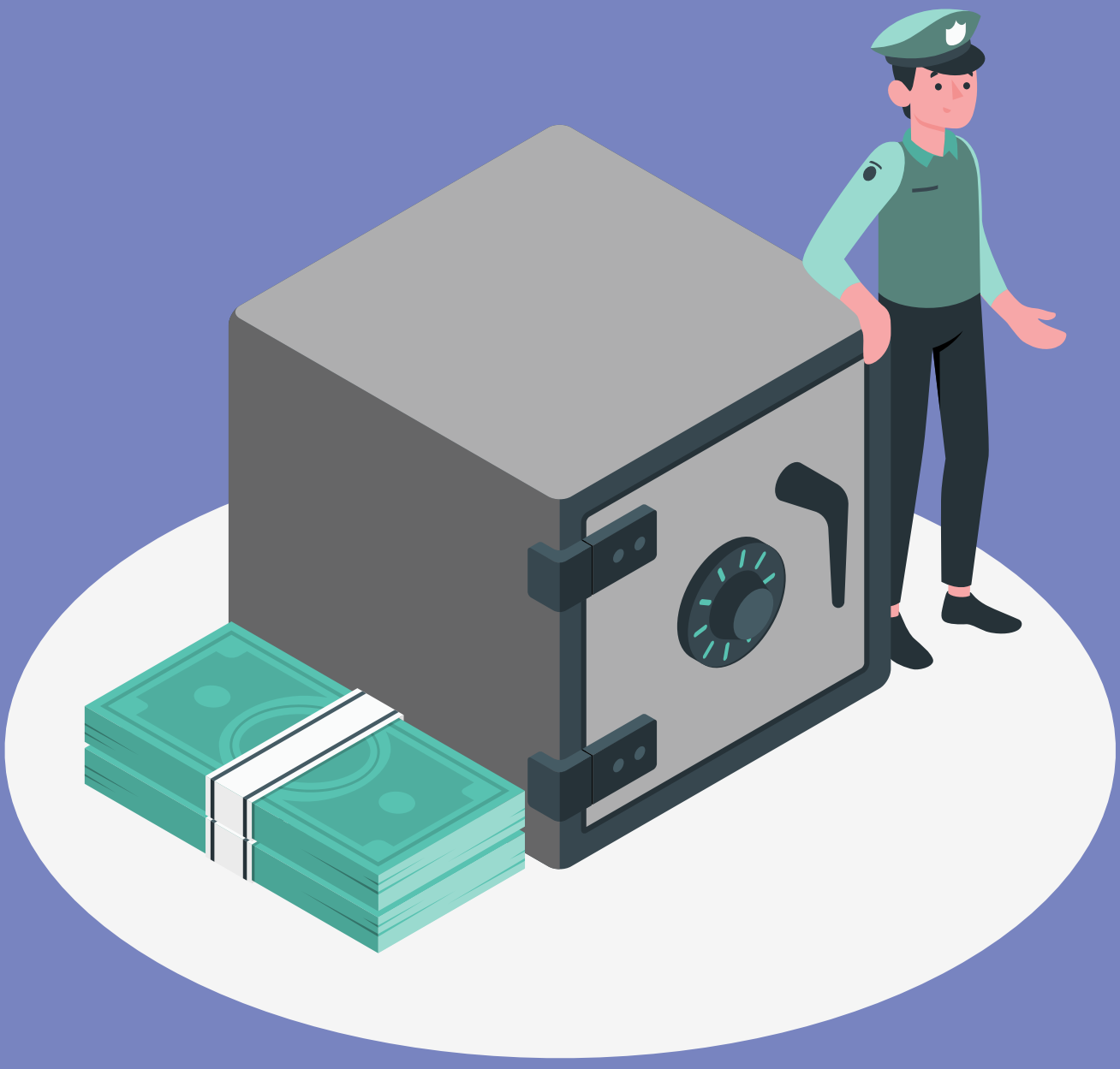
Assembly is finished and you can move on with coding steps.

MicroBlocks Code of The Project:

Money-Box

```
#Money Box Project
from microbit import *
from picobricks import *

# Function Initialization
motor = motordriver()
motor.servo(1,180)

trashDetected=0
distance=100

while True:
    rawdistance=measure_distance()
    if rawdistance<1200:
        distance=rawdistance
    motor.servo(1,180)
    sleep(500)
    if distance<5:
        trashDetected=1
        sleep(300)
    if distance>9 and trashDetected==1:
        trashDetected=0
        motor.servo(1,90)
        sleep(500)

```

Safe Box



Safe Box Project

The PicoBricks Pass Box is an educational project kit designed to create a pass box that automatically locks after assembling the wooden pieces and PicoBricks modulesaccording to the installation steps.

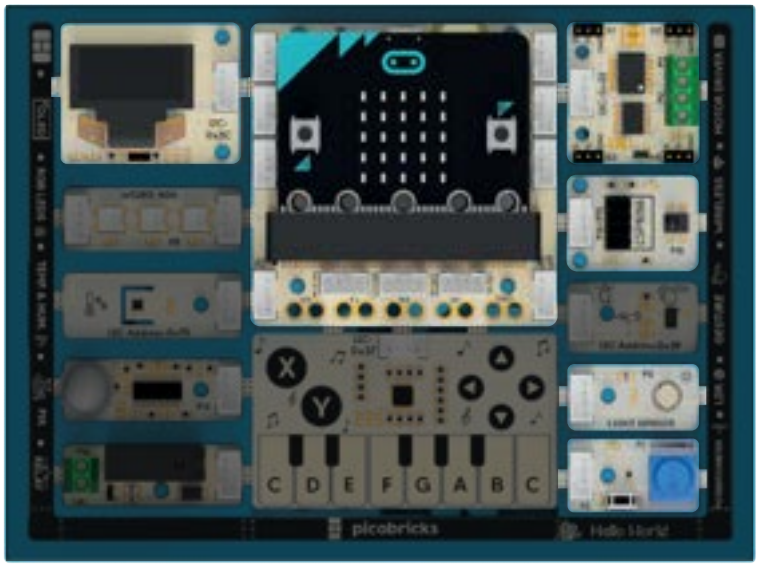
In this project, when the correct password is entered by using a potentiometer and button, the door of the safe opens. After closing the door, it automatically locks thanks to the LDR sensor inside the safe.

Project Details:

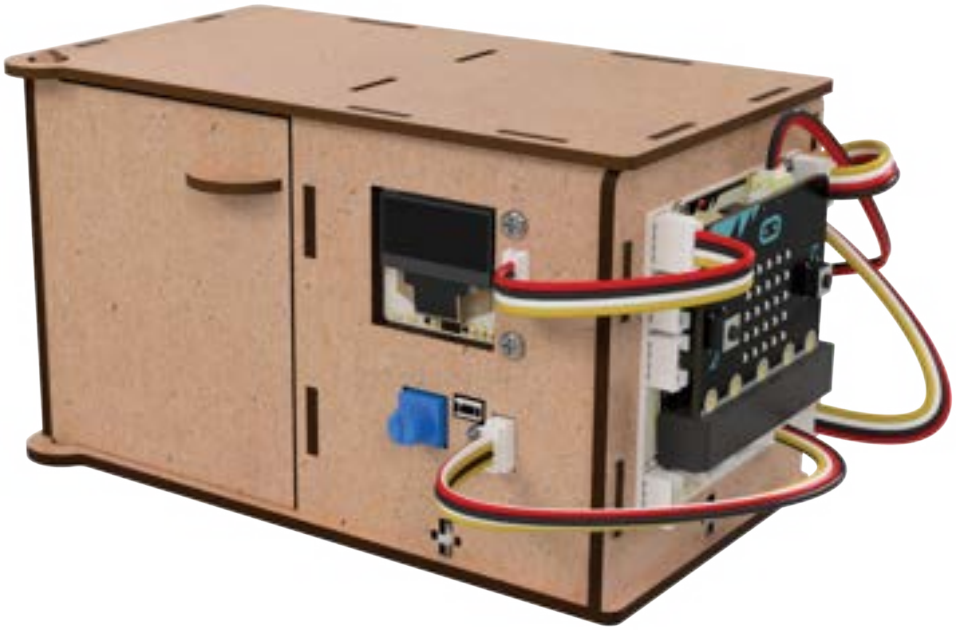
In this project, when we correctly enter the password we have set in the code by using the potentiometer and button module, the servo motor moves to the specified position, and the door opens. Through the LDR module, the closure of the pass box lid is detected, triggering the servo motor to operate and lock the door. We will create the code blocks that enable these functions. With the PicoBricks OLED display module and Micro:Bit Matrix LEDs in the Pass Box project, we will obtain visual output.

Connection Diagram:

You can assemble this project by breaking the PicoBricks modules at the proper points.



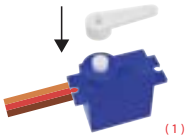
Project Images:



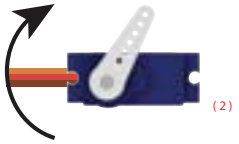
Setup Steps of The Project:

Servo Motor Calibration

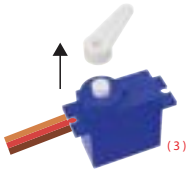
Before starting the assembly, you have to manually calibrate the angles of the servo motors. Otherwise, Servo Motors won't be working properly.



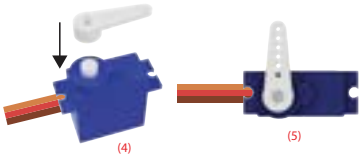
Attach the servo horn to the servo motor (1)



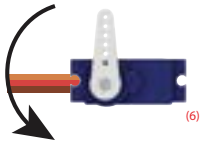
Then slowly turn the servo horn clockwise until it stops. It is not a problem if the servo horn is not the same as the angle shown in the image above. The important thing here is that you have hit the last angle of the servo. (2)



Remove the servo horn from the servo motor (3)

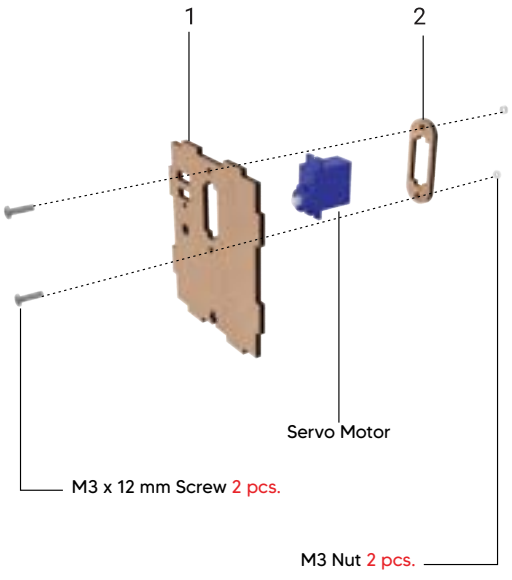


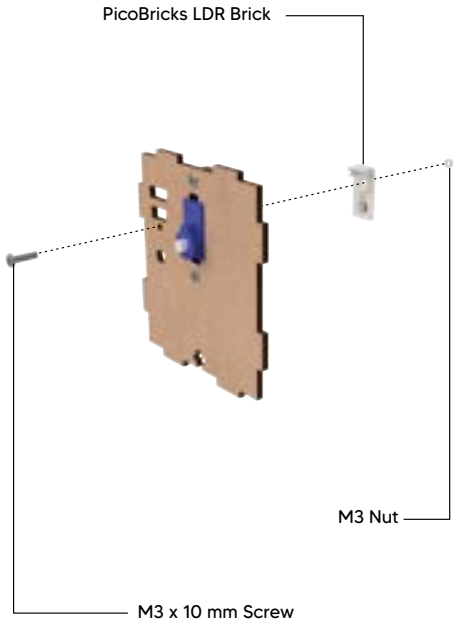
Reattach (4) and reposition the servo horn perpendicular to the servo motor as shown. (5)

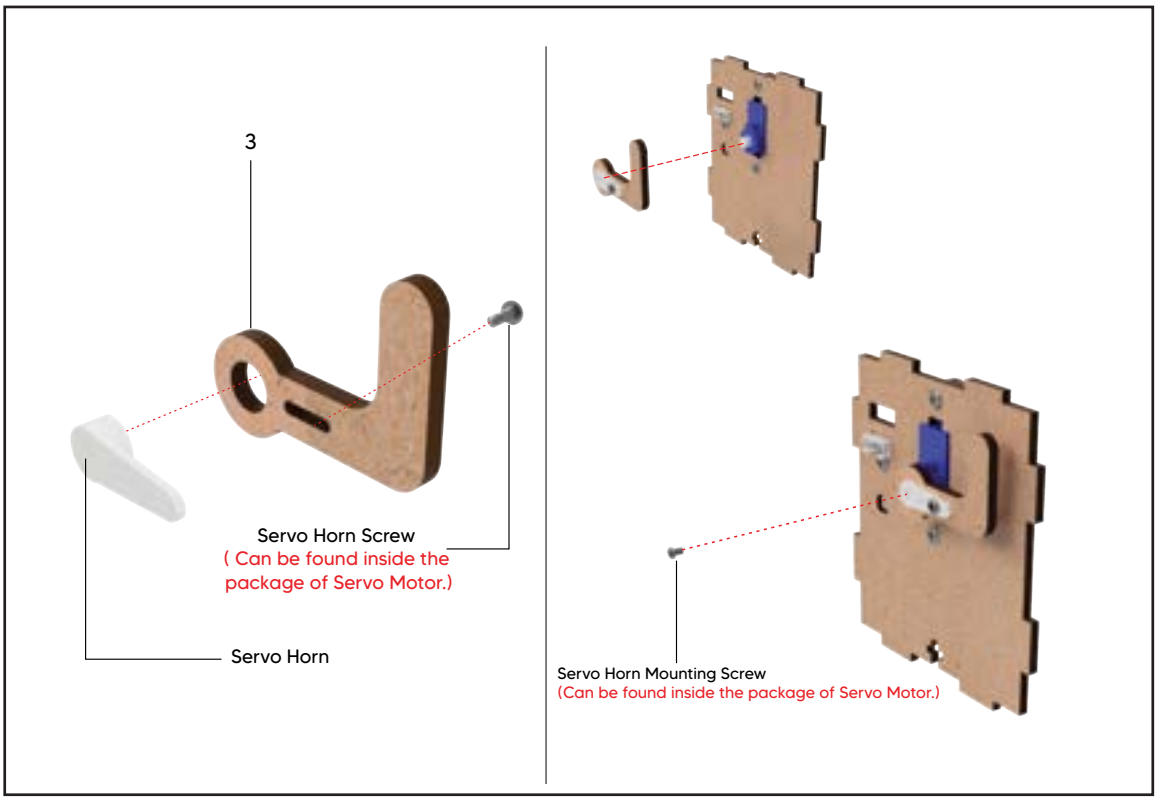


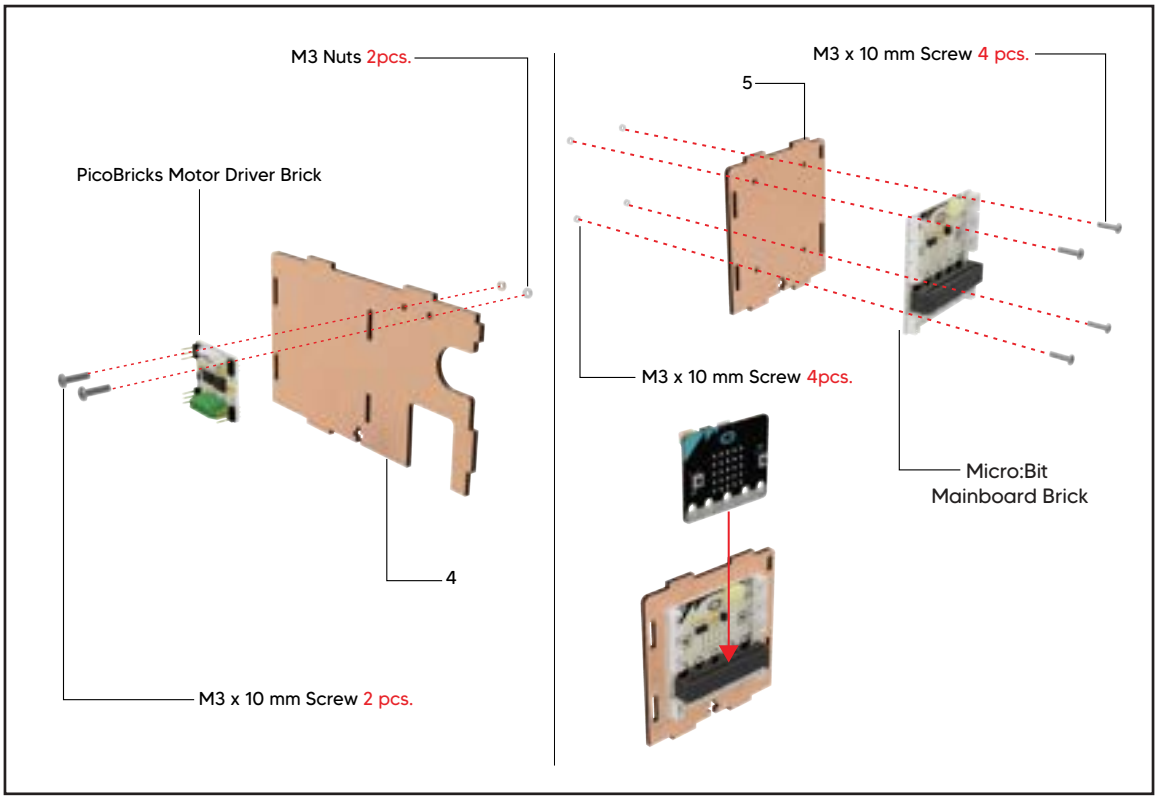
Slowly turn the servo horn counterclockwise (6) until it is parallel with the servo motor, as seen in the image.(7)

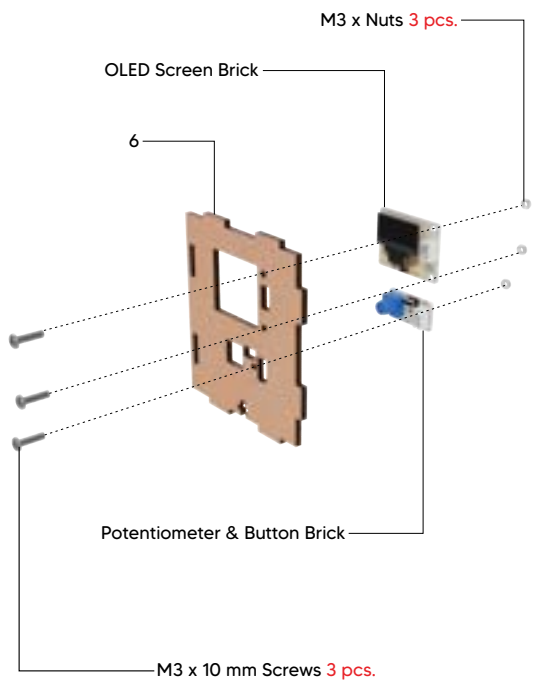
When this step is finished, it means that the servo motor is in the center position. It is important that you apply this process to other servo motors in the set. Afer processing the other motors, remove the servo horn and set aside for assembly.

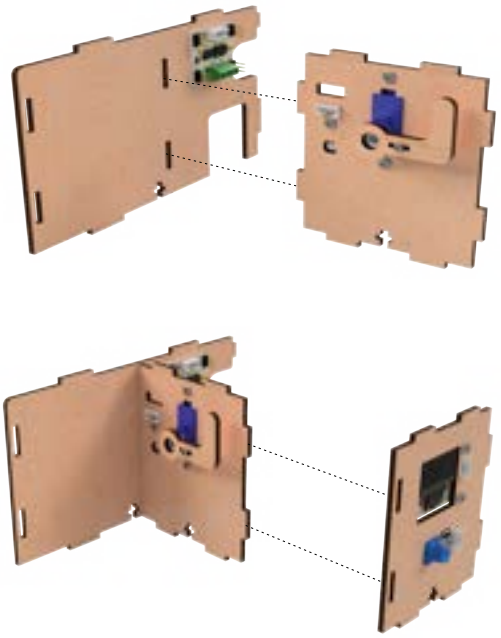


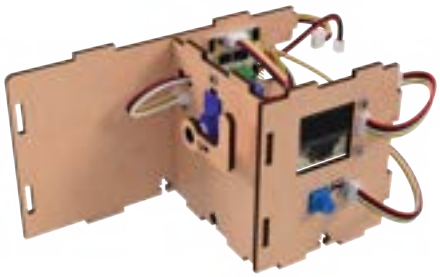



3
Servo Horn Screw
(Can be found inside the
package of Servo Motor.)
Servo Horn
Servo Horn Mounting Screw
(Can be found inside the package of Servo Motor.)

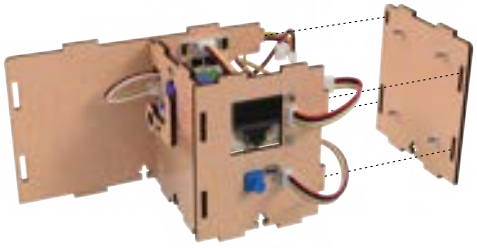

M3 Nuts 2pcs.
PicoBricks Motor Driver Brick
4
M3 x 10 mm Screw 2 pcs.
M3 x 10 mm Screw 4 pcs.
5
M3 x 10 mm Screw 4pcs.
Micro:Bit
Mainboard Brick

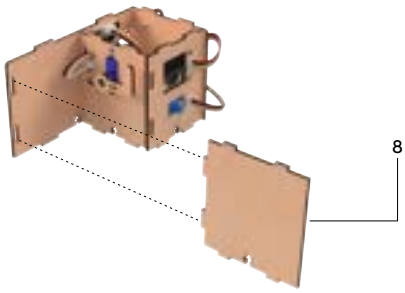


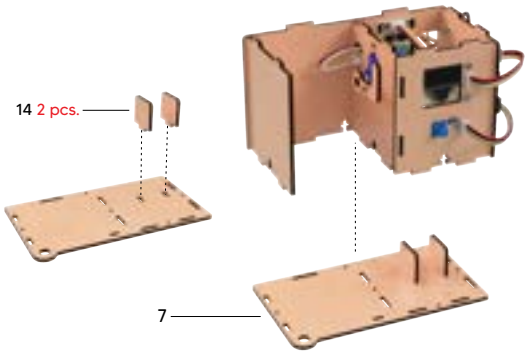


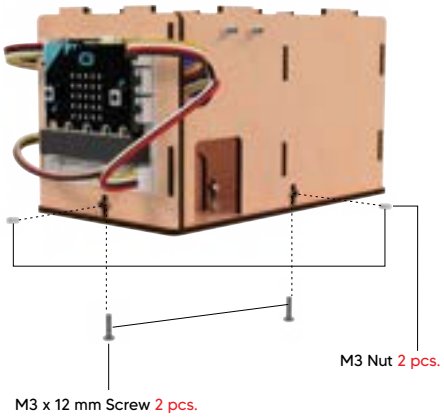


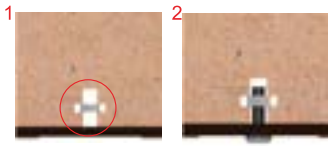
Make sure all cables are ready to be socketed to MainBoard. You can go back to page 18 for circuit diagram.



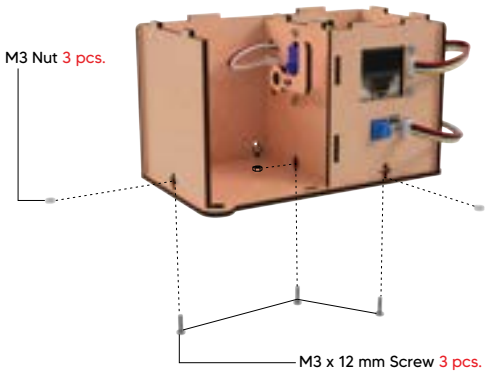


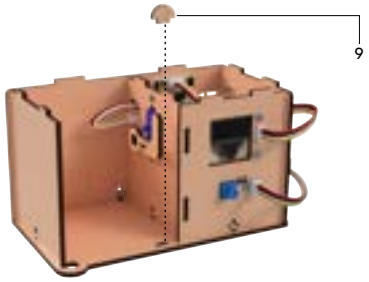


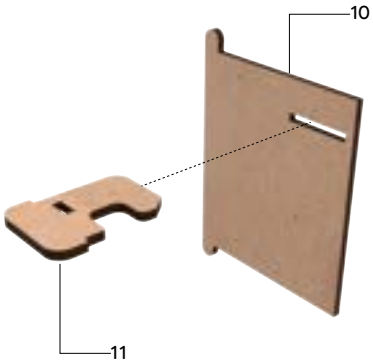


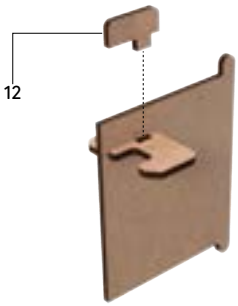


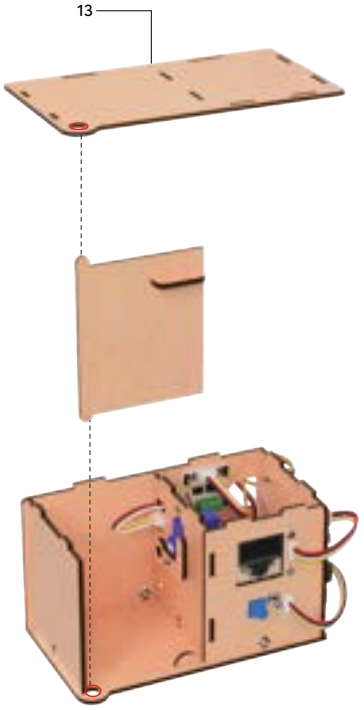
For T Slot joint setup first, place the M3 Nut into the Nest (1) Then tighten the Joint with using M3 x 12 mm Screw.(2)



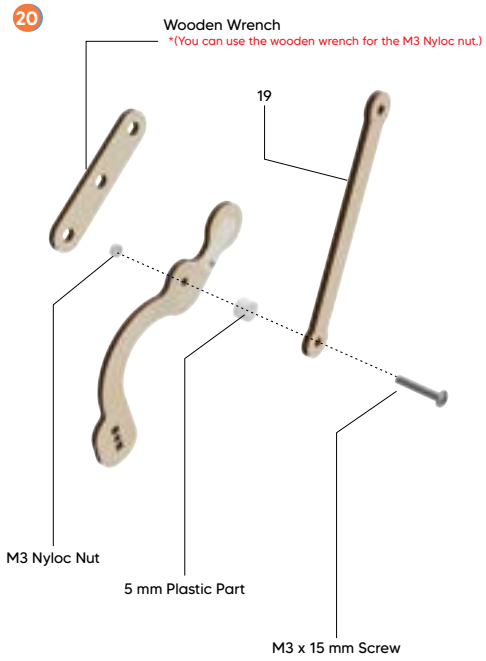




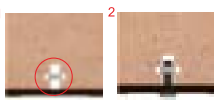




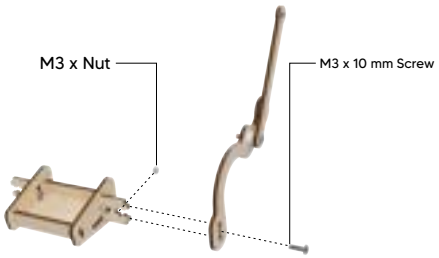
20



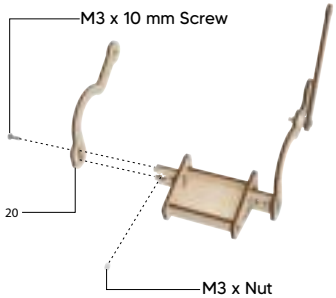
21

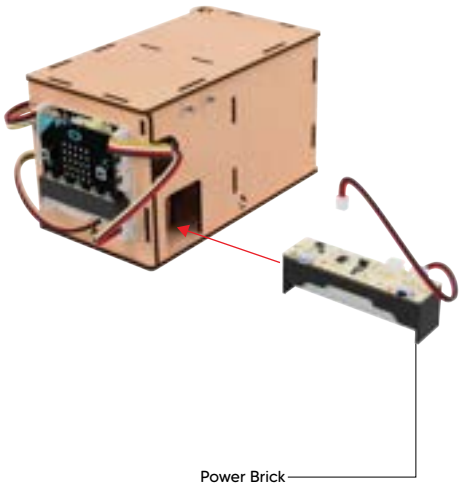


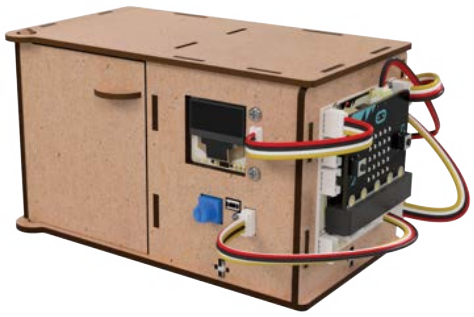
For T Slot joint setup first, place the M3 Nut into the Nest (1) Then tighten the Joint with using M3 x 10 mm Screw.(2)



22







The Assembly is finished and you can move on the coding steps.

MicroBlocks Code of The Project:

Safe-Box

```
#Safe Box Project
from microbit import *
from picobricks import *

# Pin Initialization
LDR_Pin = pin0
Pot_Pin = pin1
Button_Pin = pin2

# Function Initialization
oled = SSD1306()
oled.init()
oled.clear()
motor = motordriver()

display.show(Image.HAPPY)
password=[1,2,3,4]
userPass=[]

def startOLED():
    oled.add_text(0,1,"Close the lid")
    oled.add_text(0,2,"to lock the")
    oled.add_text(0,3,"SafeBox")

def addList():
    #counter=0
    if counter>0:
        oled.add_text(2,3,str(userPass[0]))
    if counter>1:
        oled.add_text(4,3,str(userPass[1]))
    if counter>2:
        oled.add_text(6,3,str(userPass[2]))
    if counter>3:
        oled.add_text(8,3,str(userPass[3]))

def controlLoop():
    control=0
    for i in range(4):
        if password[i]!=userPass[i] :
            control=1
    if control==1:
        display.show(Image.NO)
    else:
        display.show(Image.YES)
        motor.servo(1,90)
        sleep(3000)
        oled.clear()

while True:
    display.show(Image.HAPPY)
    counter=0
    startOLED()
    light = LDR_Pin.read_analog()
    pot = Pot_Pin.read_analog()
    button = Button_Pin.read_digital()
    print(light)
    if light<50:
        sleep(2000)
        oled.clear()
        motor.servo(1,180)
        while counter<4:
            pot = Pot_Pin.read_analog()
            userNumber=round(round( pot - 0 ) * ( 9 - 0 ) / ( 1023 - 0 ) + 0)
            oled.add_text(2,1,"Password:")
            oled.add_text((2+(counter*2)),3,str(userNumber))
            addList()
            button = Button_Pin.read_digital()
            if button==1:
                userPass.insert(counter,userNumber)
                counter=counter+1
                sleep(200)

        controlLoop()
    sleep(500)
    counter=0
    control=0
```