# Nibble coding - first steps

## Introduction

# Installation

Welcome to the **CircutBlocks** tutorial!

If you don't know, **CircuitBlocks is a Scratch-based** (a visual block programming language) **IDE** in which you can easily and effectively create and upload your projects to the Nibble console.

This tutorial is going to be broken down into several chapters, each representing one of the important aspects of the IDE.

**NOTE: CircuitBlocks will be referred to as CB in the future**

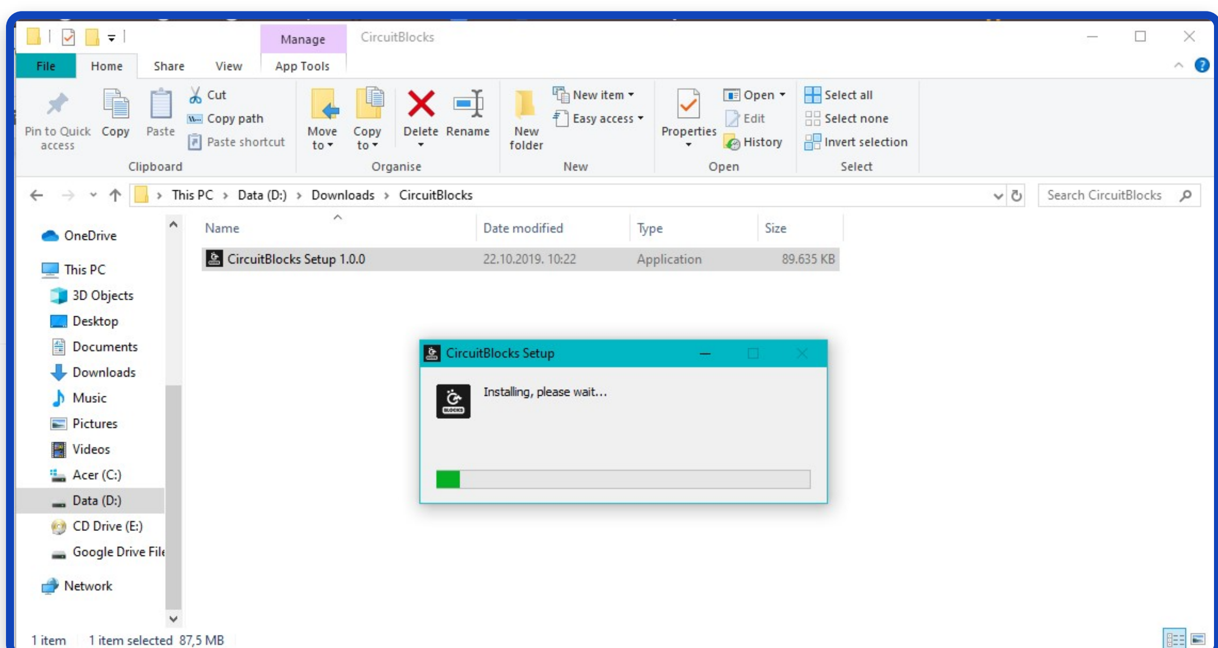
## Installation

CB is supported on all major platforms.

The installation process is really easy as you just need to download the file and install it just like you would with any other program on your preferred platform.

### Windows

- **Go to the [CircuitBlocks download page]()**
- **Download the latest version *Windows.exe (ex. CircuitBlocks-1.1.0-Windows.exe) -** Check if you have a 32 or 64 version. Go to Settings on your PC, click on the System option and find the About section where you'll see the system type.
- Double-click the file to run the executable
- CB will automatically install and create a new desktop shortcut

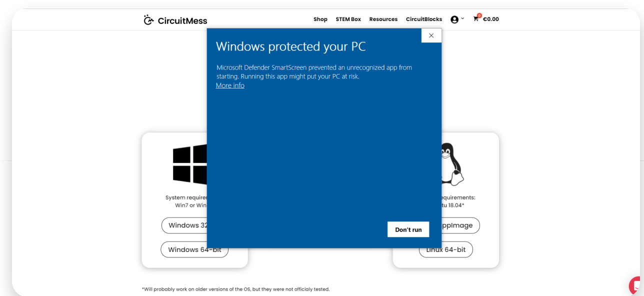



> **Your PC is NOT at risk!**

There is a possibility that a notification that says your PC is at risk may pop up when you try to install CircuitBlocks. Don't worry, this happens regardless of CircuitBlocks being safe to run. See the instructions below on how to handle this notification.

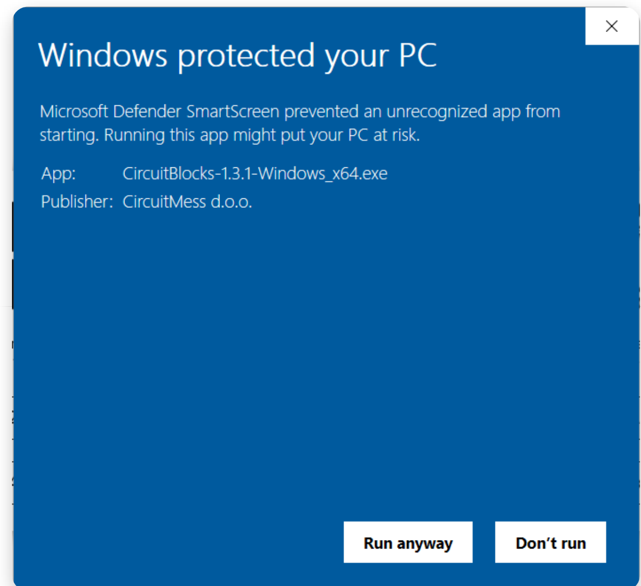
This is the message you might get when trying to install CircuitBlock on your PC. Windows reports a threat despite the program being safe to download and run.

Please proceed with installing by clicking on '*More info*' option.



After you click on '*More info*' option, an option to '*Run anyway*' should appear at the bottom of the window.

Proceed by clicking on *'Run anyway'.*




## Linux

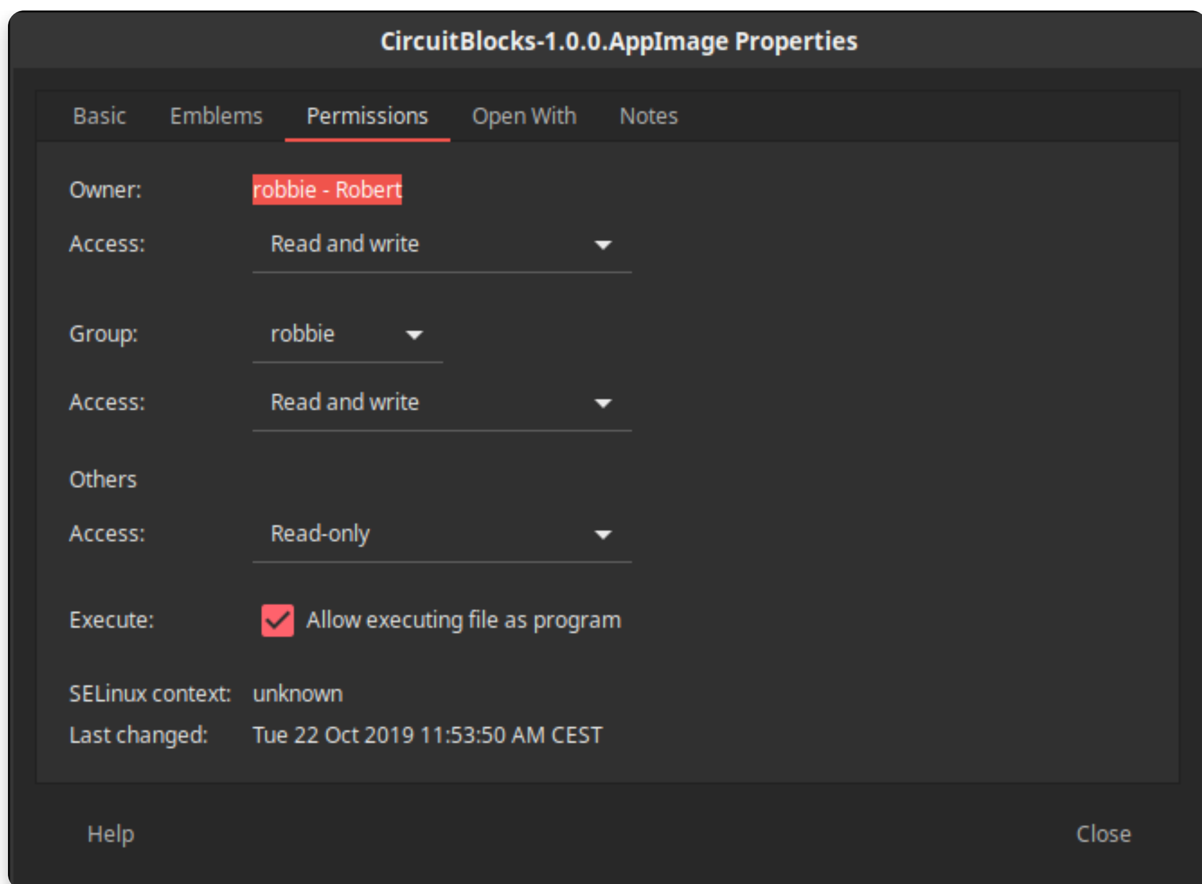
There are two ways of installing CB on Linux

**Installation:**

- **Go to the [CircuitBlocks download page]**
- **Download the latest version *Linux_amd64.deb (ex. CircuitBlocks-1.0.1-Linux_amd64.deb)**
- Double-click the file to run the installation (Ubuntu)
- Write inside a terminal `sudo dpkg -i <path to the downloaded file .deb>` (Other Linux distros)
- CB will automatically install and create a desktop entry

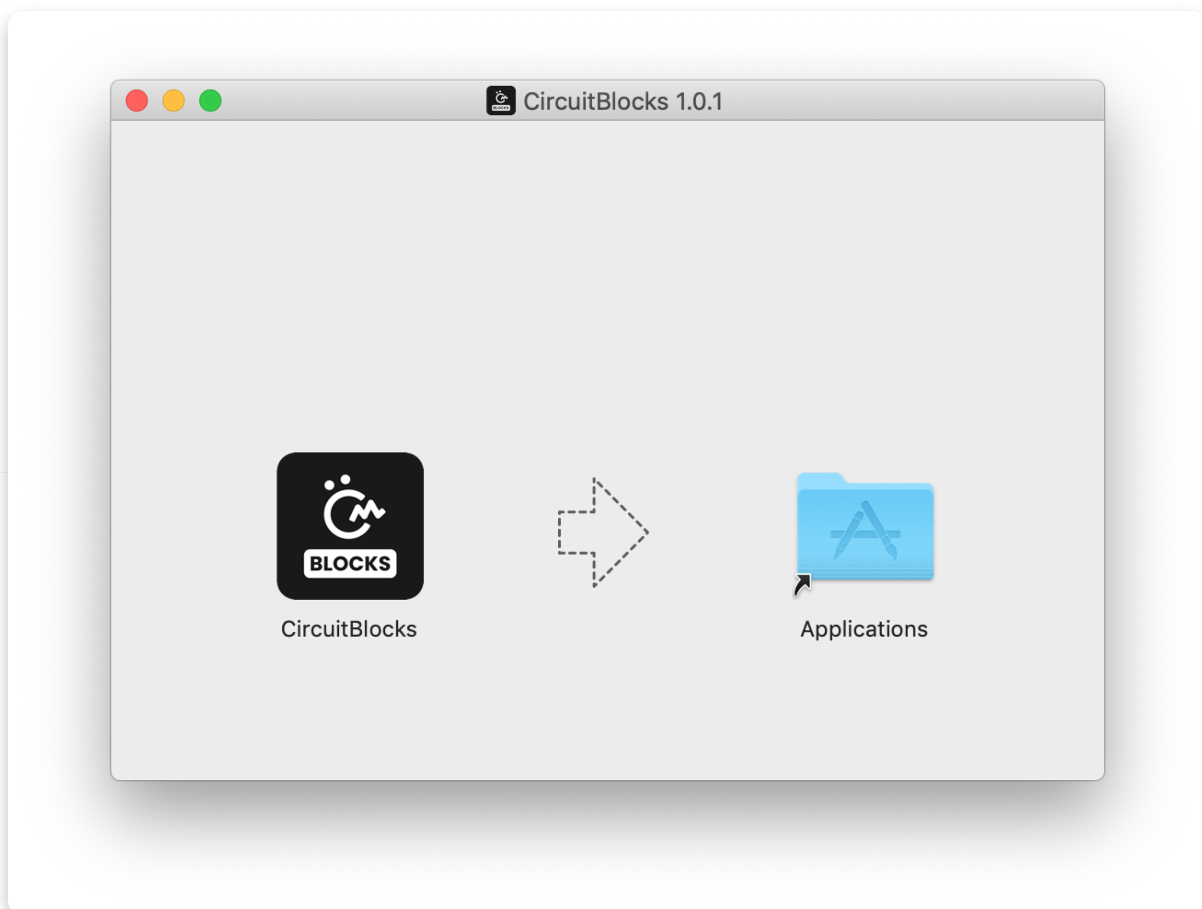
**Stand-alone (AppImage):**

- **Go to the [CircuitBlocks download page]**
- **Download the latest version *Linux.AppImage (ex. CircuitBlocks-1.0.1-Linux.AppImage)**
- Right-click on the file and select **'Properties'**
- Go to '**Permissions**' and tick '**Allow executing file as program**'
- Double-click the file and the installation will complete automatically



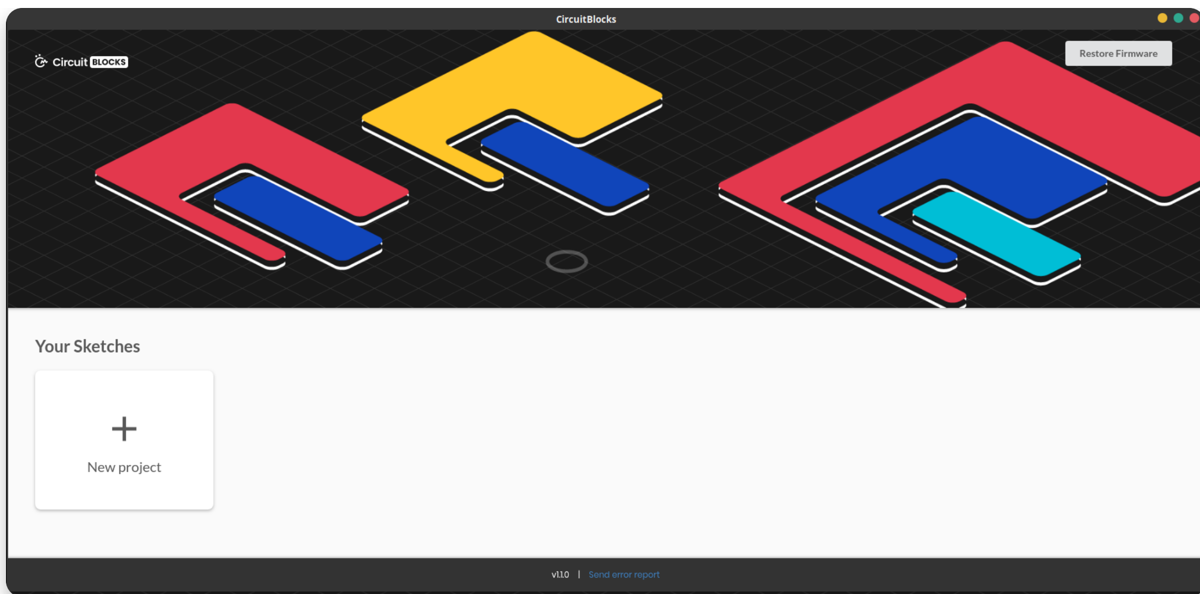

## Mac OS

- **Go to the [CircuitBlocks download page]()**
- **Download the latest version *Mac.dmg (ex. CircuitBlocks-1.0.1-Mac.dmg)**
- Move the files to **'Applications'** folder
- CB will be installed automatically




# Basics

## Interface




Starting CB will open a window like this.

It's pretty simple – starting a **new project (sketch)** can be done by clicking the 'New project' button.

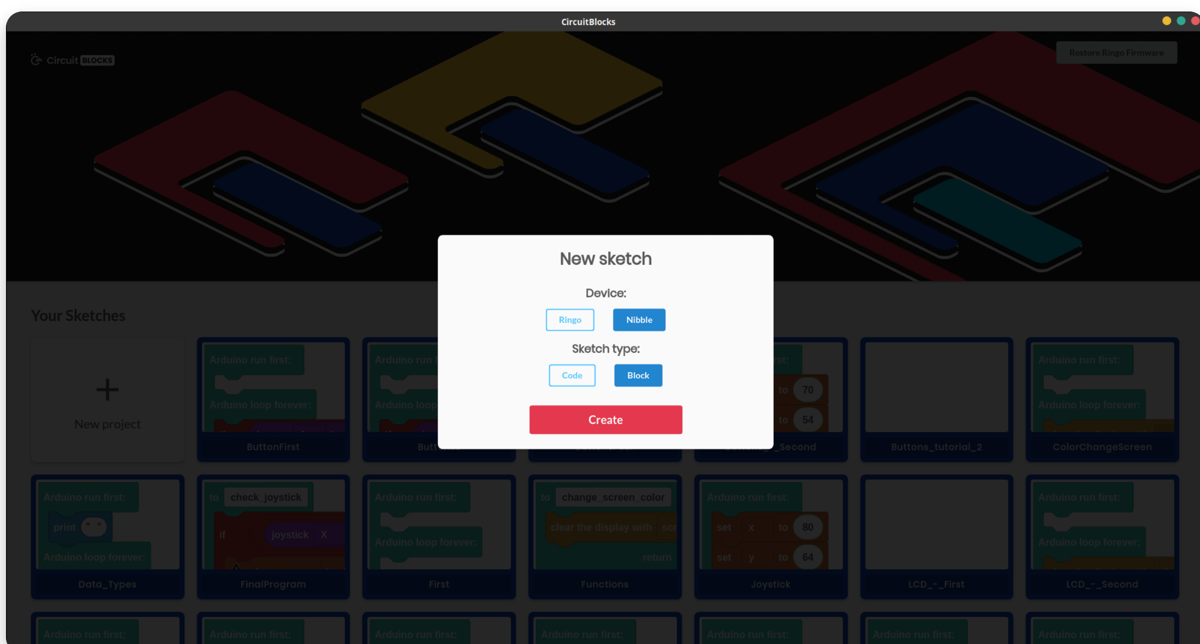
**Saved sketches** will appear right next to that button and you can access them at any time.

Whenever you encounter an error, press the **'Send error report'** at the bottom of the main screen. You'll get the report and the error number - you can use that in the CircuitMess community forum so that our team members can help you more efficiently.

# Starting a new sketch

When starting a new sketch you'll get an option to choose from a **Ringo project** and a **Nibble project**, as well as a **code project** and a **block project**.

Since this tutorial is going to be focused on Nibble, we're going to be selecting a Nibble project.




We've added a whole new layer to the CircuitBlocks - you don't have to use the blocks anymore, but rather can write whole games by using C/C++ just like in Arduino IDE.

However, we're going to focus on the **block building**, rather than **code writing**.

**What you can also do, is to make part of your game using blocks and then transfer it to the coding version to add some additional code, like buzzer**

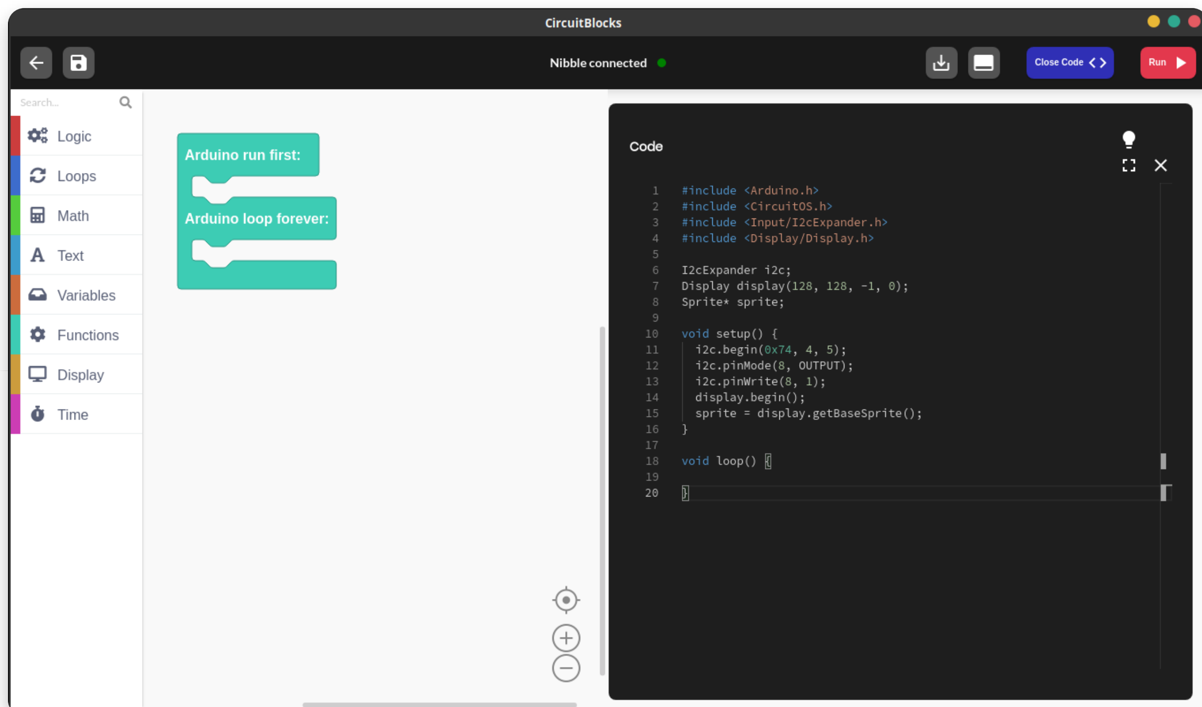
**sounds. This will allow you to use the easier way of coding (via blocks) most of the time and in the end, just add a few details to top it off.**

You can always copy the code from the block version and switch it to the coding version. However, you cannot go from the coding version back to the block version. That's why we encourage you to use the block version whenever you can.

**Code** project is pretty much the same thing you'll get in **Arduino IDE** - straight up Arduino C/C++ that will run your programs based on the code you've written.

On the other hand, **block** projects are the real ones here. There you'll be able to generate program code from the **blocks that you drag-and-drop.** This is a real Scratch-like experience and it is highly recommended to everyone beginning their programming career or learning the ins and outs of the Nibble library. The code that is generated can then be copied and modified in the code project.

**We advise that you start the block project regardless of your programming skills just so you can get to know the device better!**



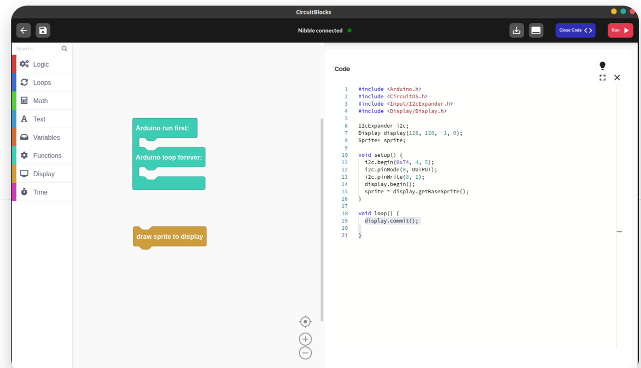
This is the main interface you will be looking at most of the time.

On the **top of the screen**, there is a **toolbar** from which you can access main program functionalities.

The **block selection** tool is located on the **left side**.

**In the middle of the screen is where you'll be "drawing" your code with the blocks**.

On the **right** is where those drawings will be translated into **code**. It is the same code editor used in the VS Code, but it is not editable. If you want to edit the code, you have to copy it and transfer it to a new code based project.



If you don't like the dark mode (blasphemy if you ask us!), you can easily switch it by pressing the light bulb button in the top right corner of the code editor.

## Toolbar

There are seven main components here:

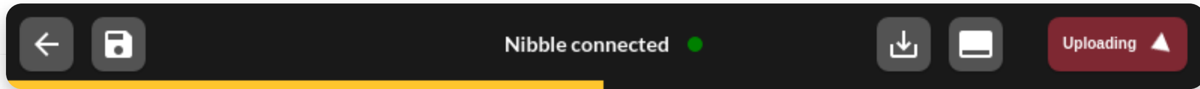
1. **Back to the main menu** – returns to the main menu without saving
2. **Save/Save As...** – saves the file in the default sketch directory
3. **Nibble connected indicator** – indicates whether the console is connected or not
4. **Export to binary** – creates .bin file of the code which can be directly uploaded to the console
5. **Open serial** - open Serial port
6. **Close Code** – closes the code editor to expand the 'drawing' area
7. **Run** – compiles and uploads the code to your console

When the console is not connected, the red 'Run' button will turn grey and the indicator will say '**Nibble disconnected**' with the red dot instead of a green one.

While a code you've written is being uploaded, a progress bar will appear right

below the toolbar which indicates how much data has been compiled/uploaded so far.

When it reaches 50% it means that the compilation is over and when it reaches 100% it means that the upload to the console is finished.

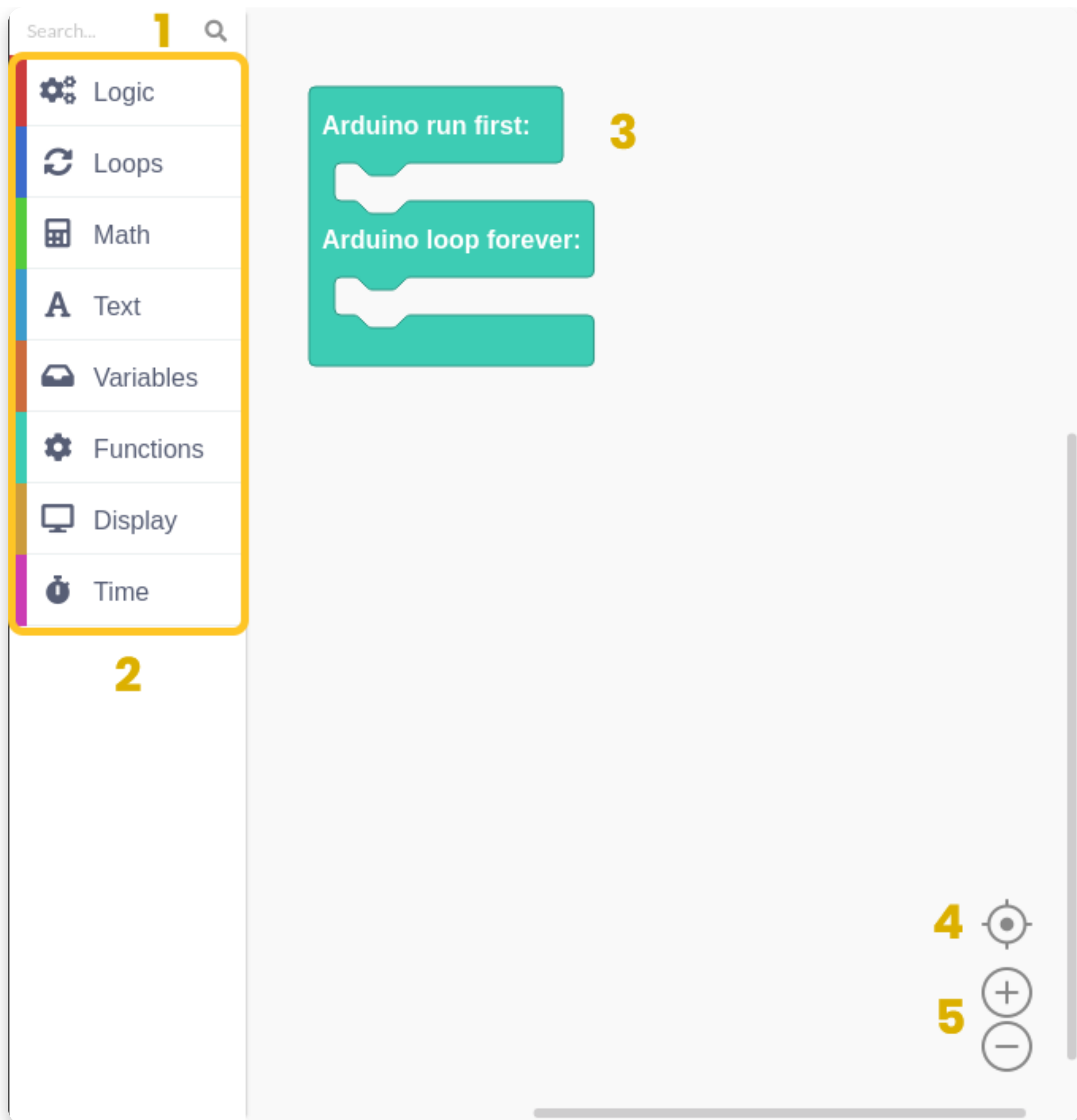



1. **Main code screen** – this is where the blocks will appear in the textual format, code words are colored while the regular text is white
2. **Light mode switch** – switch the background of the code editor between black and white
3. **Expand** – stretches the code editor over the whole window
4. **Close** – closes the code editor, same functionality as 'Close Code' button from the toolbar

Code

```
#include <Arduino.h>
#include <CircuitOS.h>
#include <Input/I2cExpander.h>
#include <Display/Display.h>

I2cExpander i2c;
Display display(128, 128, -1, 0);
Sprite* sprite;

void setup() {
  i2c.begin(0x74, 4, 5);
  i2c.pinMode(8, OUTPUT);
  i2c.pinWrite(8, 1);
  display.begin();
  sprite = display.getBaseSprite();
}

void loop() {
  sprite->clear(TFT_WHITE);
  display.commit();

}
```

The 'Drawing board' is the most complex part of the IDE. It's where all the magic happens. It is divided into two main sections. On the left is a board where you select the blocks and on the right is a board where you place them. Each type of block has its own color so it's easily recognizable.




1. **Search bar** – dynamic search bar with which you can easily find any component you're looking for
2. **Component selector** – divided into categories by names and colors
3. **Drawing board** – a place where you create your programs by placing the components in a certain order
4. **Center icon** - places your blocks in the center of the board
5. **Zoom buttons** - zoom in and out of the board

Each of the components will be explained in detail and come with a few examples of how to use them together.

## Using blocks

# Types of blocks

There are a total of **nine** block types in CB. Each of them is represented by their own color. Every block translates to code, which is then compiled and uploaded to the console, just like on every Arduino based platform.

Pressing on every block type will open a section from which you can drag-and-drop those blocks into the drawing area.

Also, pressing on **'More'** will open even more blocks that are not so commonly

used.

There are two main functions of every Arduino code – **void setup()** and **void loop()**.

Everything that goes into the **void setup()** the function will run only once. It is primarily used for starting the software, initializing and declaring variables and running functions that only have to run once (ex. Intro screen in a video game).

The **void loop()** is where everything else takes place. It basically runs every bit of code inside it over and over again (speed depends on the device – just imagine it's ultra-fast!). It should pretty much follow the refresh rate of the screen and make the program do things accordingly.

Every block you place automatically goes into the **void loop()** function.

If you wish to put something in the **void setup()**, you have to drag the main block from **Functions** and place your blocks inside as you wish, but more on that a little bit later.

# Elliptical blocks

**Elliptical blocks** represent variables. Whether we're talking about integers, strings or other variable types (other than boolean), they can all be recognized by the same shape.

Also, larger blocks that have elliptical shape return either integer or float values.

When ever you find circular "holes" inside some blocks, that is the place where variables can be inserted. It's most commonly found in **comparison** or **action** blocks.

# Triangular blocks

Boolean variables are represented by triangular blocks.

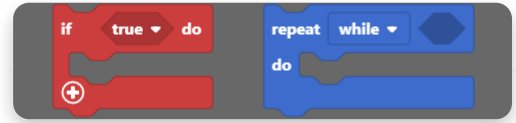
Both variables (true and false), as well as functions that return boolean values, have the same shape.

Regardless of color, each of these blocks returns either true or false.

Triangular "holes" require boolean blocks to be inserted.

# Building blocks

Everything else is basically a building block. Those are functions that have no return value (they return ***null***). Both elliptical and triangular blocks first have to be placed inside of the building blocks in order to act as part of the program.




They have a specific "puzzle" shape and can be stacked inside each other.

The main **building block** is located inside the **'Functions'** section.

It basically gives you two main building blocks sections.

Everything that is placed inside **Arduino run first** goes into **void setup()** and everything that is placed inside **Arduino loop forever** goes into a **void loop().**

## Inserting blocks

Now, this is the main part.

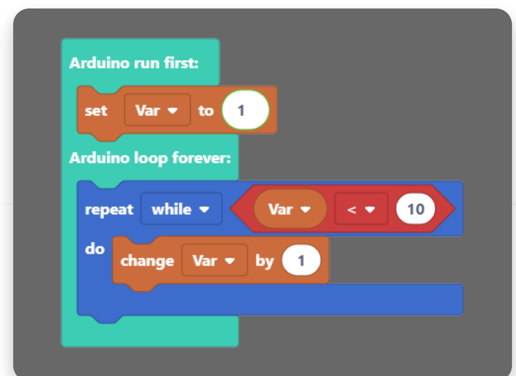
The whole point of blocks-like IDE is connecting blocks and placing them one inside another.

It is all done by simple **drag-and-drop** action.

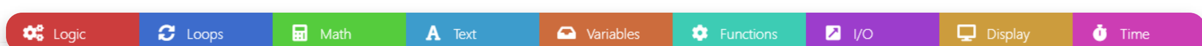
Here is an example of a program that will set the variable **Var** to **1 and then increase that variable while it is smaller than 10**.

At the end of the program, **Var will be 10**.

This is just a simple example and block-building will be further explained in the following chapters.




# Block sections




There are a total of nine sections in CircuitBlocks. We've organized them so that you'll be able to find everything in maximum two clicks.

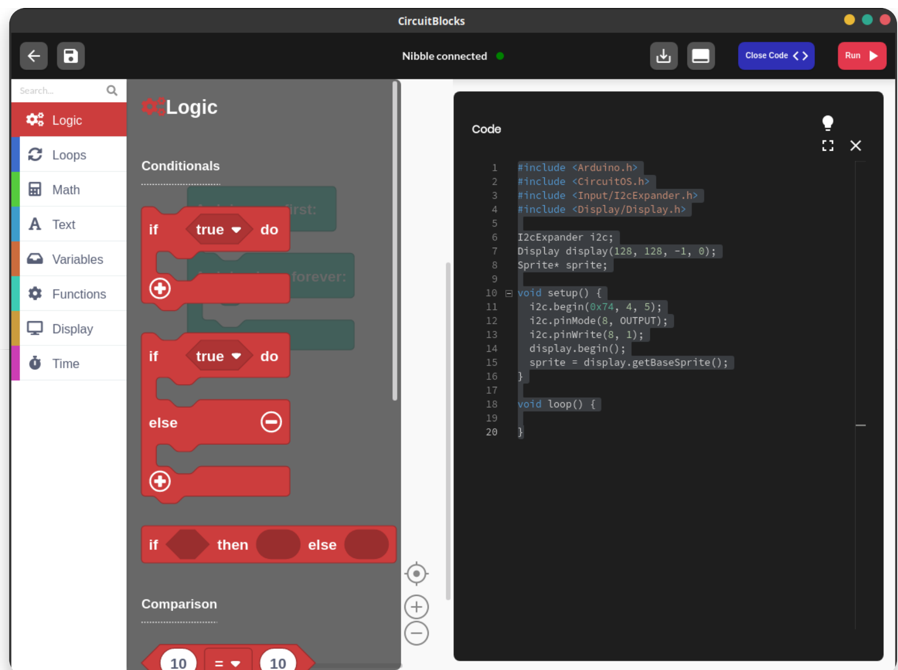
Sections themselves are pretty explanatory but we'll go through them all just so you can get a little bit better understanding of the whole concept.

Some of the sections also have additional blocks (in the **'More'** menu) where you'll be able to find some of the functions that are not used that often, but can still be useful.

## Logic

This is where the base of every code is located.

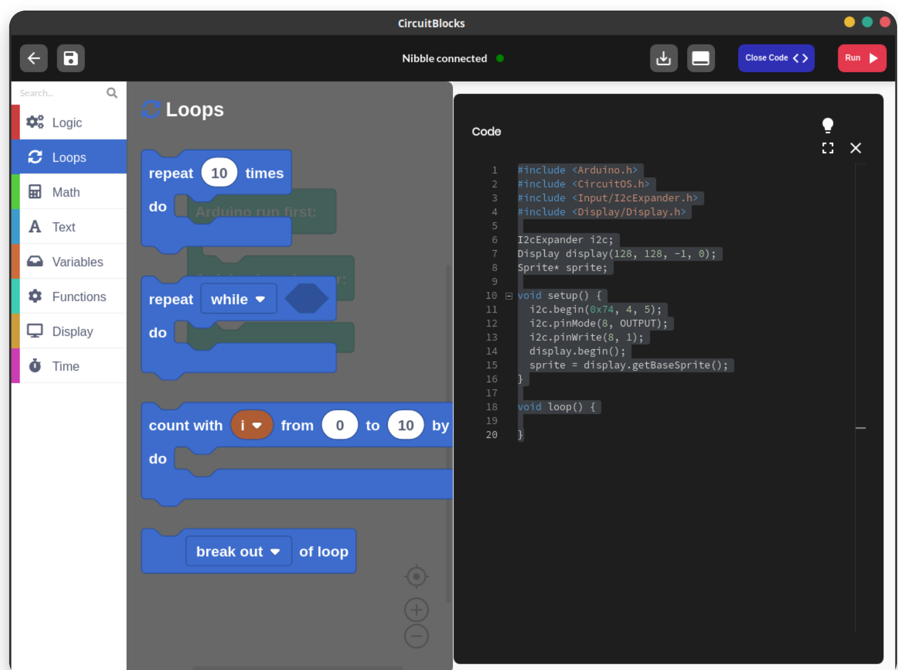
Every **if, if-else, else** function, as well as comparisons, **and/or, not, true/false** and other logical operators.




# Loops

Loops are functions that repeat everything inside for a specific amount of time.

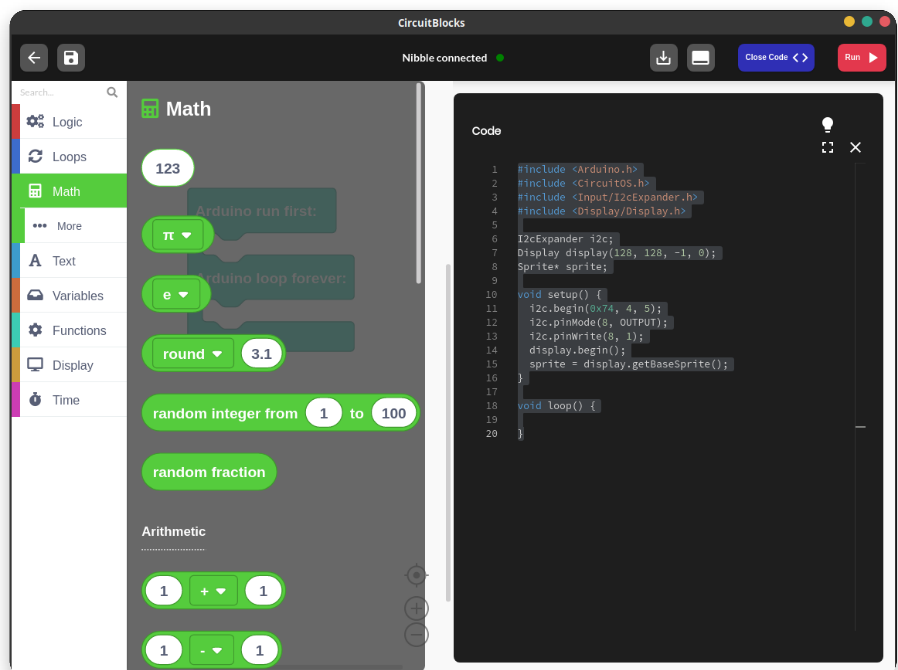
They can either have conditionals, and repeat for as long as that condition is met, or they can have a pre-determined amount of repeats.




# Math

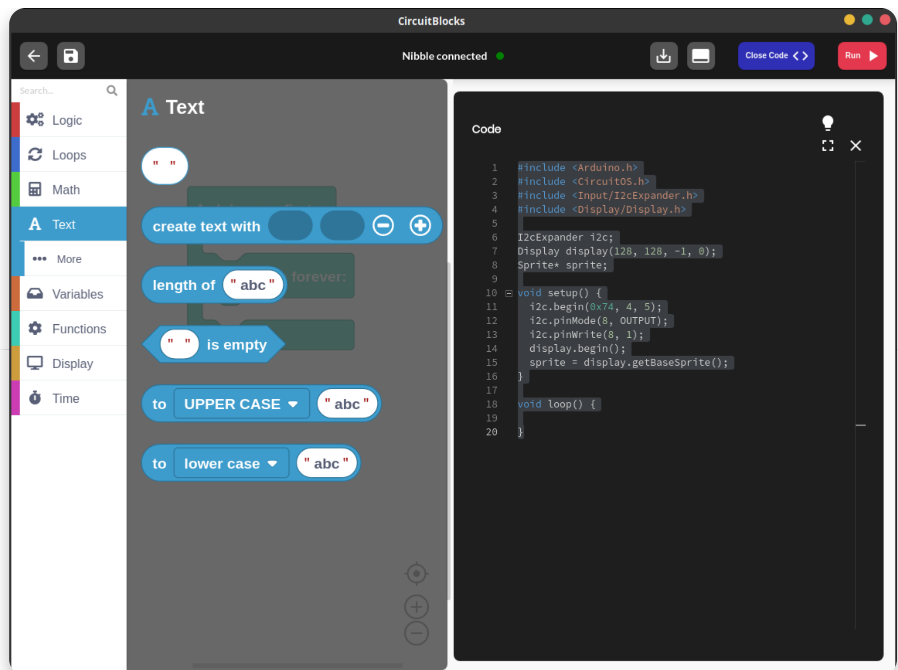
Pretty much every math function is located here.

From basic operations to rounding numbers and working with angles, you will easily trigger your inner Einstein or Pythagora in a matter of seconds!




# Text

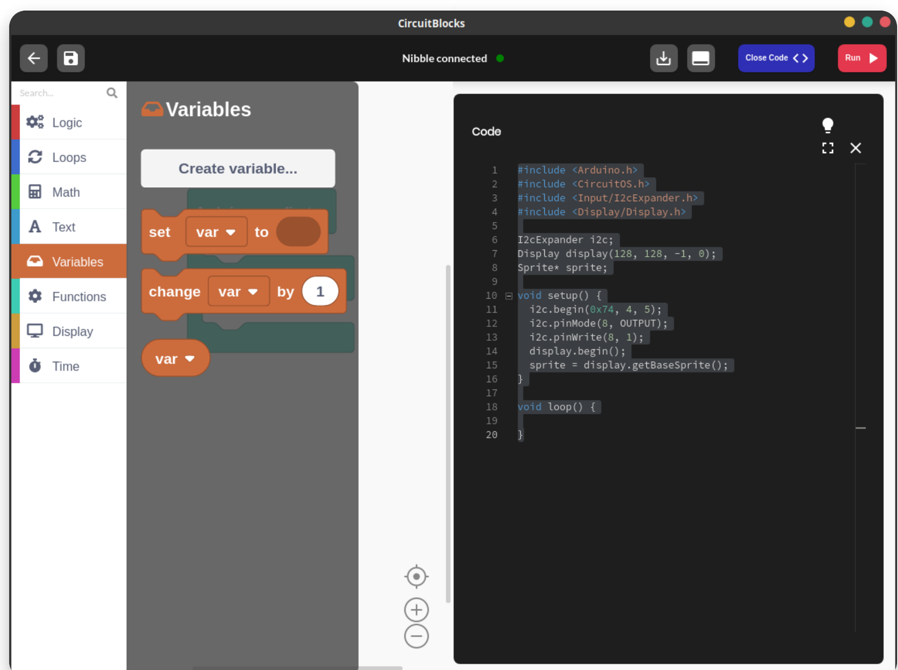
Strings, characters, and string manipulation. Great place for creating new text and implementing it to your programs.




## Variables

Create a variable of any type and set its name and desired value.

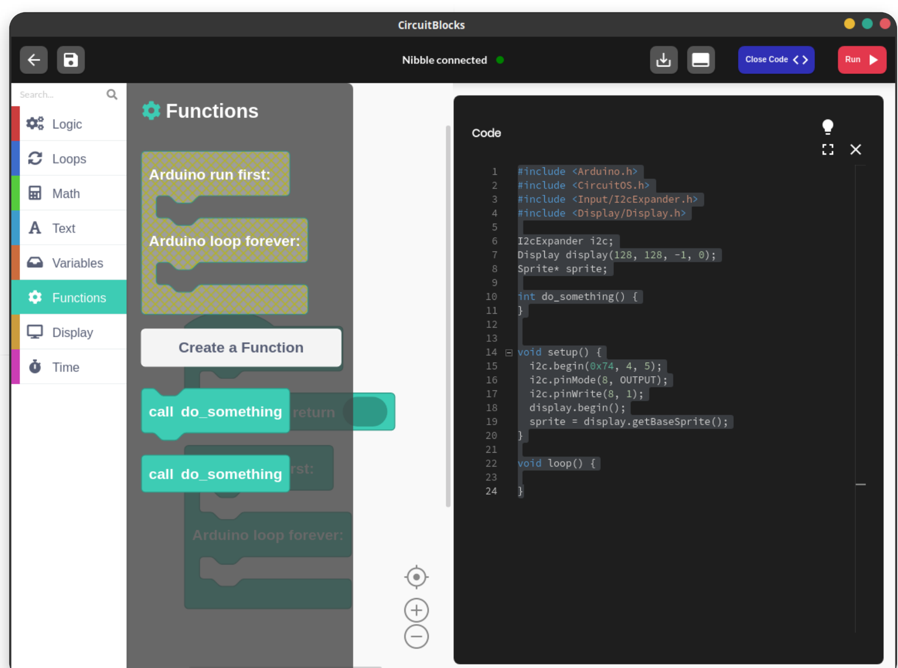
CB will automatically recognize the type of the variable (int, double, string, boolean) so you don't need to worry about that.




## Functions

The Default Arduino function (which is explained on the previous page) is located here.

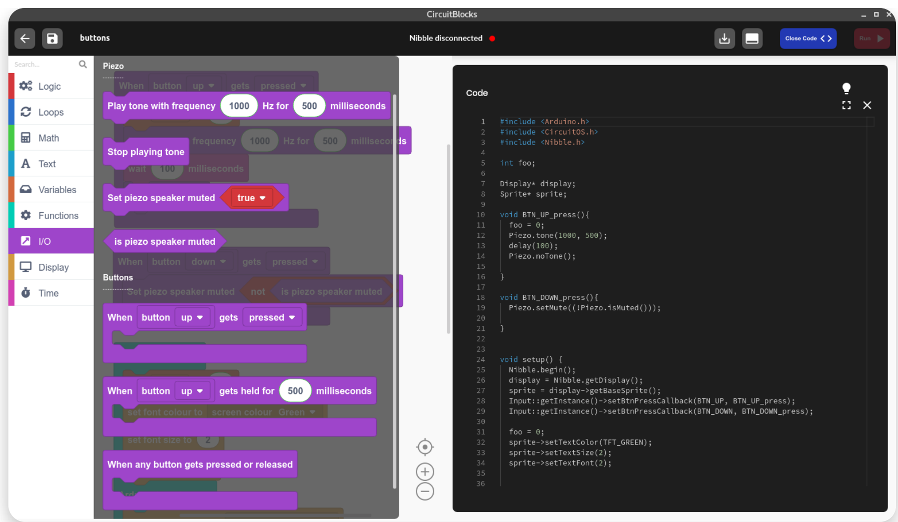
You can also create your own functions which can then be inserted as one of the main parts of your program.




## Input/Output

Everything regarding Nibble's buttons and audio is located here.

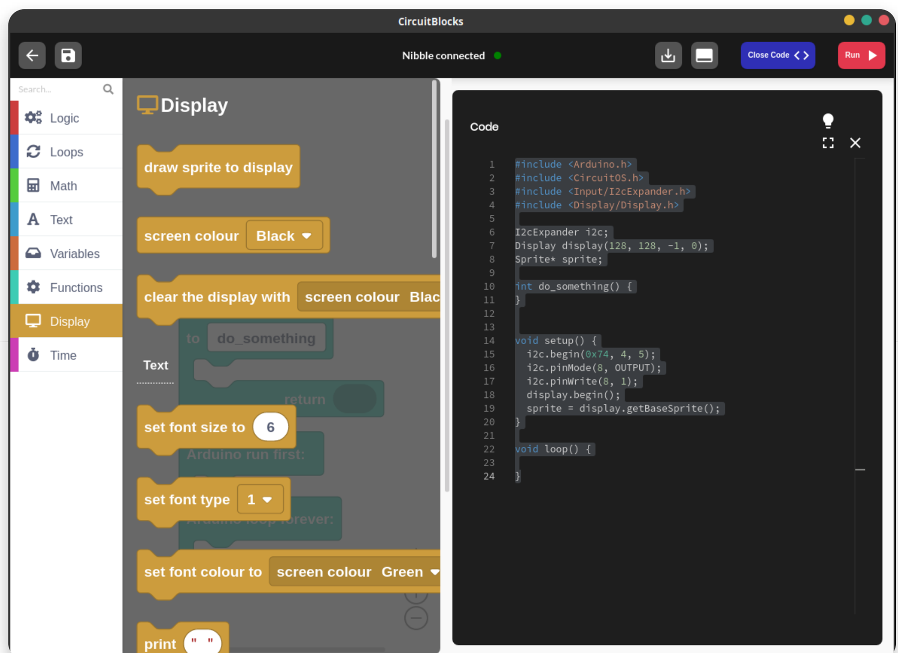
Not a lot of functions, but when used right they can do wonders!




## Display

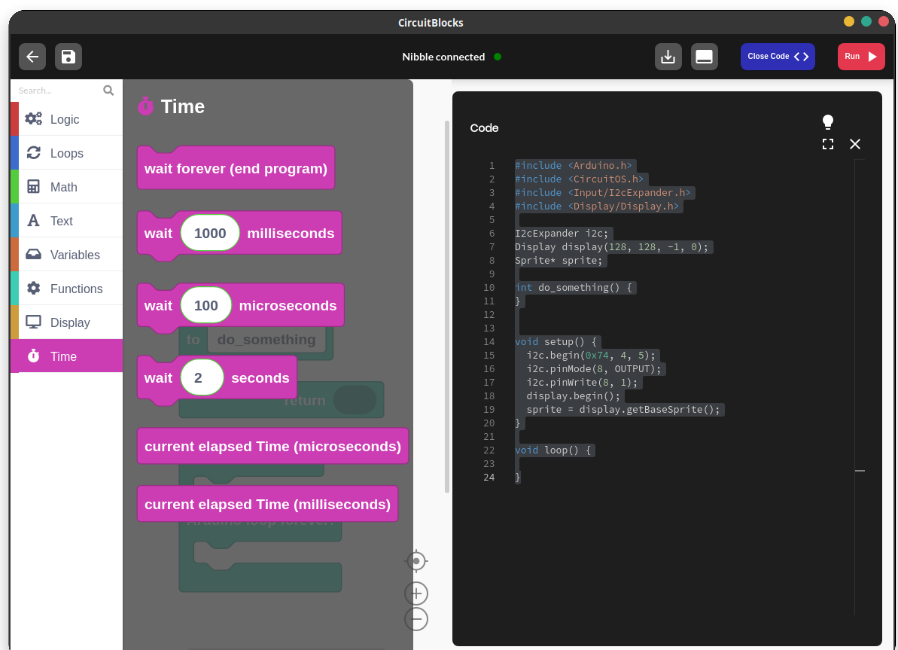
Well, all these blocks are really not important if you don't see anything on the screen!

Here is where all the magic translates to those colored pixels. You can create so much through these blocks.




## Time

Delays, timers, and other time-related stuff, great for creating cool animations and video games.




## Search bar

There is also a **search bar** above all function sections to ease the search for that one specific block you just can't seem to find (where is that PRINT???).

Just type in whatever comes to your mind and all blocks that have anything to do with the written word will be shown on the right-hand side.

Now, you really can't say that it's impossible to find something.

You've learned everything about the blocks!

It's time to move on to the next lesson...

# Changing screen color

Now that you're accustomed to every type of block, it's time to learn how to use them!

There will be a series of small examples where you'll be able to get an understanding of how the Nibble library works.

Each example will show different functionalities. In the end, we'll just bring all of that together to create a cool app!

**Let's begin!**

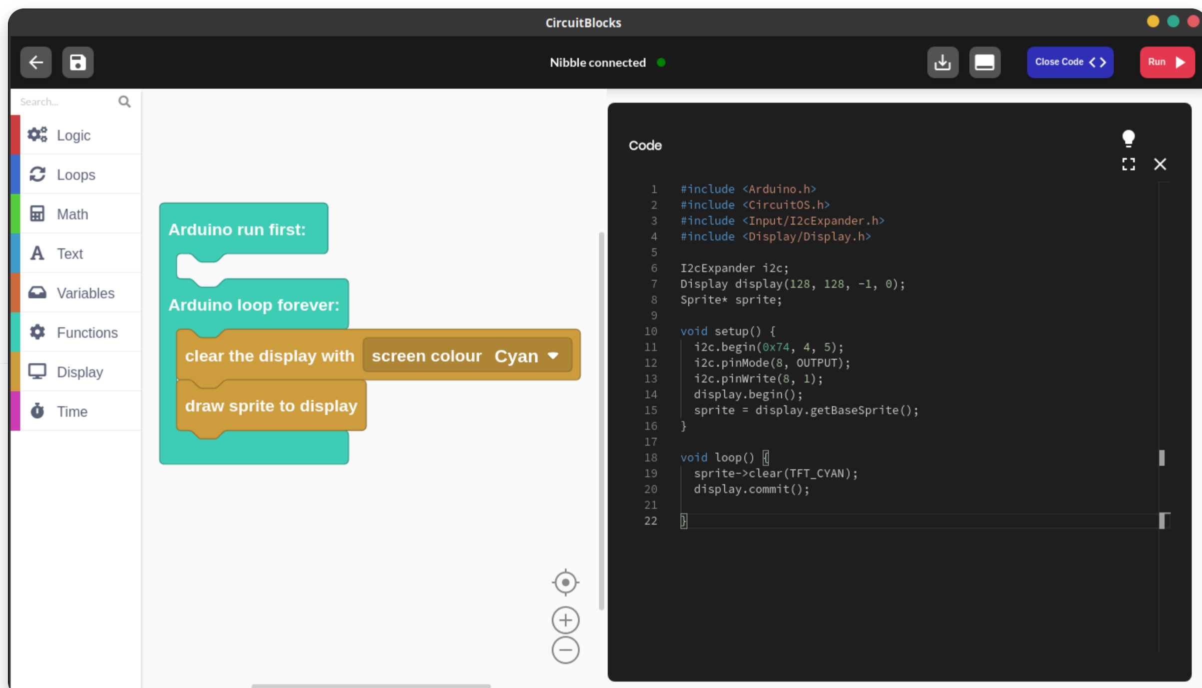
## Example #1

Let's kick things off as simple as possible.

The main component on Nibble is definitely the screen since working without one would be pretty much impossible.

What we want to do as a test is to change the screen color of our program.

The default screen color is **black**, or as it is known in our library, **TFT_BLACK**. All colors from this
library are labeled with **TFT_** prefix because they are made to be used on TFT screens.




Now let's move to the code generated by the blocks.

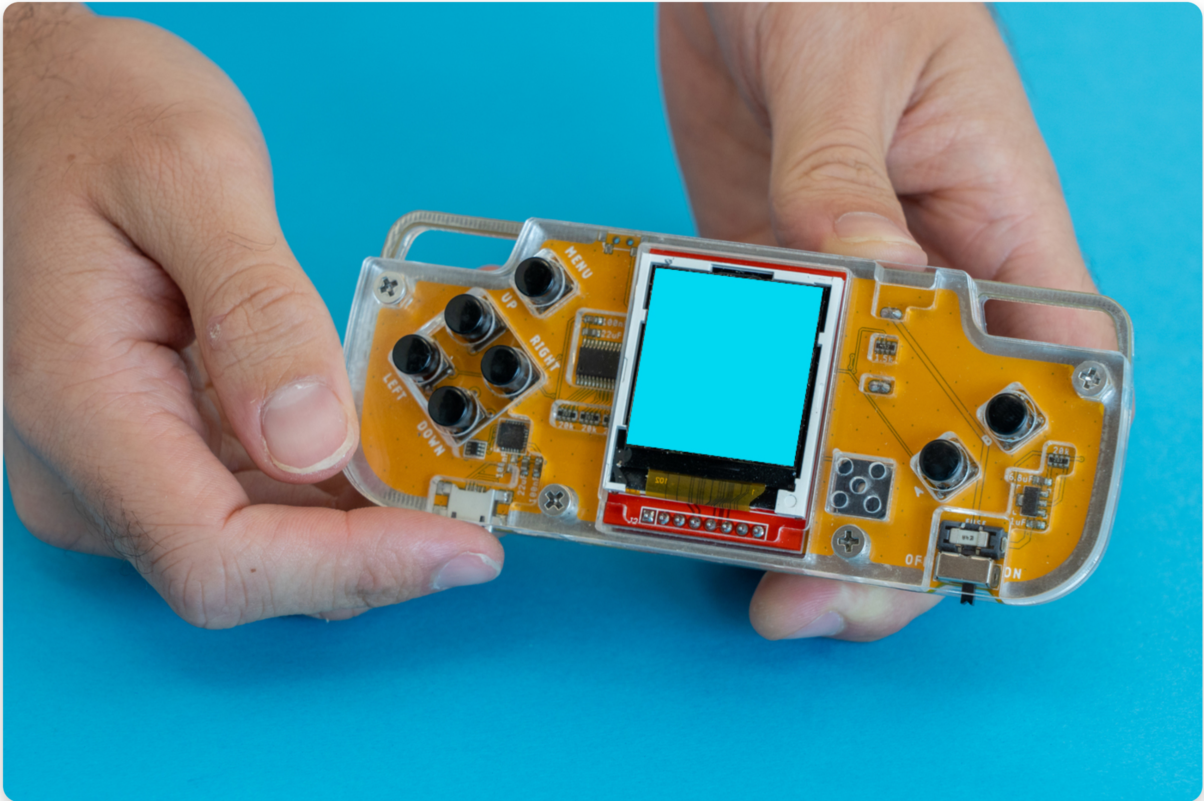
Since we want to turn our screen **cyan**, we are calling **sprite->clear()** function from the **Display section**, just with a different parameter.

Now that we've made sure everything is set, we can run the program. It will first be **compiled and then uploaded** to the console.

Ringo's screen should now be cyan. It's really as simple as that.

This is just a beginning but you can imagine that the possibilities of this screen

are countless.



**Colors**

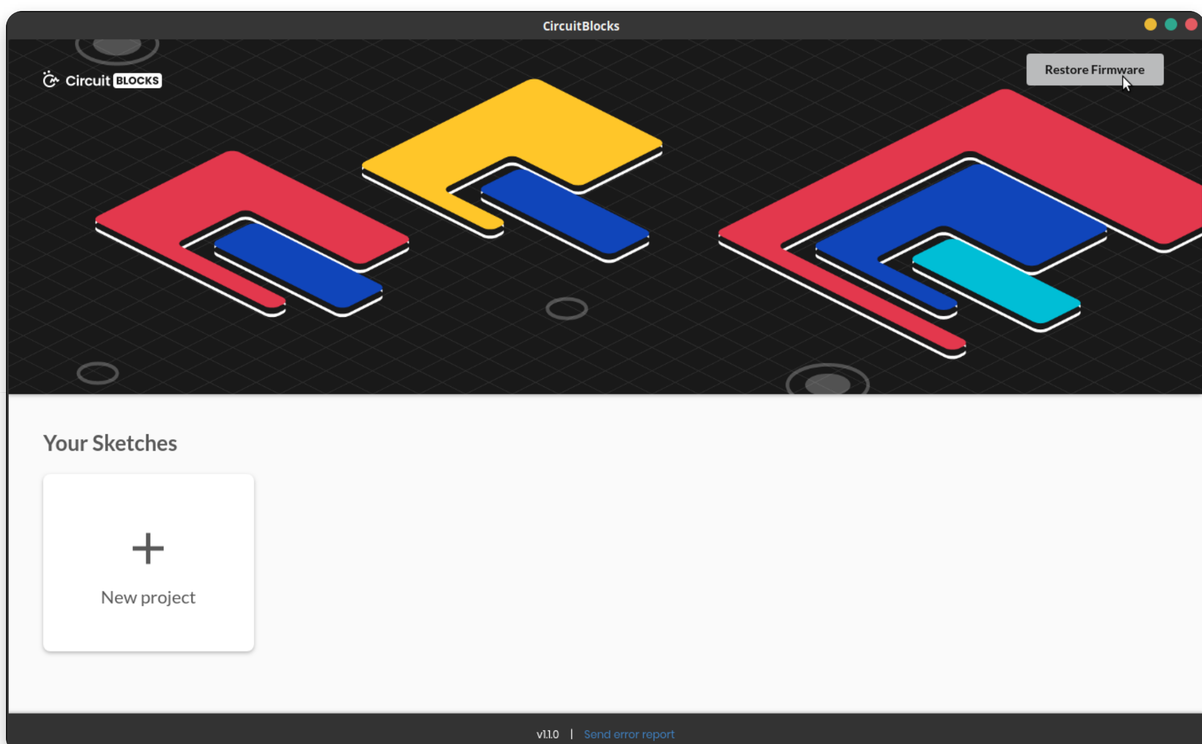
The full list of available colors can be found in the Display section

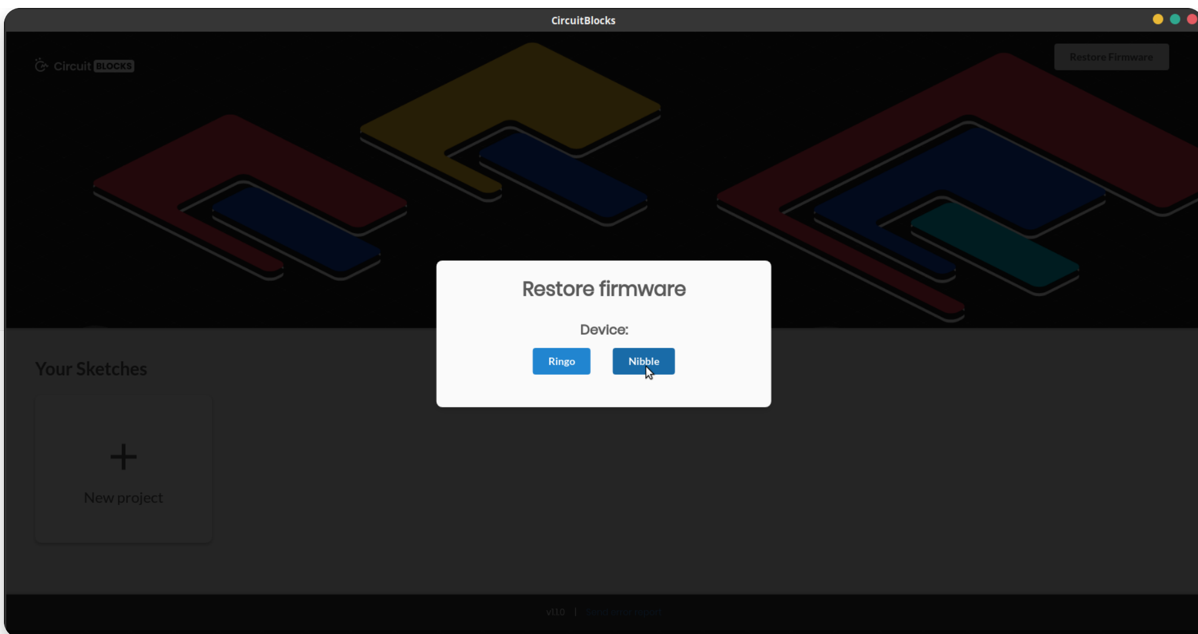
## Restoring defaults

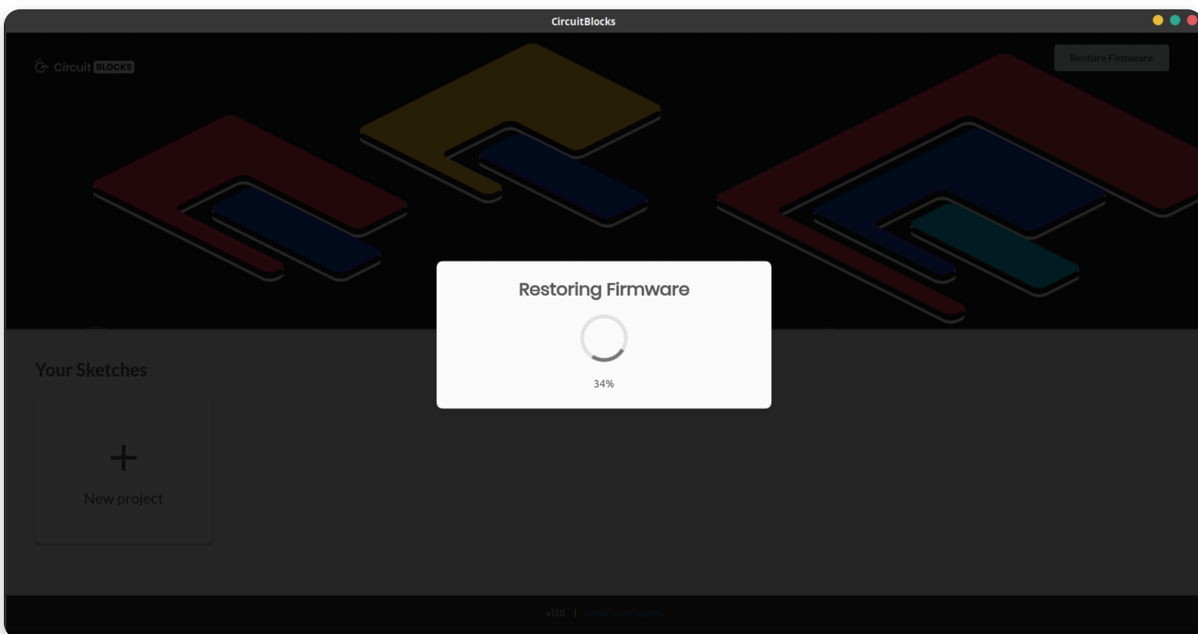
Now, what if we want to stop making cool things and just use our phone? Well, it takes only a couple of clicks to get everything back to normal.

**Anytime you want to restore the original Nibble firmware to your phone, you can do it by pressing the 'Restore firmware' button on the main menu and choose 'Nibble'.**

**Just don't forget to save the project before exiting!**



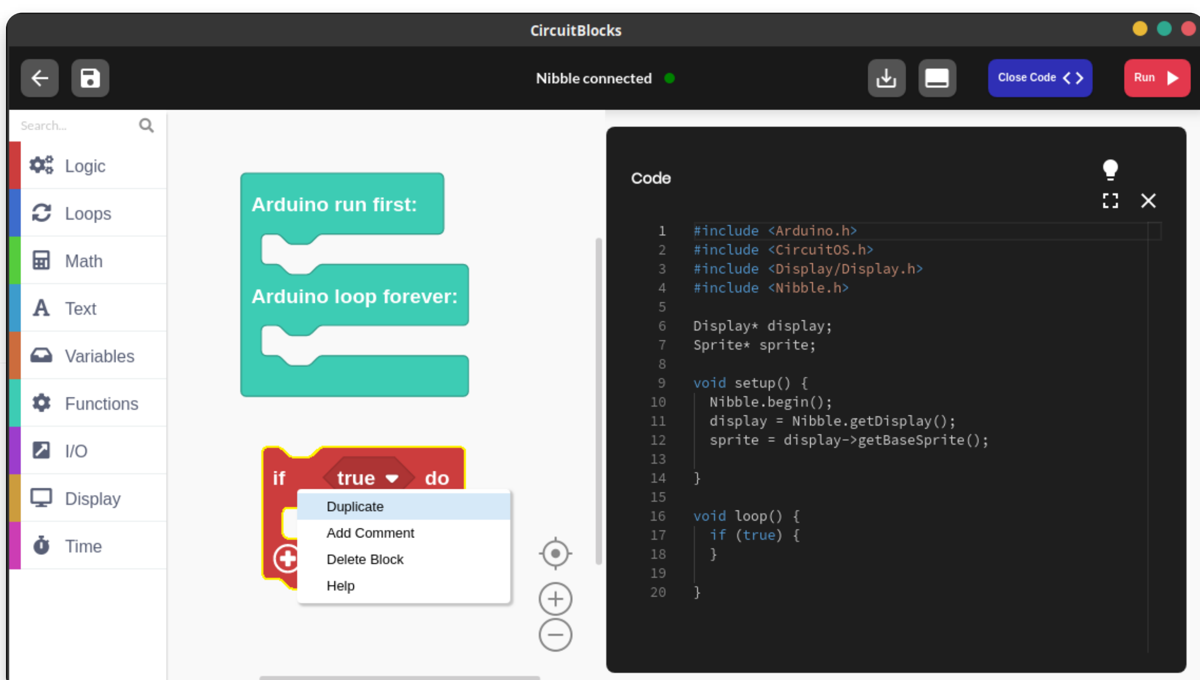



Now that we've covered the basics, let's head out to something a little bit more advanced.

# Controlling the display

## Quick commands




Before we go onto the next step, there is a little trick that might help you when

programming.

Press the **right-click** on your mouse on one of the blocks to open **a quick action menu** to easily **duplicate blocks, add commnets, delete blocks, or seek for help.**

You can also do some of these commands by pressing other buttons on your keyboard.

To delete a block, press **DEL button on Windows/Linux** or **DELETE on MacOS.**

To copy a block, press **Ctrl+C on Windows/Linux** or **Cmd+C on MacOS.**

To copy a block, press **Ctrl+X on Windows/Linux** or **Cmd+X on MacOS.**

To paste a block, press **Ctrl+V on Windows/Linux** or **Cmd+V on MacOS.**

# Scrolling colors

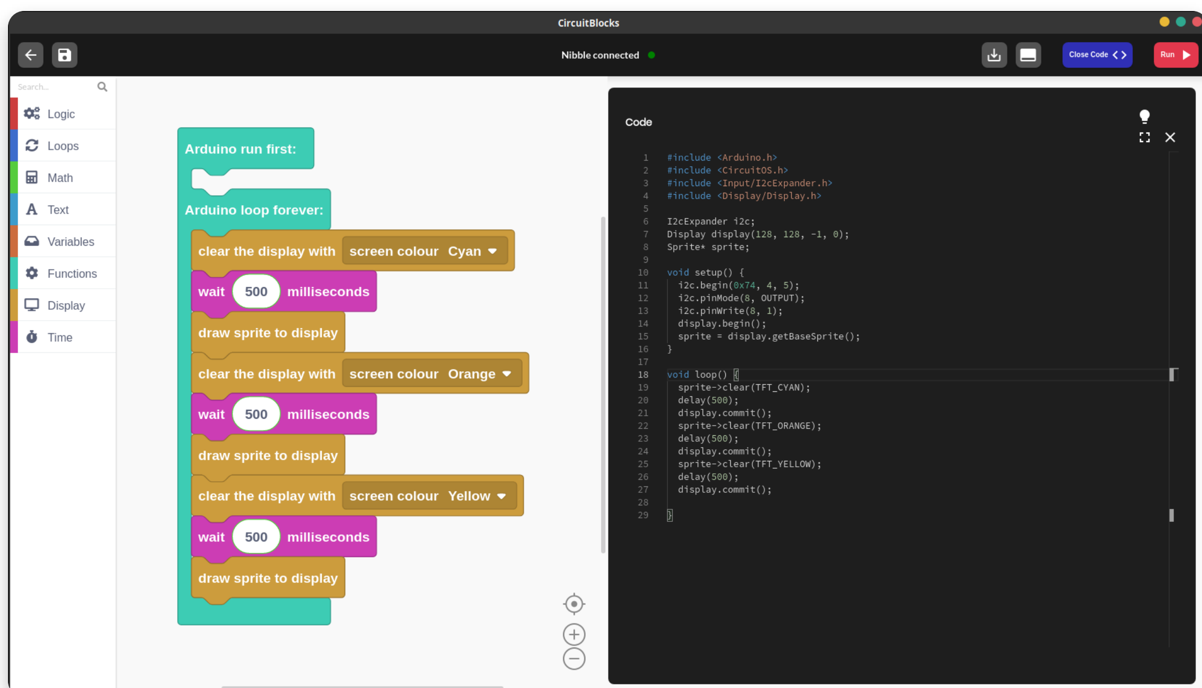
Now we're going to use the **loop()** part of the code.

See that **display.commit() function**? It's probably the most important function of all and it's located under the **display section.**

Its mission is to read everything that has happened between the last time it was called and now and to transfer those changes to the hardware.

For example, sprite->clear() function would not change anything if the display.commit() isn't called.

That function says **'draw sprite to display'** in the blocks!




We've also used another new block called **wait()**.

**It translates to the delay() function which stops everything for a certain amount of milliseconds.**

The number value block can be found in the Math section and its value can be changed to suit our needs.

Since the loop() function is really fast, sometimes we need this delay to actually see what is going on on the screen.

Using delay when **expecting a fast and responsive program**, however, **is not the way to go**,

especially when we're using buttons. More about that in the next lesson.

What we're actually doing in this program is alternating the screen color between three values.

Firstly we change it to cyan, then wait 500 milliseconds or 0.5 seconds.

Then, we change it to orange and wait another 500 milliseconds.

Lastly, we change it to yellow, wait 500 milliseconds, and return back to the beginning of the loop, where the screen is again changed to cyan.

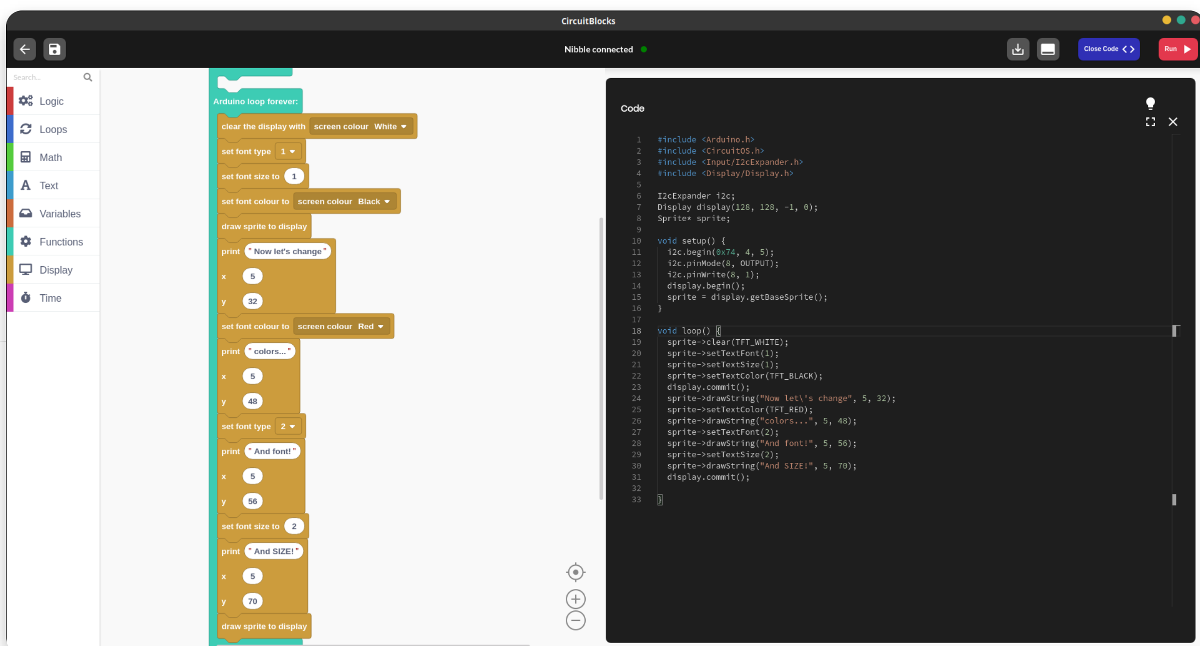
**Notice that we're calling the display.commit() function after each color change in order to transfer that color to the screen.**

# Writing out some text

Now that we know how to change the background color, it's time to start writing something on that canvas.

**Nibble library offers three different fonts that can be re-sized and re-colored as we like**.

Functions for that are rather simple and should not be a problem to understand for anyone.




With this program, we're writing some words out on the screen in **different fonts, sizes, and colors.**

The font type is easily changed with the **drop-down menu** of the set font type block. The font size, however, **multiplies the size of the selected font** by the desired value. **By default, both of those values are 1,** so setting them at the beginning of the program to that same value actually made no difference.

It's important to specify font type and to put the 'set font type' block inside the program, otherwise the program might crash!

Later both of those values have been changed. In the last example, we're printing out two words **"And SIZE!"** in a different font that is **twice the size.**

The **print()** function here is something that we're going to use quite often. The

first empty slot is a string of characters that are meant to be written out.

You can find an empty one in the **Text section**.

The second and third slot are the location of the first letter. **Setting both of those values to 0 will print out words in the upper left corner of the screen.**

With these numbers, we're actually referencing **pixel locations**. **Our screen is 128x128 pixels** and we can manipulate each and every one of them.

**Everything that we write/draw between the x values of 0 and 127 and y values of 0 and 127 will be visible on the screen.**

If any of those values is out of that range, we will not see the component.

## Loading...

## Loading...

## Loading...

## Loading...

## Something more

# Adding a bit of code

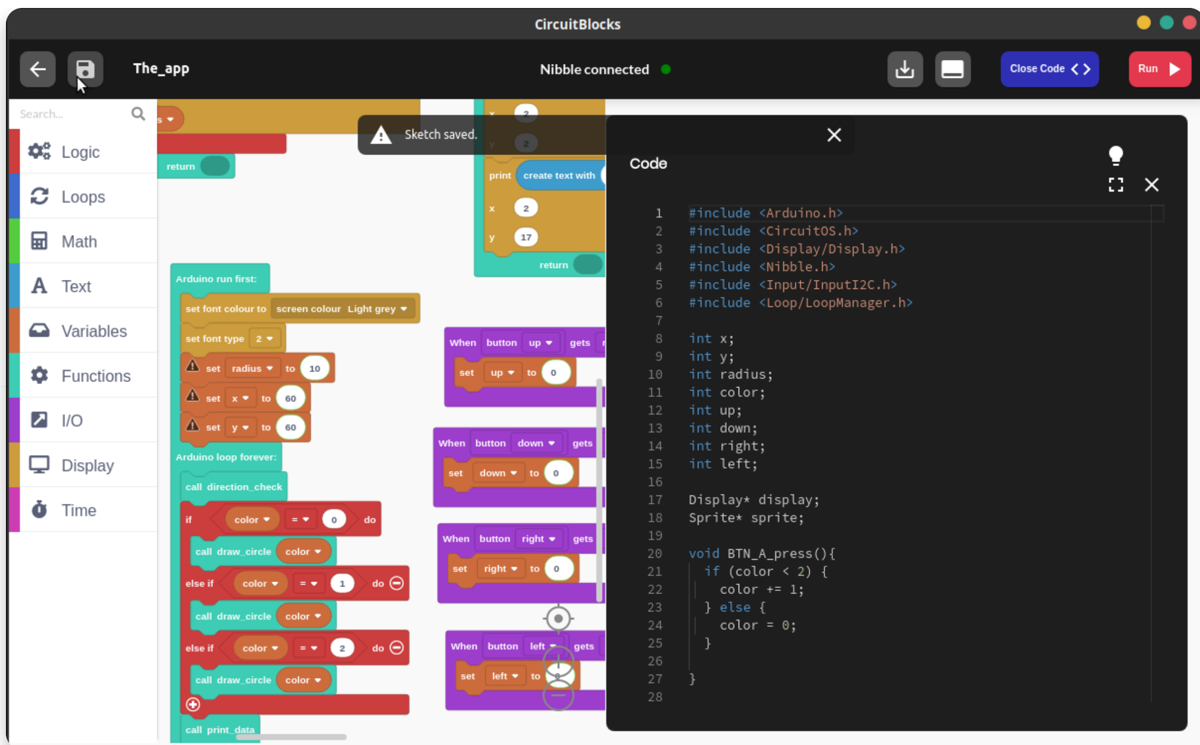
Now that our game has been finished we can notice one thing - **there is no sound!**

There is a small **buzzer** on the device that is able to produce many different **beeps, buzzes, and other crazy noises**.

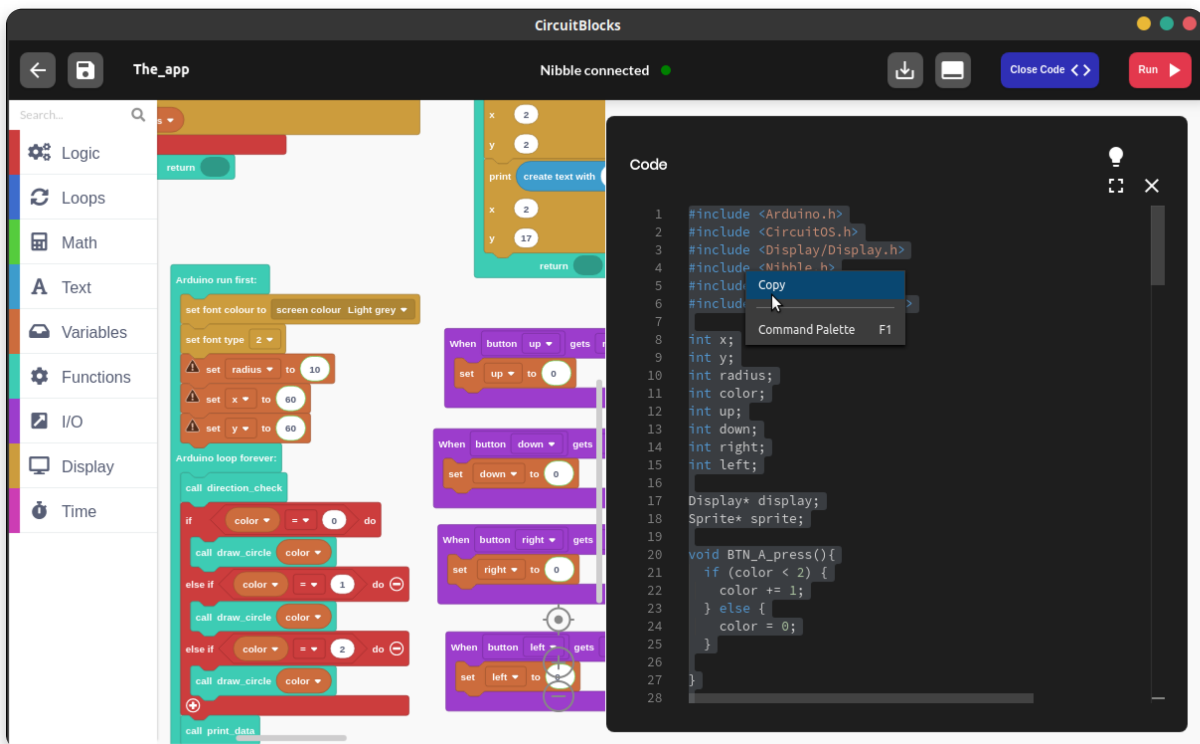
We could also use blocks to add it, but this way we'll explain how to do it by code.

This is the perfect time to copy the entire game code from the right side of the screen and to create a new project!

**First, remebmer to save the project and name it accordingly.**

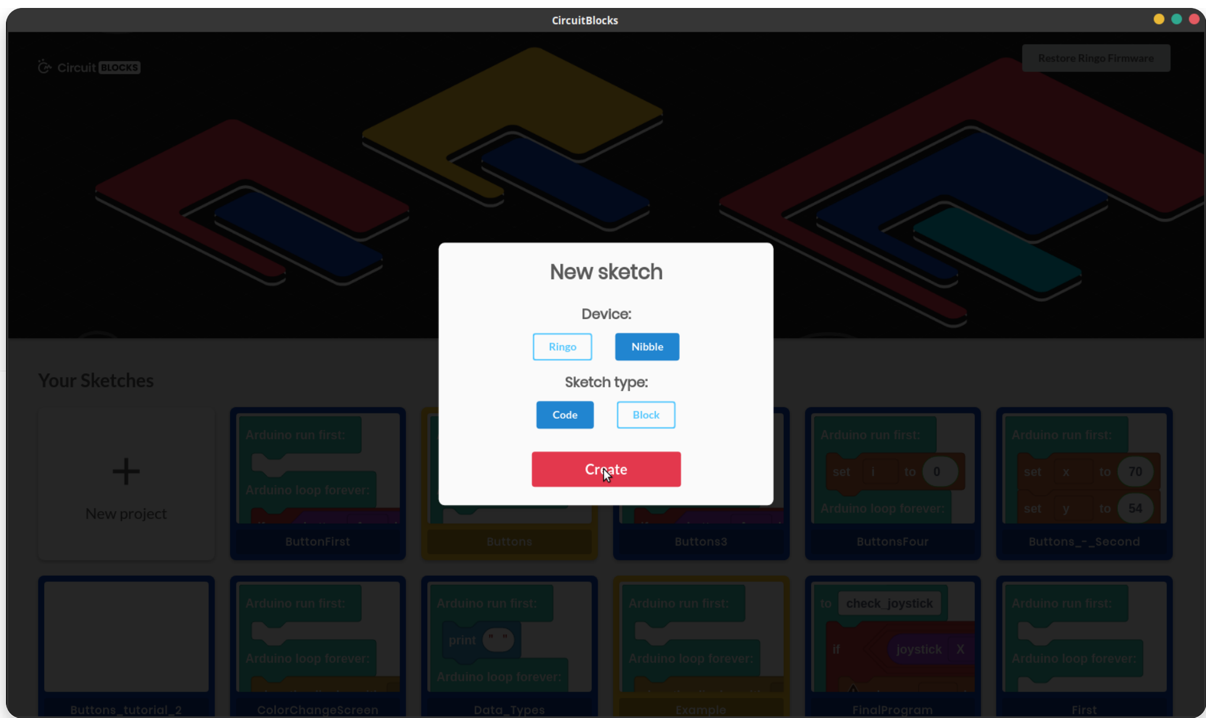



Then, copy the entire code by **selecting it, pressing the right mouse button, and clicking 'copy' or by pressing Ctrl+C/Cmd+C.**

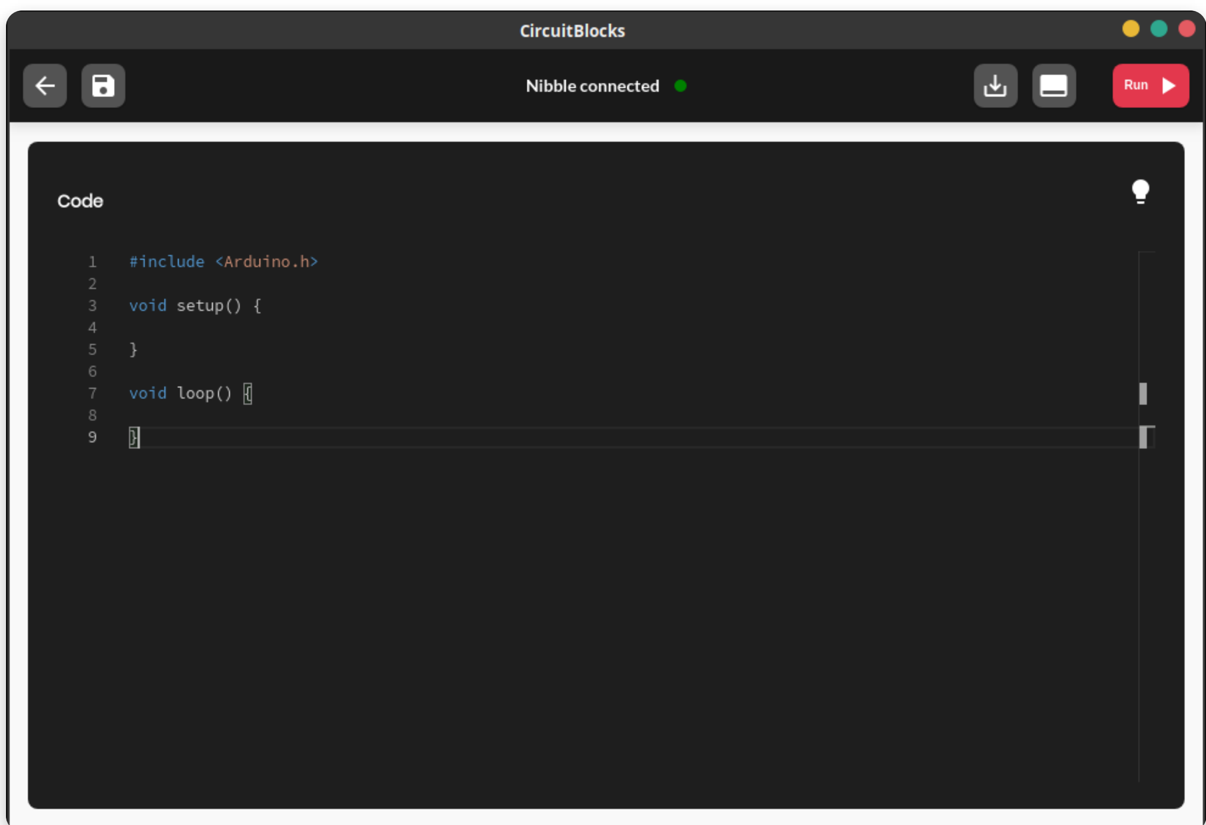



Go back to the main menu and open a new project.

Make sure to select **Nibble and code for your project type.**




Your screen should look something like this.




Delete the code that is inside and **paste the copied code on an empty screen.**

When you upload the code by clicking **'Run',** your Nibble should have the exactly same game as it did before!

Now, let's add the sound!

First thing we need to do is add a sound library.

The library we use is called **Piezo** and you should add the following line **anywhere among the #include section**.

ARDUINO

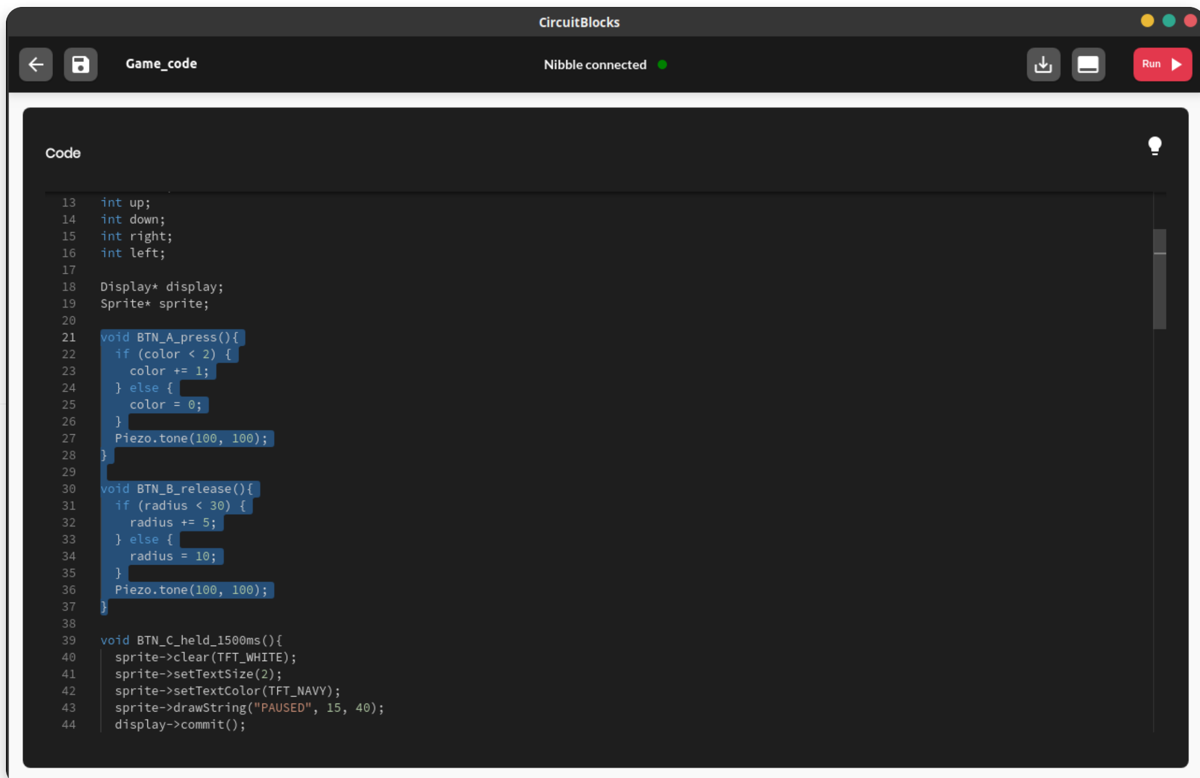
```
1 #include "Audio/Piezo.h"
```

Now, let's add a bit of tone wheneve we press buttons **A or B**.

The function we're looking for is **Piezo.tone()** which has two numbers inside the brackets.

The **first number determines the frequency of the tone**, while the second number determines **the length of the tone**.

You can change the numbers up a bit to get different sounds!

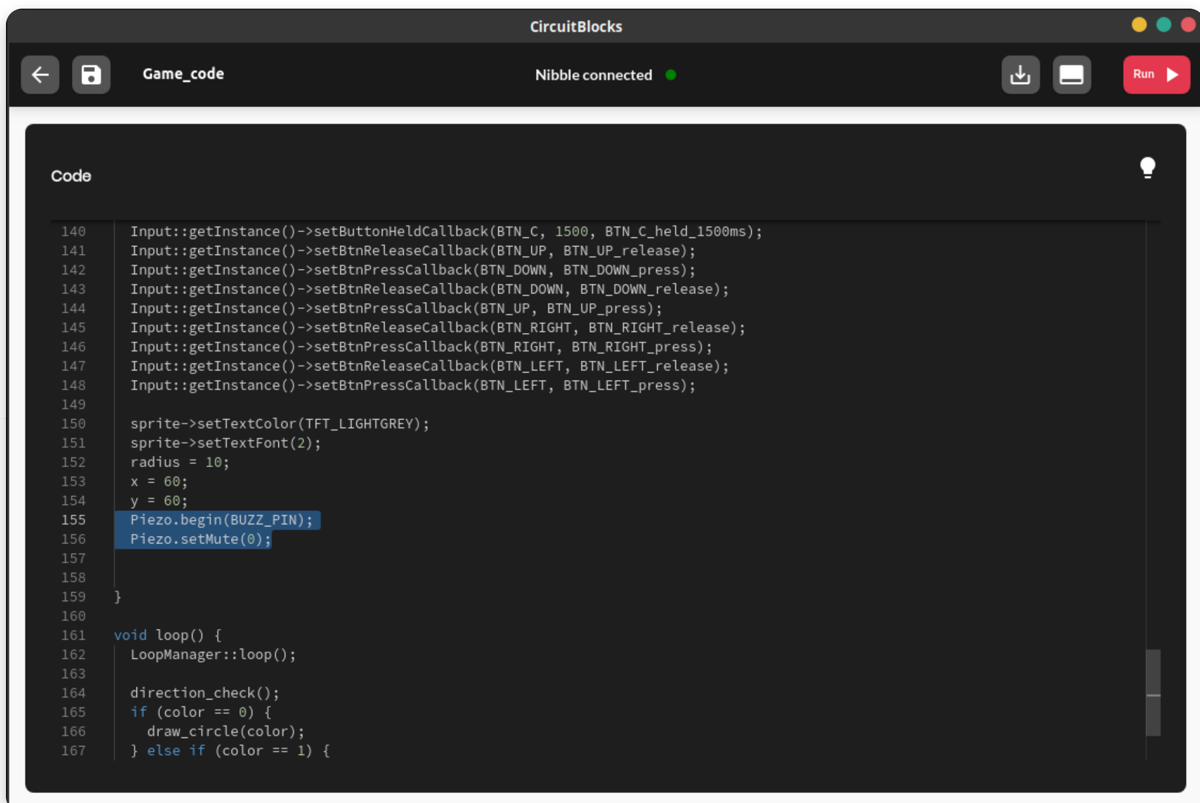



One final thing before finishing up - turning the sound on!

Find the main **void setup()** function and put the following line at the bottom of it.

**ARDUINO**

```
1 Piezo.begin(BUZZ_PIN);
2 Piezo.setMute(0);
```




Now re-upload the game by pressing the **red 'Run' button** and here it is - you've just added some sounds to your game!

If you wish, you can add even more sounds to make the game even cooler!

# Game ideas

On **CircuitMess GitHub page** you can find many more games for not only the Nibble, but for other consoles as well.

If you study these codes, you will learn how are these games written, which kind of methods are used for different actions, and most importantly - get some ideas for you own new games!

You can also scroll many different forums online, like our CircuitMess Community forum, and exchange knowledge and ideas with people like you all over the world!

Only the sky is the limit - now get to work and code some games!

## Loading...