

Nibbleコーディング

最初のステップ

はじめに

インストール

CircuitBlocksのチュートリアルへようこそ。

ご存じない方のために説明すると、**CircuitBlocks**は**Scratch**のような言語（ブロックタイプのビジュアルプログラミング言語）の統合開発環境（IDE：Integrated development environment）で、簡単かつ効率的にプロジェクトを作成し、**Nibble**コンソール（Nibbleゲーム機）にアップロードすることができます。

このチュートリアルはいくつかの章に分かれており、それぞれが IDE の重要な側面の1つを表しています。

注：今後、**CircuitBlock**は**CB**と表記します

インストール

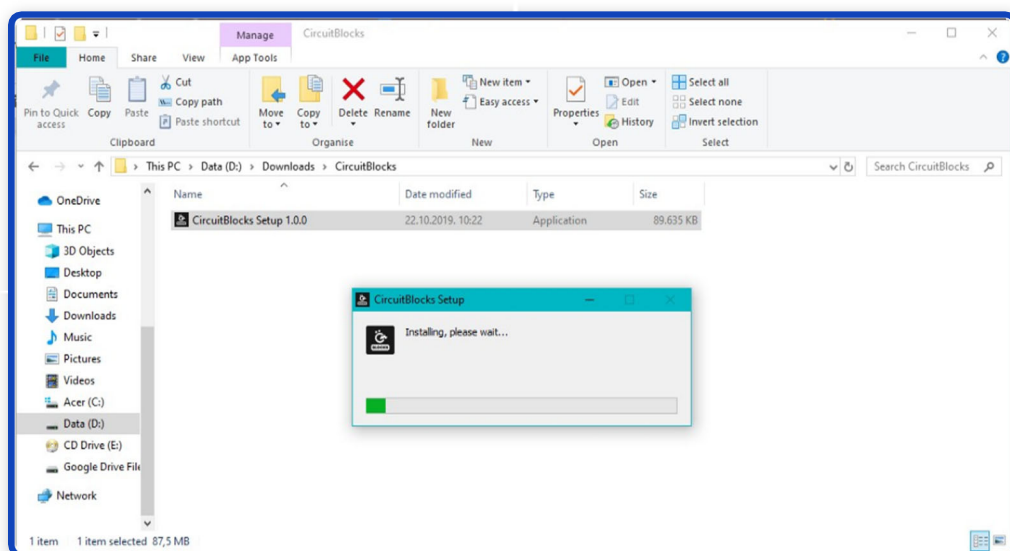
CBは主要なプラットフォームでサポートされています。

インストール方法はとても簡単です。ファイルをダウンロードし、お好みのプラットフォームで他のプログラムと同じようにインストールするだけです。

ダウンロードページで、あなたのメールアドレスとパスワードを入れてsign upしてください。

Windows（画面は英語版です）

- [CircuitBlocksのダウンロードページ](#)へ移動します。
- 最新バージョンの *Windows.exe (ex. CircuitBlocks-1.1.0- Windows.exe) をダウンロードします。バージョンが 32 または 64 かどうかを確認します。PCの [設定] から [システム] をクリックし、[バージョン情報] で、システムの種類を確認します。
- ファイルをダブルクリックして、実行可能ファイルを実行します。
- CBは自動的にインストールされ、新しいデスクトップ・ショートカットが作成されます。



あなたのPCは危険ではありません



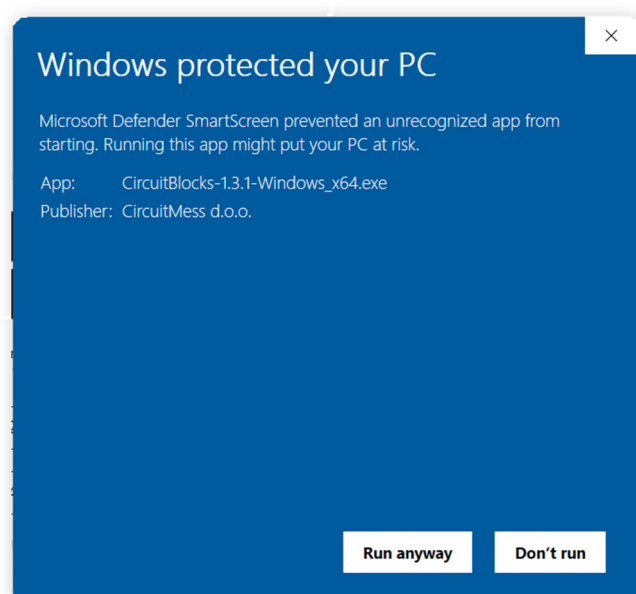
CircuitBlocksをインストールしようとする時、PCが危険であるという通知が表示される可能性があります。これはCircuitBlocksが安全に実行できるかどうかに関係なく発生するものですので、ご安心ください。この通知に対する対処方法は、以下の手順をご覧ください。

CircuitBlockをPCにインストールしようすると、右図のようなメッセージが表示されることがあります。プログラムを安全にダウンロードして実行できるにもかかわらず、Windowsが脅威を報告します。

[More info (詳細情報)] オプションをクリックしてインストールを続行してください。

[More info (詳細情報)] オプションをクリックすると、ウィンドウの下部に [Run Anyway (実行)] オプションが表示されます。

[Run Anyway (実行)] をクリックして続行します。



Linux

LinuxにCBをインストールするには、2つの方法があります。

インストール:

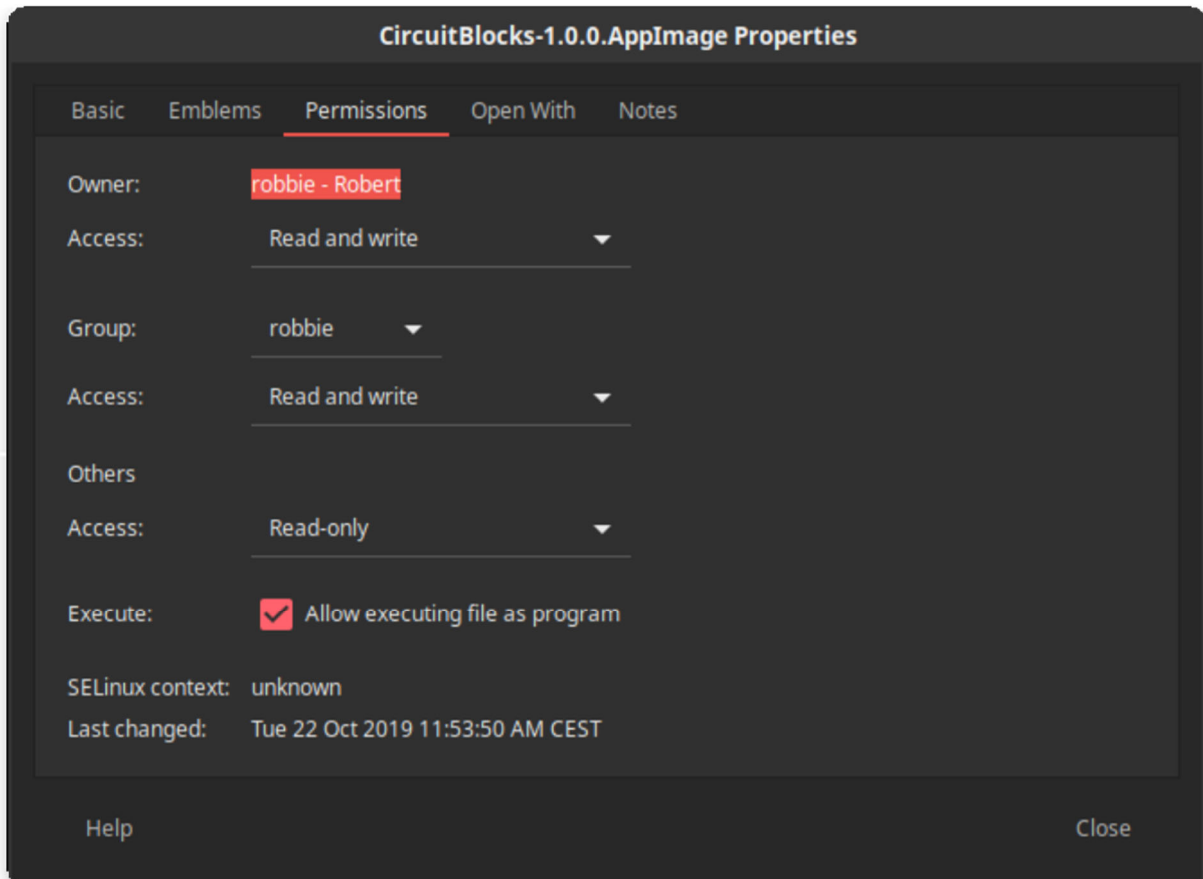
- [CircuitBlocksのダウンロードページ](#)へ移動します。
- 最新バージョンの ***Linux_amd64.deb (ex. CircuitBlocks-1.0.1-Linux_amd64.deb)** をダウンロードします。
- ファイルをダブルクリックして、インストールを実行します (Ubuntu)。
- ターミナルに書き込む `sudo dpkg -i <path to the downloaded file .deb>` (他のLinuxディストリビューション)
- CBは自動的にインストールされ、デスクトップエントリーが作成されます。

スタンドアロン (Applmage):

- [CircuitBlocksのダウンロードページ](#)へ移動します。
- 最新バージョンの ***Linux.Applmage (ex. CircuitBlocks-1.0.1-**

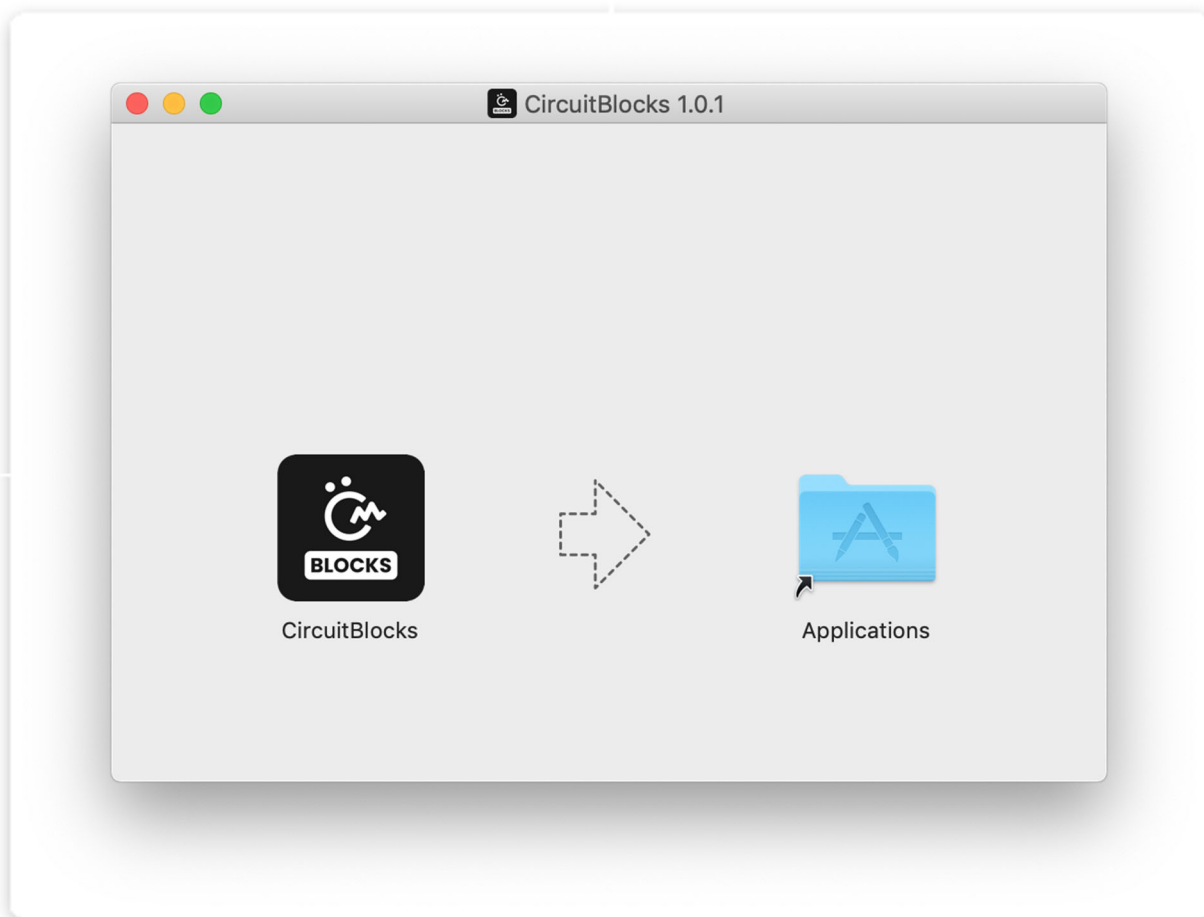
Linux.AppImage) をダウンロードします。

- ファイルを右クリックし、[Properties (プロパティ)] を選択します。
- [Permissions (アクセス権)] を開き、[Allow executing file as program (プログラムとしてファイルの実行を許可する)] にチェックを入れます。
- ファイルをダブルクリックすると、自動的にインストールが完了します。



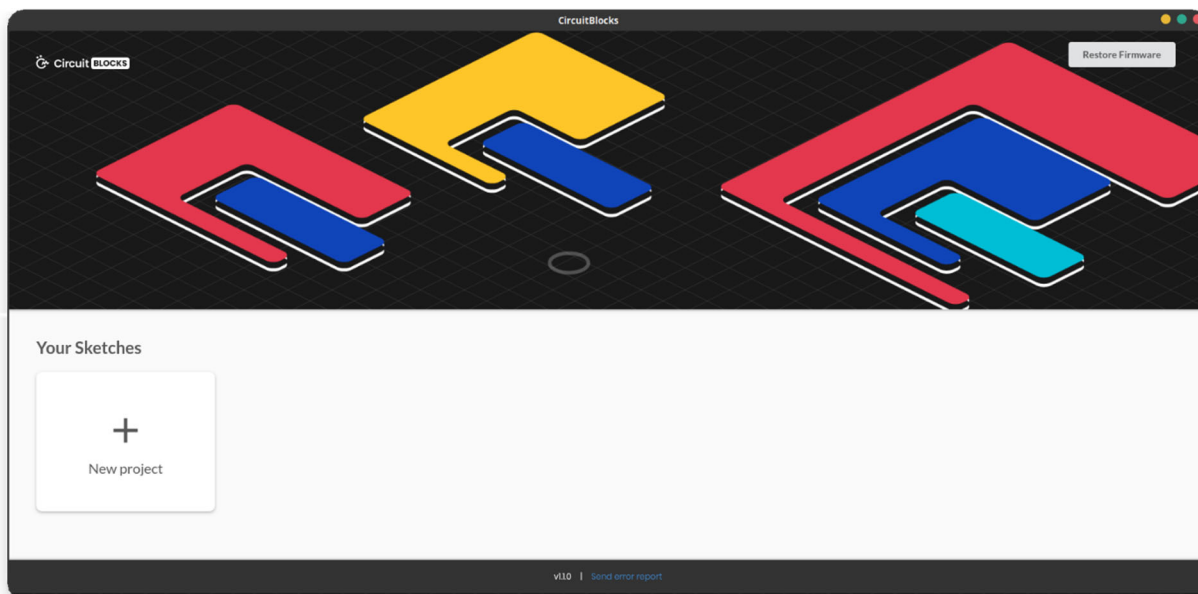
Mac OS

- [CircuitBlocksのダウンロードページ](#)へ移動します。
- 最新バージョンの***Mac.dmg (ex. CircuitBlocks-1.0.1-Mac.dmg)** をダウンロードします。
- ファイルを [Applications (アプリケーション)] フォルダに移動します。
- CBは自動的にインストールされます。



基本

インターフェース



CBを起動すると、このようなウィンドウが表示されます。（英語表示です）

新しいプロジェクト（スケッチ）の開始は、【New project (New sketch)】ボタンをクリックするだけです。

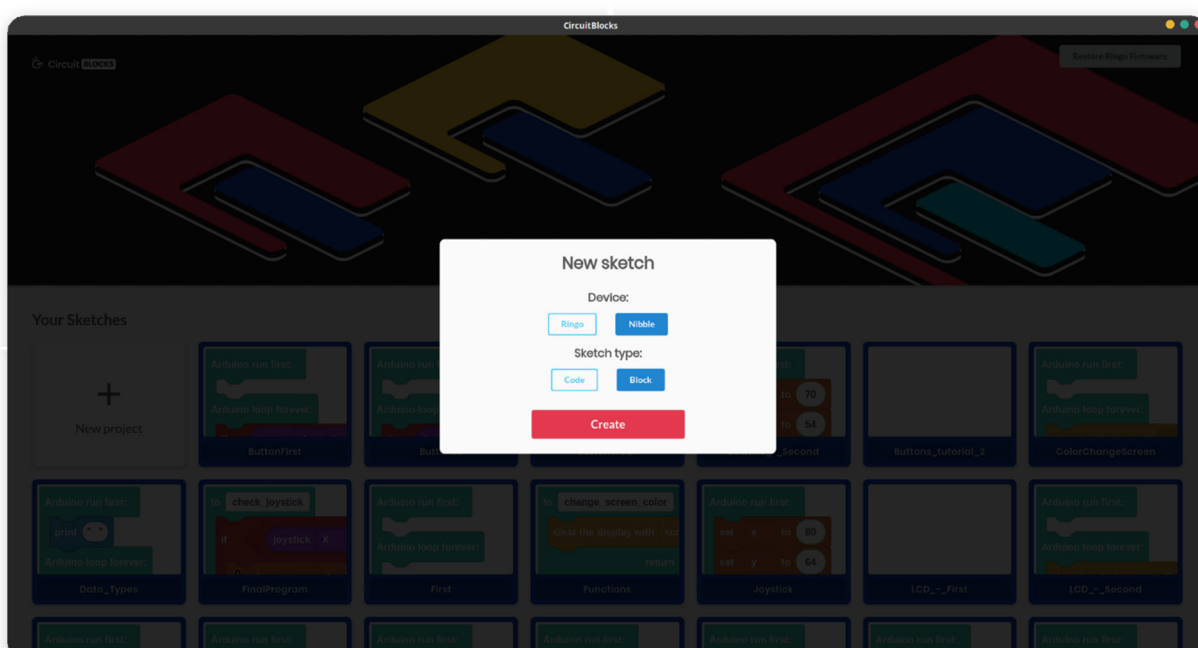
保存したスケッチはそのボタンのすぐ横に表示され、いつでもアクセスすることができます。

エラーが発生した場合は、メイン画面の下にある【Send error report（エラーレポートを送信）】を押してください。エラー番号をCircuitMessのコミュニティフォーラムで使用することで、チームメンバーがより効率的にお客様をサポートすることができます。

新しいスケッチの作成

新しいスケッチを開始すると、Ringoプロジェクト、Nibbleプロジェクト、コードプロジェクト、そしてブロックプロジェクトから選択するオプションが表示されます。

このチュートリアルではNibbleに焦点を当てるため、Nibbleプロジェクトを選択します。



CircuitBlocksにまったく新しいレイヤーを追加しました。ブロックを使う必要がなくなり、Arduino IDEと同じようにC/C++を使ってゲーム全体を作成できます。

ただし、ここではコードの記述ではなく、**ブロックの構築**に焦点を当てます。

また、ブロックを使用してゲームの一部を作成し、それをコーディングバージョンに移行して、ブザー音などのコードを追加することもできます。

これによりほとんどの場合、（ブロックを介した）より簡単なコーディング方法を使用できるようになり、最終的にはいくつかの詳細を追加するだけで完成させることができます。

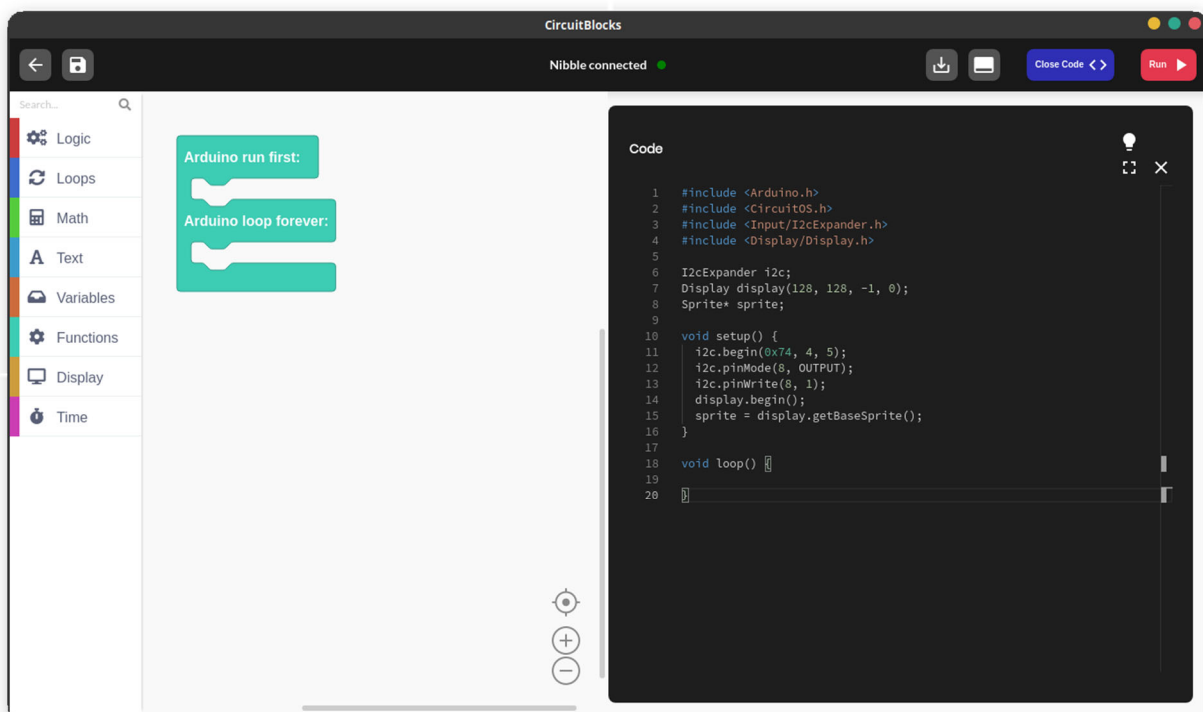


いつでもブロックバージョンからコードをコピーして、コーディングバージョンに切り替えることができます。ただし、コーディングバージョンからブロックバージョンに戻すことはできません。そのため、可能な限りブロックバージョンを使用することをお勧めします。

コードプロジェクトは、**Arduino IDE** で得られるものとほぼ同じです。作成したコードに基づいてプログラムを実行する **Arduino C/C++** です。

一方、ここではブロックプロジェクトが実際のプロジェクトです。そこで、ドラッグ&ドロップしたブロックからプログラムコードを生成できます。本物のScratchのような体験ができ、プログラミングのキャリアを始めたり、Nibbleライブラリの詳細を学習したりするすべての人に強くお勧めします。生成されたコードは、コードプロジェクトでコピーおよび変更できます。

プログラミングのスキルに関係なく、ブロック・プロジェクトを始めることをお勧めします！



これは、ほとんどの場合に使用する
メイン・インターフェースです。
画面上部にはツールバーがあり、ここから
プログラムの主な機能にアクセスする
ことができます。

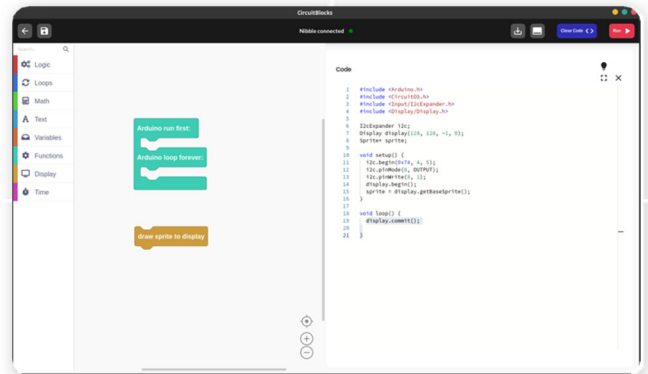
ブロック選択ツールは左側にあります。

画面中央は、ブロックを使ってコード
を「ドローイング」する場所です。

右側は、これらのドローイングをコード
に変換する場所です。これはVS Codeで
使われているものと同じコードエディタ
ですが、編集はできません。コードを編
集したい場合は、コードをコピーして、
新しいコードベースのプロジェクトに転
送する必要があります。

ダークモードが苦手な方は、コードエ
ディタの右上にある電球ボタンを押す
と、簡単に切り替えることができま
す。

ツールバー



主に7つのコンポーネントについて説明
します。

1. **メインメニューに戻る** : 保存せずにメインメニューに戻ります。
2. **保存/名前を付けて保存** : ファイルをデフォルトのスケッチディレクトリに保存します。
3. **Nibble connected (Nibble接続)**
インジケータ : コンソールが接続されているかどうかを表示します。
4. **バイナリへのエクスポート** : コンソールに直接アップロードできるコードの.bin ファイルを作成します。
5. **シリアルを開く** : シリアルポートを開きます。
6. **Close Code (コードを閉じる)** : コードエディタを閉じて、[ドローイング] 領域を拡張します。
7. **Run (実行)** : コードをコンパイルし、コンソールにアップロードします。



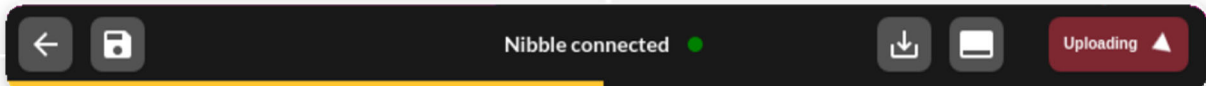
コンソールが接続されていない場合、赤い [Run] ボタンが灰色に変わります。

インジケータには「Nibble disconnected (Nibbleが切断されました)」と表示され、緑色の代わりに赤い点が表示されます。

作成したコードがアップロードされている間、ツールバーのすぐ下に進捗バーが表示されます。

これまでにコンパイル/アップロードされたデータの量が示されます。

50%になったらコンパイルが終了、100%になったらコンソールへのアップロードが完了したことを意味します。



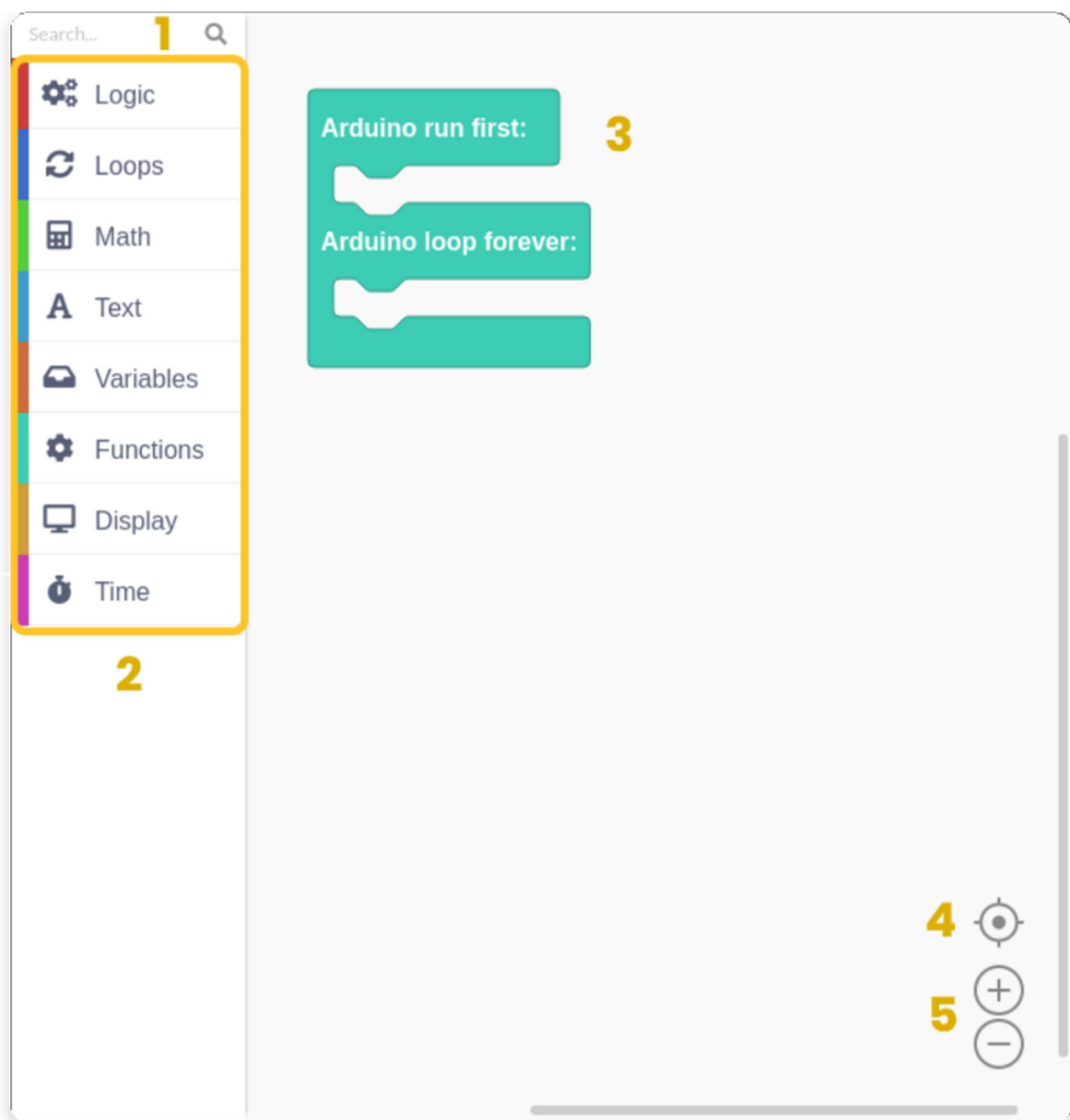
1. **メインコード画面**：ブロックがテキスト形式で表示される場所です。コードワードは色付きで、通常のテキストは白です。
2. **ライトモードの切り替え**：コードエディタの背景を黒と白の間で切り替えます。
3. **拡張**：コードエディタをウィンドウ全体に広げます。
4. **閉じる**：コードエディタを閉じます。ツールバーの [Close Code] ボタンと同じ機能です。

A screenshot of the code editor window. The title bar says 'Code'. The code is as follows:

```
1 #include <Arduino.h>
2 #include <CircuitOS.h>
3 #include <Input/I2cExpander.h>
4 #include <Display/Display.h>
5
6 I2cExpander i2c;
7 Display display(128, 128, -1, 0);
8 Sprite* sprite;
9
10 void setup() {
11     i2c.begin(0x74, 4, 5);
12     i2c.pinMode(8, OUTPUT);
13     i2c.pinWrite(8, 1);
14     display.begin();
15     sprite = display.getBaseSprite();
16 }
17
18 void loop() {
19     sprite->clear(TFT_WHITE);
20     display.commit();
21 }
22
```

On the right side of the editor, there are icons for a lightbulb (numbered 2), a refresh icon (numbered 3), and a close icon (numbered 4).

[ドローイングボード] は、IDEの中で最も複雑な部分です。すべての魔法が起きる場所です。大きく2つのセクションに分かれています。左側はブロックを選択するボードで、右側はブロックを配置するボードです。ブロックの種類ごとに色がついているので、すぐにわかります。



1. **検索バー**：動的な検索バーで、探しているコンポーネントを簡単に見つけることができます。
2. **コンポーネントセレクトター**：名前と色でカテゴリーに分けられます。
3. **ドローイングボード**：コンポーネントを一定の順序で配置し、プログラムを作成する場所です。
4. **センターアイコン**：ブロックをボードの中央に配置します。
5. **ズームボタン**：ボードを拡大、縮小します。

各コンポーネントについて詳しく説明し、それらをどのように組み合わせて使うか、いくつかの例を紹介します。

ブロックの使いかた

ブロックの種類

CBIには、合計9種類のブロックがあります。それぞれ色で表されます。すべてのブロックはコードに変換され、コンパイルされてコンソールにアップロードされます。これは、すべてのArduinoベースのプラットフォームと同様です。

各ブロックタイプを押すとセクションが開き、ブロックをドローイングエリアにドラッグ&ドロップできます。

また、**[More]** を押すと、あまり使われないブロックがさらに開きます。

Arduinoコードには、**void setup()** と **void loop()** という 2つの主要な関数があります。※() はゼロではなく、カッコ () を合わせたものです。

void setup() 関数に入るものは、すべて一度だけ実行されます。主にソフトウェアの起動、変数の初期化と宣言、および一度だけ実行する関数（例：ビデオゲームのイントロ画面）の実行に使用されます。

void loop() は、他のすべてが行われる場所です。内部のすべてのコードを何度も実行します（速度はデバイスによって異なりますが、超高速だと想像してみてください）。画面のリフレッシュレートにほぼ従い、それに応じてプログラムを実行させる必要があります。

配置したブロックはすべて、自動的に**void loop()**関数に入ります。

void setup()に何かを入れたい場合は、**Functions**からメインブロックをドラッグし、必要に応じてブロックを内部に配置する必要があります。これについては後で詳しく説明します。

楕円ブロック

楕円形のブロックは変数を表します。整数、文字列、その他の変数型（ブーリアン以外）のいずれであっても、同じ形状で認識することができます。

また、楕円形の大きなブロックは、整数値またはfloat値のどちらかの値を返します。

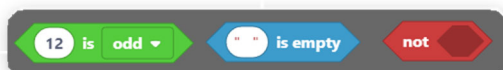


ブロックの中に丸い [穴] がある場合、変数を挿入できます。これは、**比較ブロック**や**アクションブロック**によく見られます。

三角ブロック

ブール型変数は三角形のブロックで表されます。

両方の変数 (true と false)、およびブール値を返す関数は、同じ形をしています。



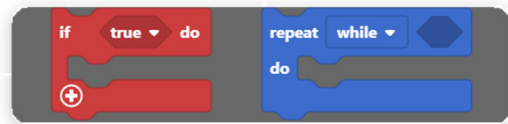
色に関係なく、これらのブロックはそれぞれ true または false のどちらかを返します。

ビルディングブロック

三角形の [穴] には、ブーリアンブロックを挿入する必要があります。

【MEMO】 ブール型とは
プログラミン言語などに用意されているデータ型の一つで、「真」(wuxh)と「偽」(idozh)の二種類の値だけを取りうるもの。

それ以外はすべてビルディングブロックです。これらは、戻り値を持たない関数です（*null*を返します）。楕円形ブロックも三角形ブロックもプログラムの一部として機能させるために、まずビルディングブロックの中に置かなければなりません。



これらは特定の [パズル] の形をしており、互いに積み重ねることができます。

メインビルディングブロックは、[Functions] セクションの中にあります。



これは、2つの主要なビルディングブロックのセクションを提供します。

Arduino run firstの中に置かれたものはすべてvoid setup()に入り、Arduino loop foreverの中に置かれたものはすべてvoid loop()に入ります。

ブロックの挿入

ここがメインパートです。

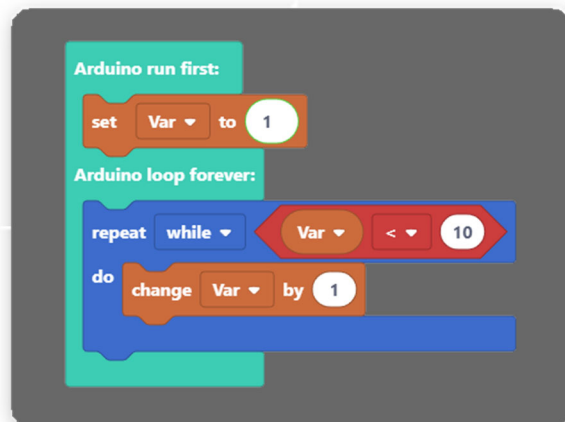
ブロックのようなIDEは、ブロックをつないで別のブロックの中に入れるのがポイントです。

これは、ドラッグ&ドロップで簡単にできます。

ここでは、変数Varを1に設定し、10より小さい間はその変数を増加させるというプログラムの例を示します。

プログラム終了時には、Varは10になります。

これは簡単な例であり、ブロックの構築については次の章でさらに説明します。



ブロックセクション



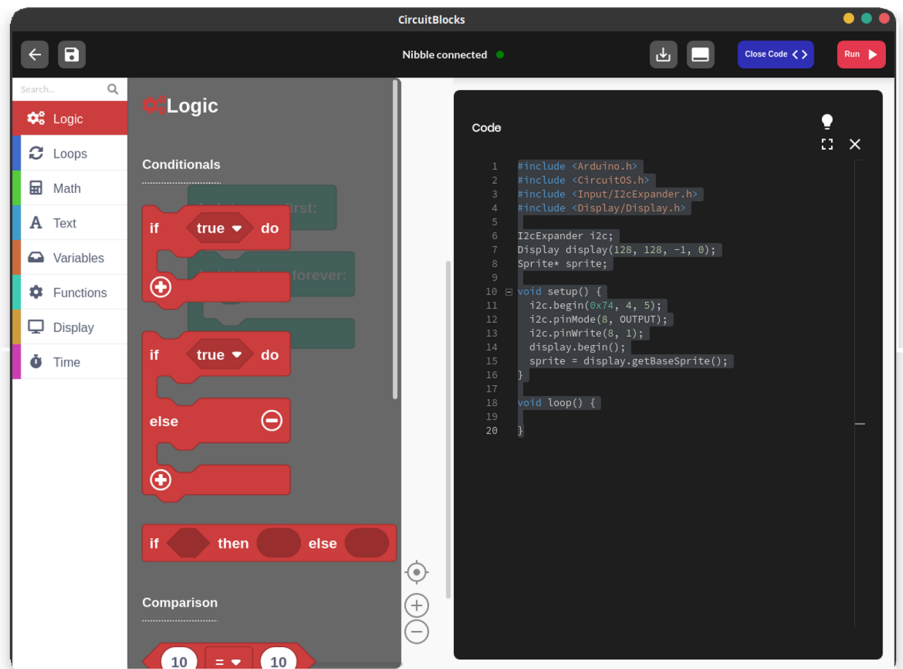
CircuitBlocksには、合計9つのセクションがあります。最大2回のクリックですべての項目が見つけられるように構成されています。

各セクションの説明はとてもわかりやすいのですが、全体のコンセプトを少しでも理解していただくために、すべてのセクションを紹介します。

一部のセクションには [More (その他)] メニューに追加のブロックがあり、あまり使われないけれども便利な機能を見つけることができます。

Logic (ロジック)

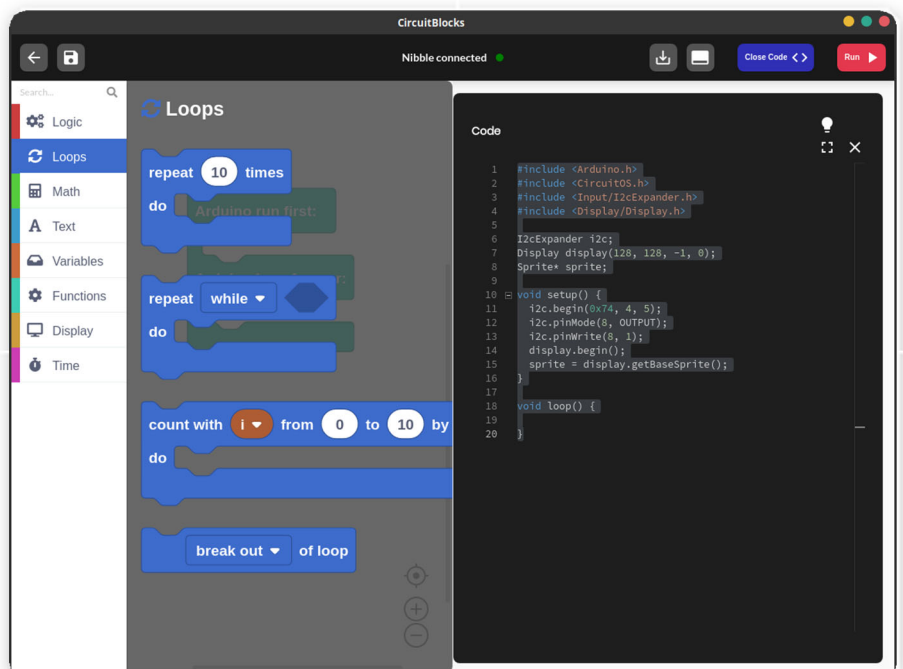
これは、すべてのコードのベースが配置されている場所です。すべての if、if-else、else 関数、比較、and/or、not、true/false、およびその他の論理演算子。



Loops (ループ)

ループとは、特定の時間、内部のすべての処理を繰り返し関数のことです。

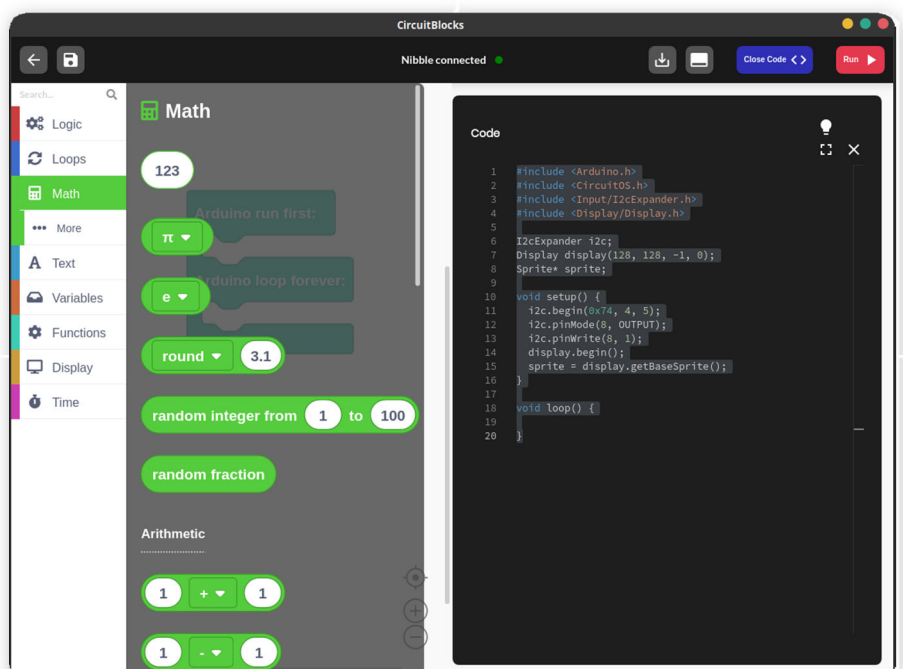
条件付きで、その条件が満たされる限り繰り返されるものと、あらかじめ決められた回数だけ繰り返されるものがあります。



Math (数学関数)

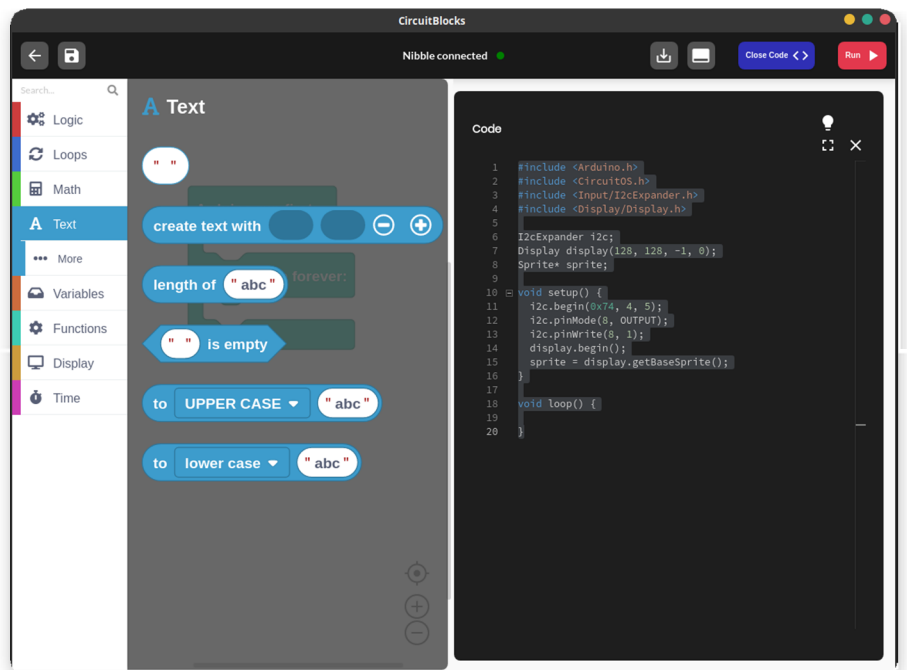
ほとんどの数学関数がここにあります。

基本的な演算から四捨五入、角度の操作まで。あなたの内なるインシュタインやピタゴラスを一瞬で呼び覚ますことができます。



Text (テキスト)

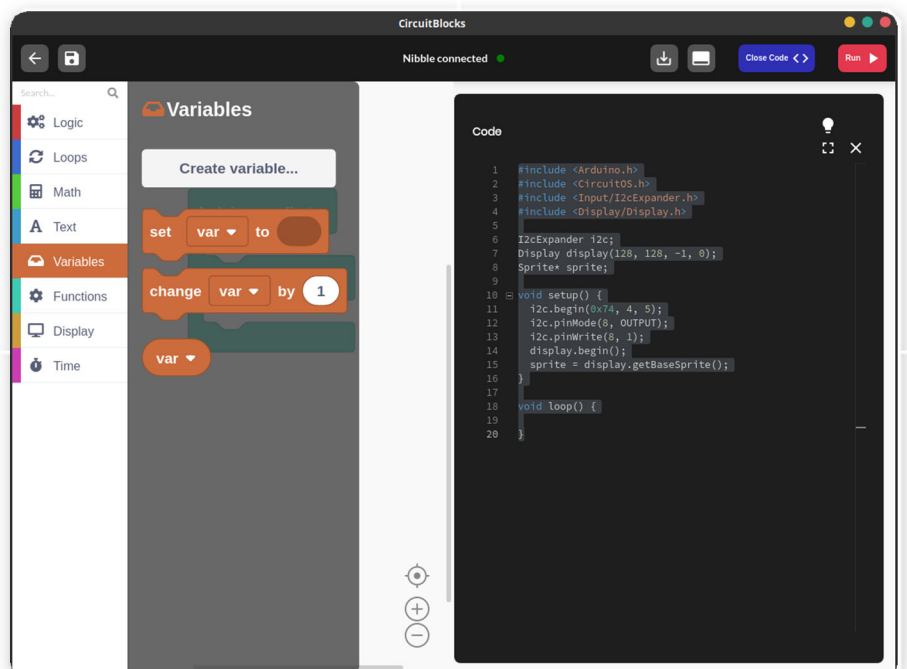
文字列、文字、文字列操作。
新しいテキストを作成し、プログラムに実装するのに最適な場所です。



Variables (変数)

任意の型の変数を作成し、その名前と希望する値を設定します。

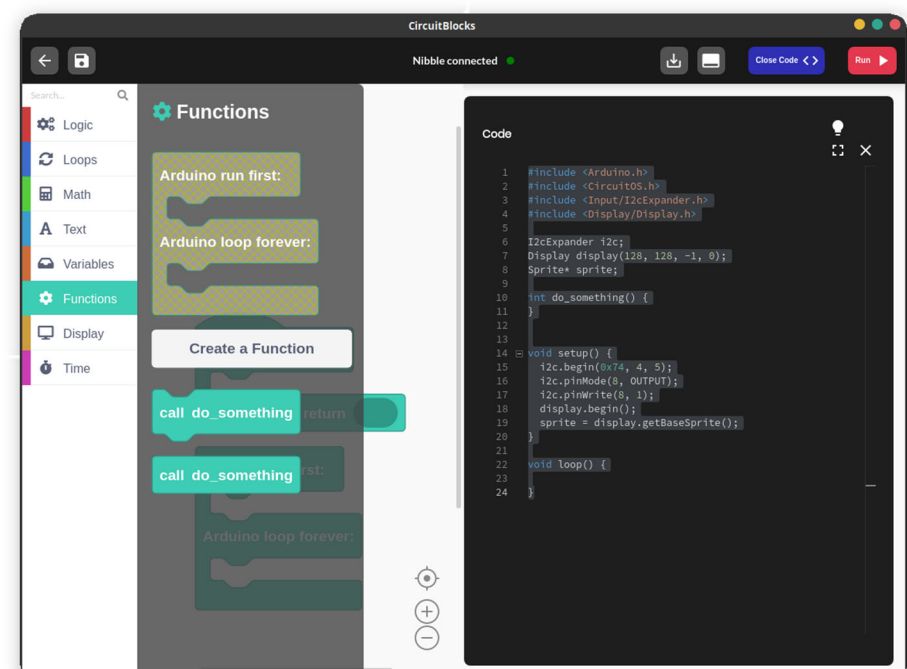
CBは自動的に変数の型 (int, double, string, boolean) を認識するので、心配する必要はありません。



Functions (ファンクション)

デフォルトのArduino機能 (前のページで説明) は、ここにあります。

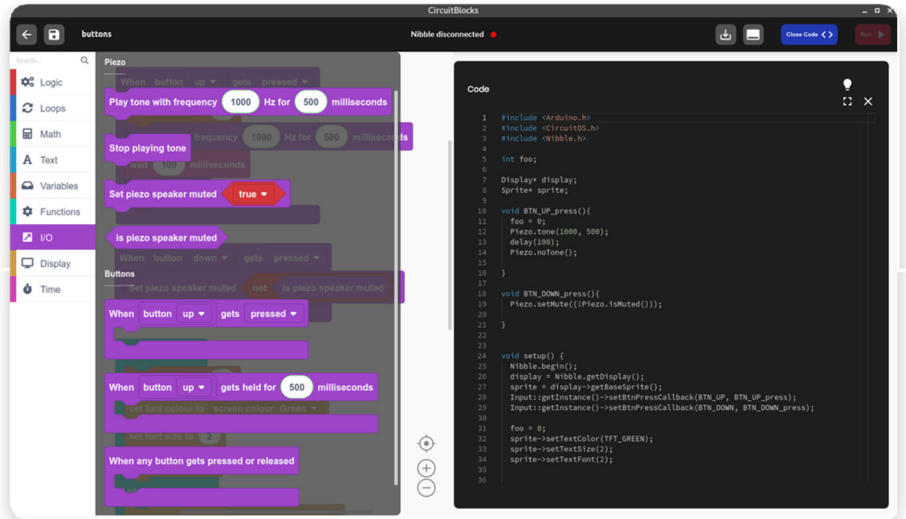
また、自分で関数を作成して、プログラムの主要な部分として挿入することもできます。



Input/Output（入力/出力）

Nibble のボタンとオーディオに関するすべてがここに
あります。

機能は多くありませんが、
正しく使えば驚くべきこと
ができます。

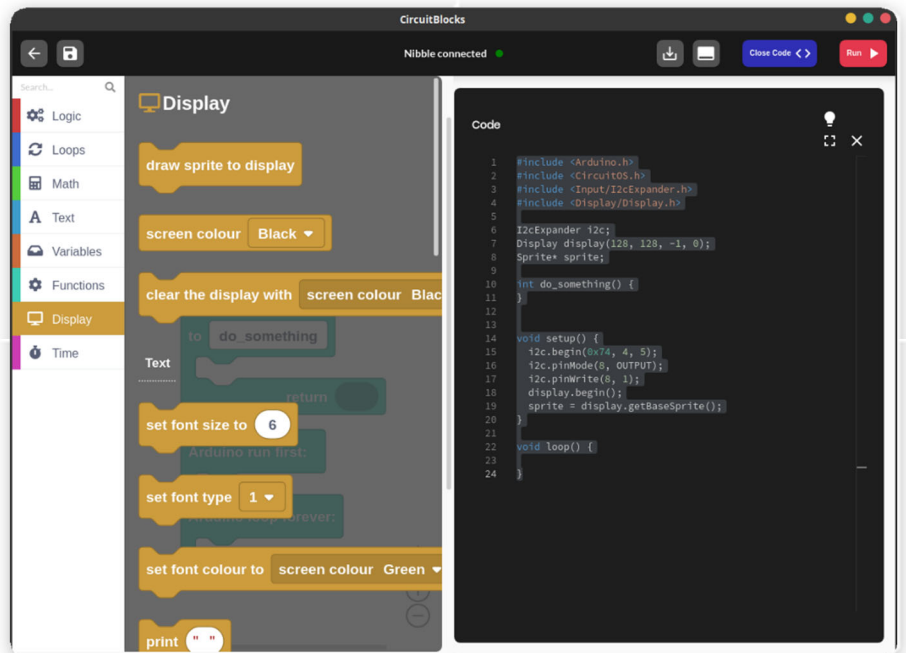


Display（ディスプレイ）

さて、これらのブロック
は、画面上に何も表示され
なければ、本当に重要では
ありません！

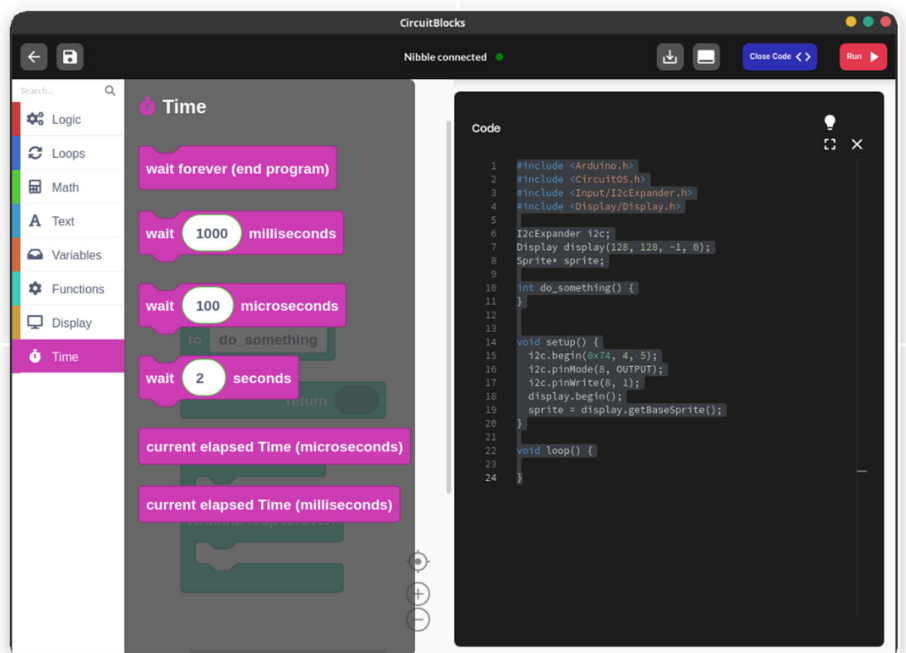
ここで、すべてのマジック
が色のついたピクセルに変
換されます。

あなたはこれらのブロック
を通して多くのものを作り
出すことができます。



Time（時間）

遅延、タイマー、および時
間に関連するものは、クール
なアニメーションやビデオ
ゲームを作るのに適してい
ます。



検索バー

すべての機能セクションの上に**検索バー**があり、どうしても見つからない特定のブロック
を簡単に検索できます（PRINTはどこ？）。

頭に浮かんだことを入力するだけで、書かれた文字に関係するすべてのブロックが右側に表示されます。

これで、見つけることが不可能とは言えません。

ブロックについてすべてを学びました。

次のレッスンに進みましょう。

画面の色を変える

あらゆる種類のブロックに慣れてきたので、今度はそれらの使い方を学びましょう。

Nibbleライブラリがどのように機能するかを理解できる一連の小さな例があります。

それぞれの例で、異なる機能を紹介します。最後に、すべてを組み合わせクールなアプリを作成します。

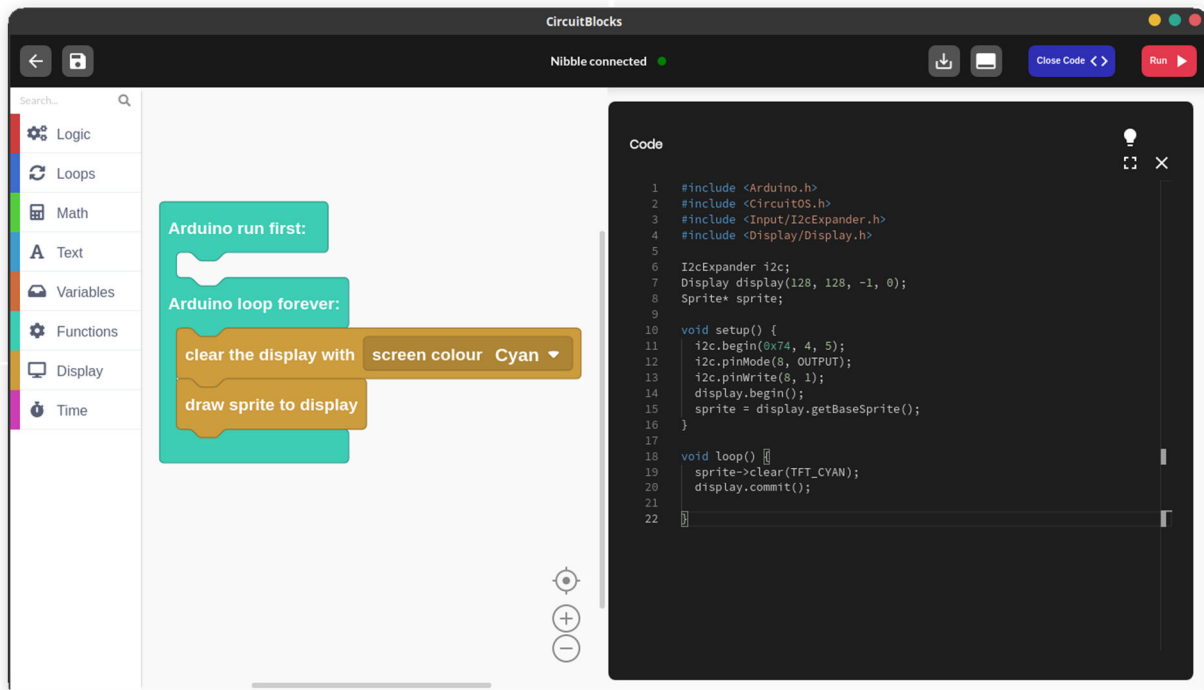
さあ、始めましょう。

例1

できるだけシンプルに始めましょう。

Nibbleの主なコンポーネントは画面です。画面なしで動作させることは、ほとんど不可能だからです。

テストとしてやりたいことは、プログラムの画面の色を変更することです。デフォルトの画面の色は黒で、ライブラリではTFT_BLACKと呼ばれています。ライブラリのすべての色は、TFT_プレフィックスでラベル付けされています。これは、TFT画面で使用するよう作成されているためです。



次に、ブロックによって生成されるコードに移りましょう。

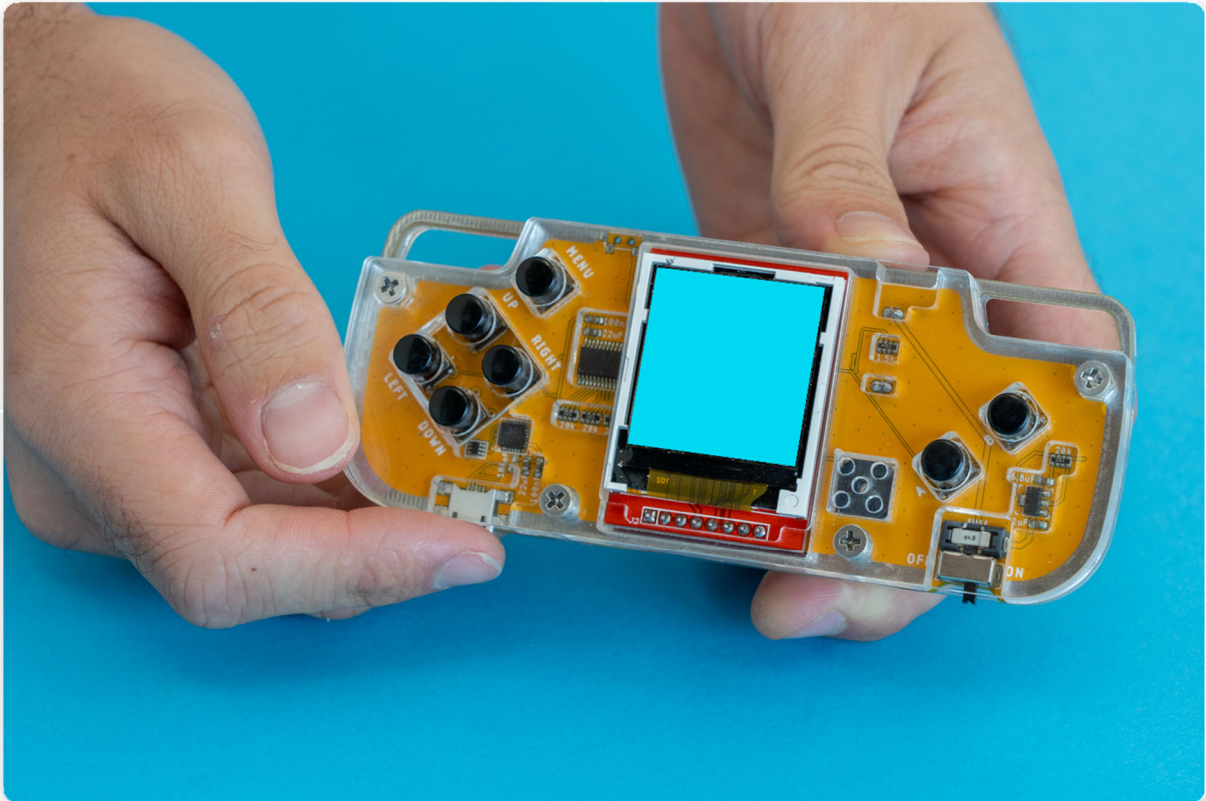
画面をシアン色にしたいので、別のパラメーターを使用して、Displayセクションから `sprite->clear()` 関数を呼び出しています。

すべての設定が完了しましたので、プログラムを実行できます。

プログラムはコンパイルされた後、コンソールにアップロードされます。

Ringoの画面はシアン色になっているはずですが、このようにシンプルです。

これは始まりにすぎませんが、このスクリーンの可能性は無限大であることが想像できます。



カラー

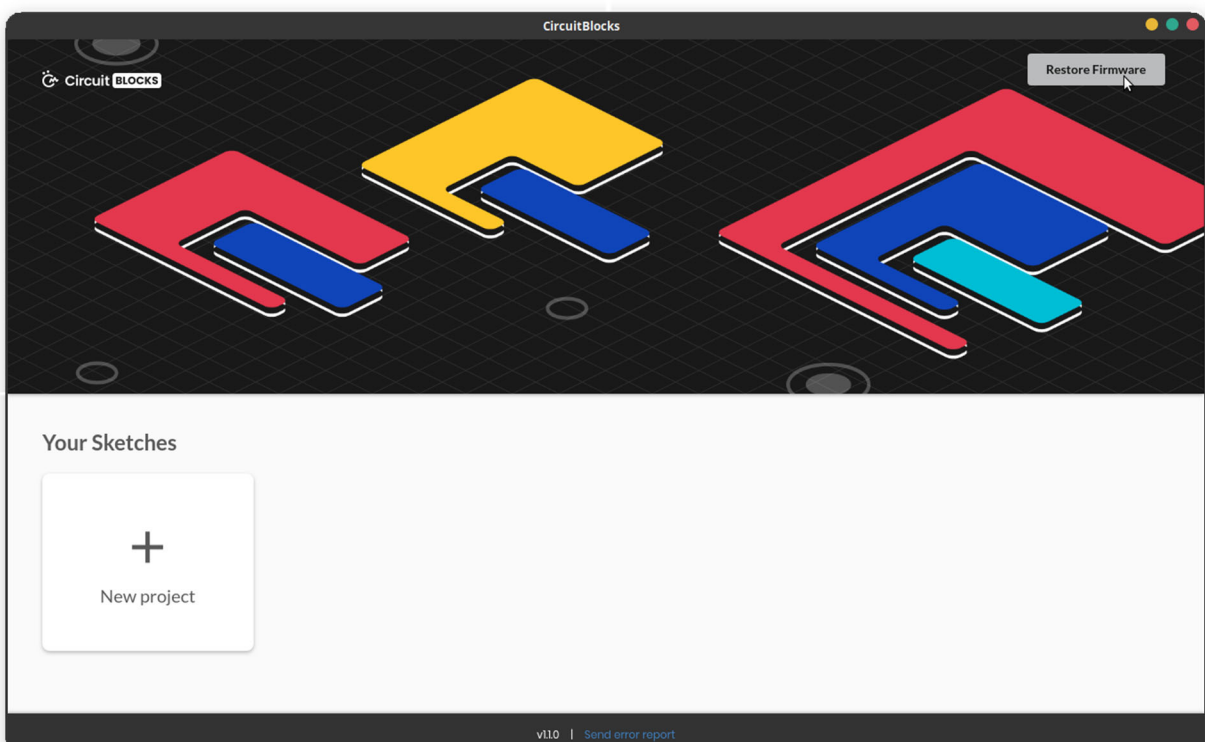
カラーバリエーションは、ディスプレイセクションでご確認いただけます。

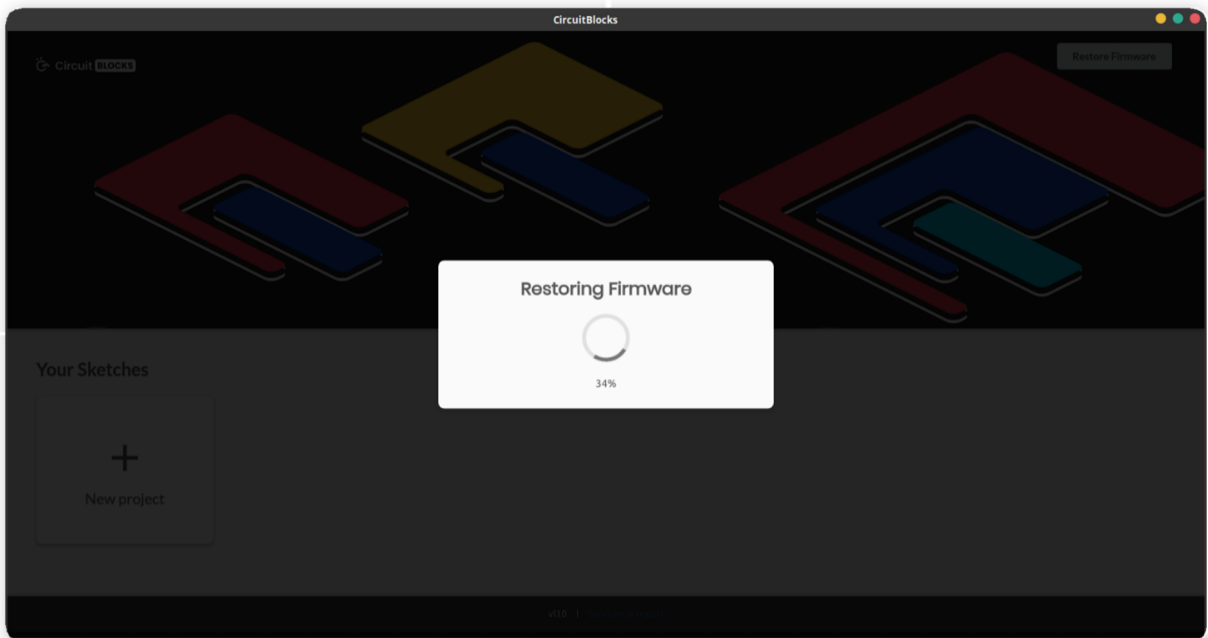
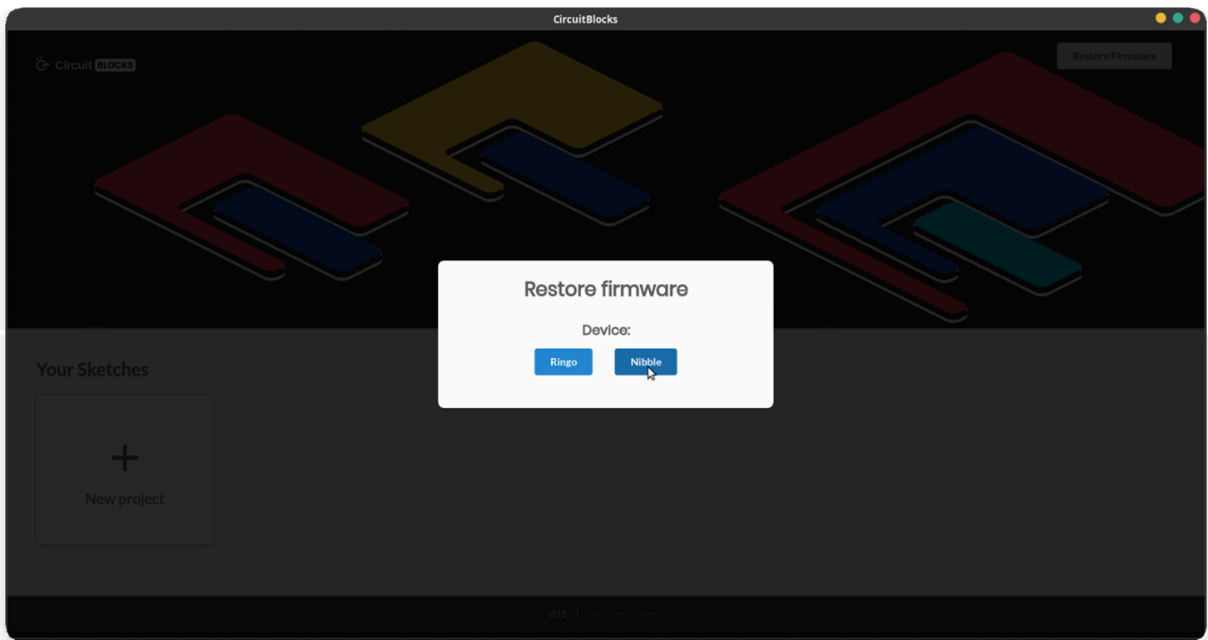
デフォルトに戻す

では、クールなものを作るのをやめて、プレインストールされたゲームだけを使いたい場合はどうすればいいでしょうか。数回クリックするだけで、すべてを元通りにすることができます。

Nibbleファームウェアをプレインストールされたゲームに戻したいときは、メインメニューの **[Restore firmware (ファームウェアを復元)]** ボタンを押し、**[Nibble]** を選択すれば、いつでも復元できます。

終了する前にプロジェクトを保存することを忘れないでください。

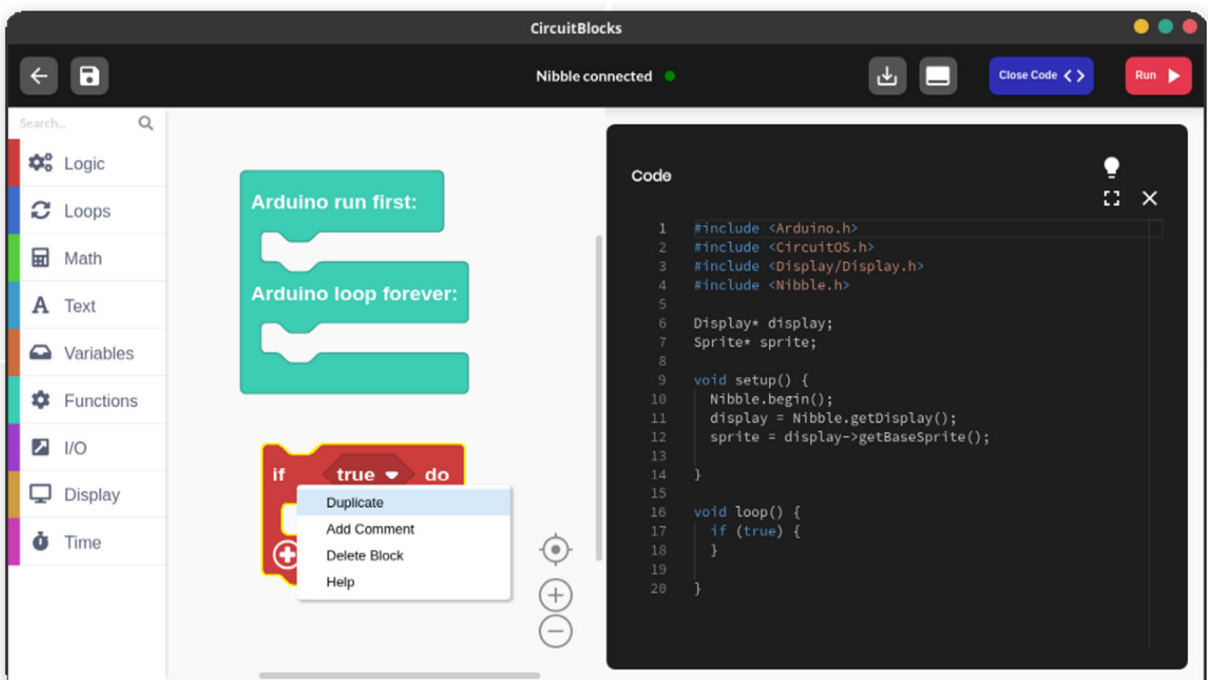




基本について説明したので、もう少し高度な内容に進みましょう。

ディスプレイの制御

クイックコマンド



次のステップに進む前に、プログラミングに役立つちょっとしたコツがあります。

ブロックの1つをマウスで右クリックすると、クイックアクションメニューが開き、ブロックの複製、コメントの追加、ブロックの削除、ヘルプの検索が簡単にできます。

これらのコマンドの一部は、キーボードの他のボタンを押して実行することもできます。

ブロックを削除するには、**Windows/Linux**では**DEL**button、**MacOS**では**DELETE**を押してください。

ブロックをコピーするには、**Windows/Linux**では**Ctrl+C**、**MacOS**では**Cmd+C**を押してください。

ブロックを切り取りするには、**Windows/Linux**では**Ctrl+X**、**MacOS**では**Cmd+X**を押してください。

ブロックをペーストするには、**Windows/Linux**では**Ctrl+V**、**MacOS**では**Cmd+V**を押してください。

色をスクロールさせる

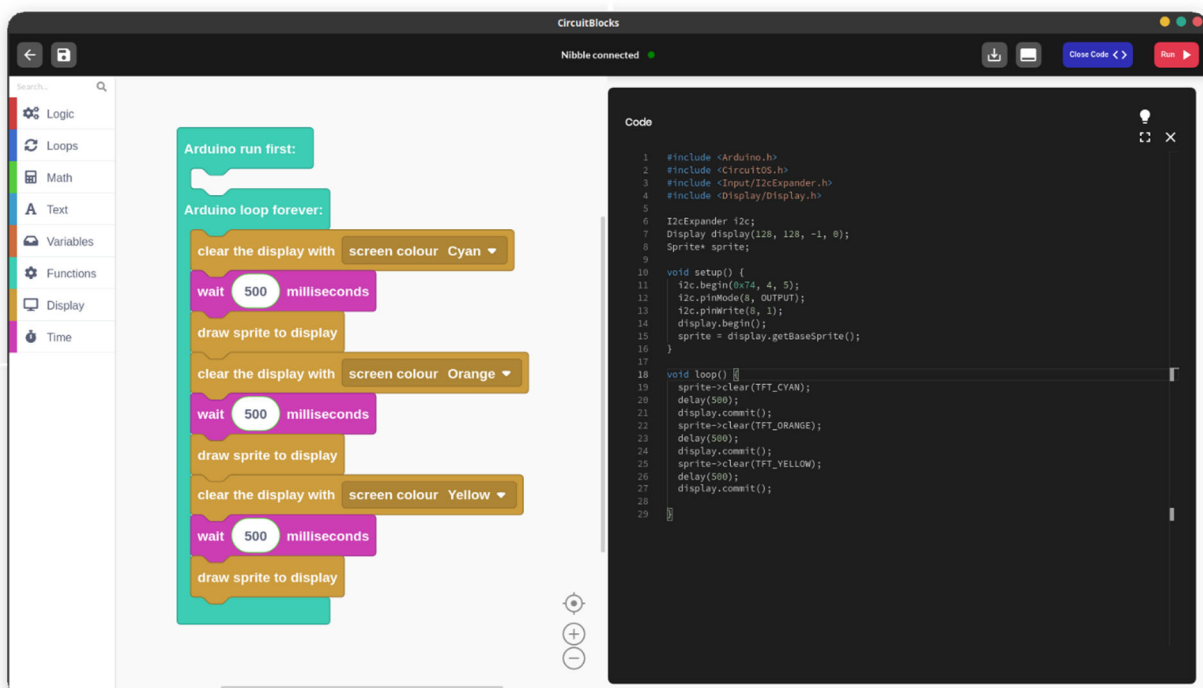
次に、**loop()**の部分を使用します。

display.commit()を参照してください。おそらく最も重要な関数であり、ディスプレイセクションの下にあります。

この関数の使命は、最後に呼び出されてから現在までに発生したすべてを読み取り、それらの変更をハードウェアに転送することです。

例えば、`sprite->clear()`関数は、`display.commit()`が呼び出されない場合、何も変更しません。

この関数は、「ブロックに表示するスプライトを描く」と言っているのです。



また、**wait()**という別の新しいブロックも使用しています。

これはあるミリ秒の間、すべてを停止させる**delay()**関数に変換するものです。

数値ブロックはMathセクションにあり、その値はニーズに合わせて変更できます。

`loop()`関数は非常に高速なので、画面上で何が起きているかを確認するために、この遅延が必要な場合があります。

ただし、高速で応答性の高いプログラムが期待される場合に遅延を使用することは、特にボタンを使用している場合には適していません。

これについては、次のレッスンで詳しく説明します。

このプログラムで行っているのは、画面の色を3つの値で交互に切り替えることです。

まずシアン色に変えて、500ミリ秒（0.5秒）待ちます。

次に、オレンジ色に変えて、さらに500ミリ秒待ちます。

最後に、黄色に変えて500ミリ秒待ち、ループの最初に戻り、再びシアン色に変えています。

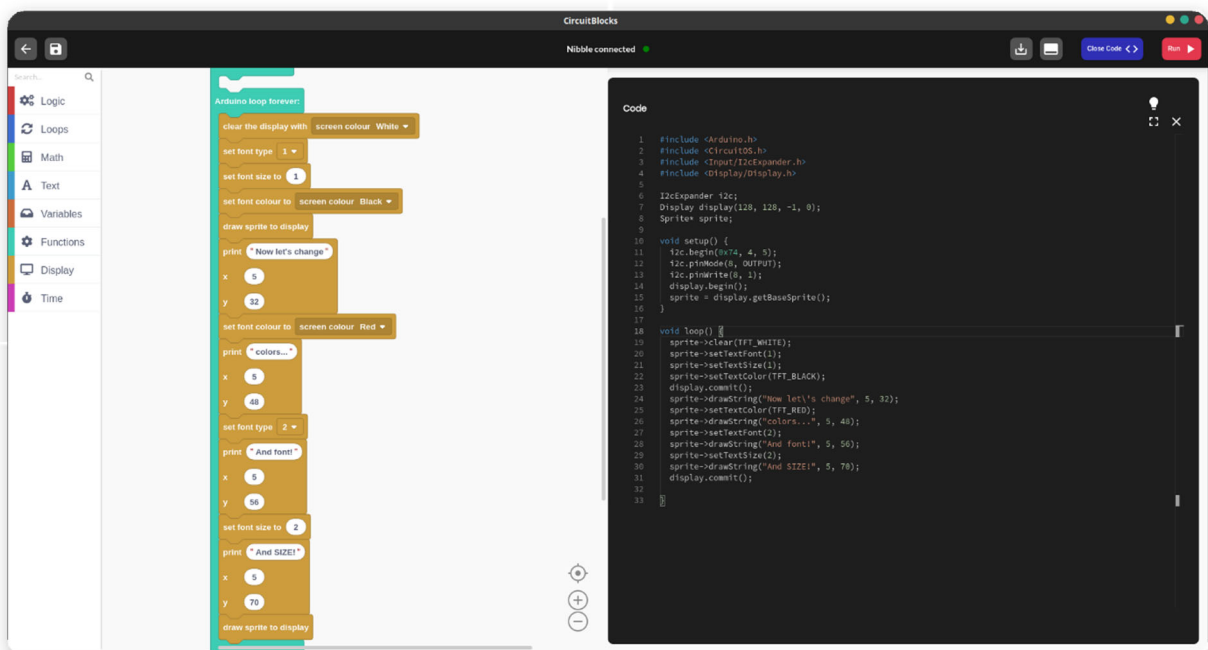
色を変更するたびに**display.commit()** 関数を呼び出して、その色を画面に転送していることに注意してください。

テキストを書き出す

背景色を変更する方法がわかったので、次はキャンバスに何か書いてみましょう。

Nibble ライブラリには**3種類**のフォントが用意されており、好きなようにサイズや色を変更することができます。

そのための関数はかなりシンプルで、誰でも理解できるはずです。



このプログラムでは、さまざまなフォント、サイズ、色でいくつかの文字を画面に書き出します。

フォントの種類は、フォントタイプ設定ブロックのドロップダウンメニューで簡単に変更できます。ただしフォントサイズは、選択したフォントのサイズに任意の値を掛けたものです。

デフォルトではこれらの値は両方とも1であるため、プログラムの最初に同じ値を設定しても違いはありません。



フォントの種類を指定し、[set font type] ブロックをプログラム内に置くことが重要です。そうしないと、プログラムがクラッシュする可能性があります。

その後、これらの値は両方とも変更されました。最後の例では、「**And SIZE!**」という2つの単語を、**2倍の大きさ**の別のフォントで出力しています。

ここでの**print()**関数は、かなり頻繁に使用するものです。最初の空のスロットは、書き出すための文字列です。

空のものは**Text**セクションで見つけることができます。

2番目と3番目のスロットは、最初の文字の位置です。これらの値を両方とも0に設定すると、画面の**左上隅**に文字が出力されます。

これらの数値は、ピクセルの位置を参照しています。画面は**128×128ピクセル**で、すべて

を操作できます。

xの値が0から127、yの値が0から127の間で書き込み/描画したものは、すべて画面上に表示されます。

xの値が0から127、yの値が0から127の間で書き込み/描画したものは、すべて画面上に表示されます。

これらの値のいずれかがその範囲外にある場合、コンポーネントは表示されません。

ちょっとしたコードの追加

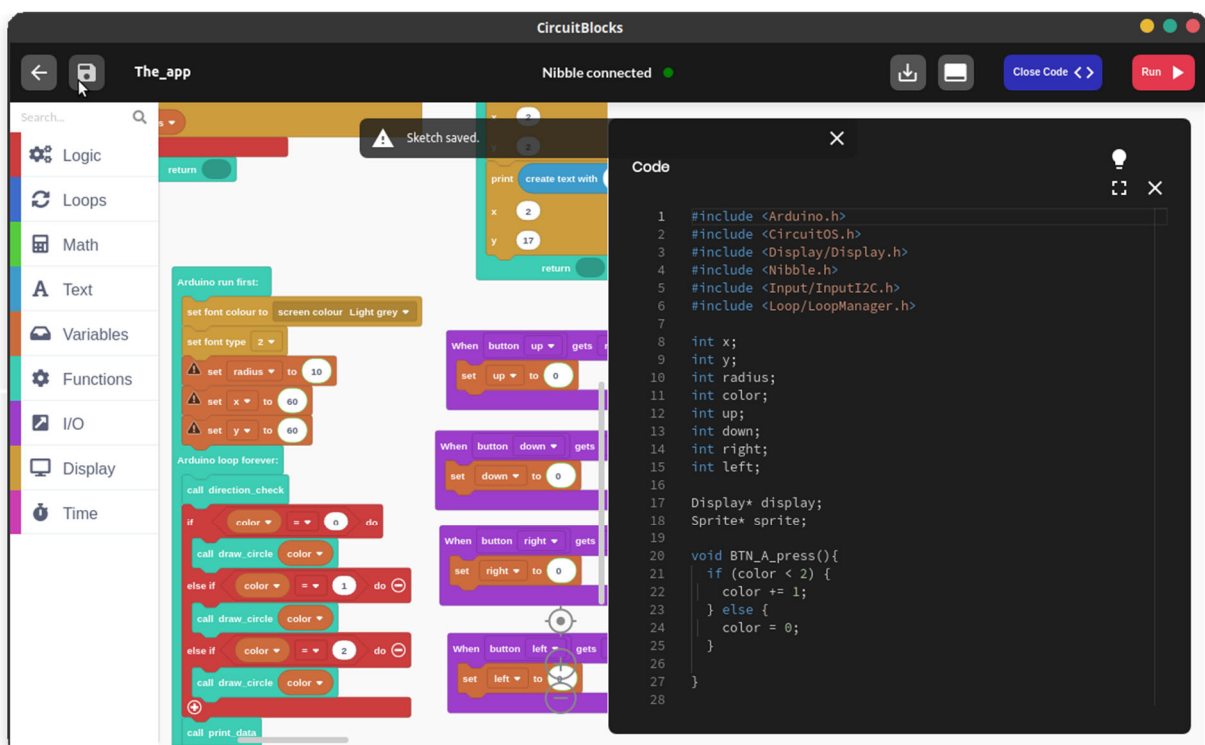
さて、ゲームが完成したところで、音が出ないことに気付きました。

デバイスには小さなブザーがあり、様々な音を出すことができます。ピープ音、バズ音など。

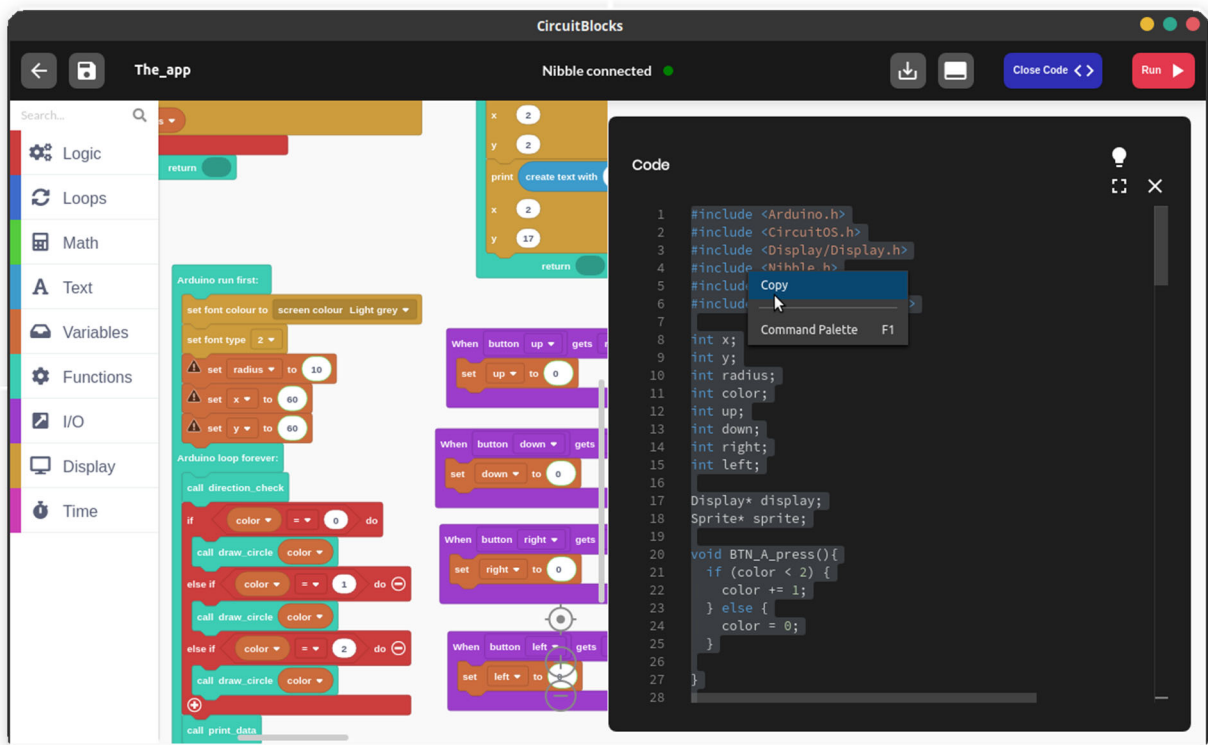
ブロックを使って追加することもできますが、コードで行う方法を説明します。

この機会に、画面右側にあるゲームコードを丸ごとコピーして、新しいプロジェクトを作りましょう。

まず、プロジェクトを保存し、名前をつけるのを忘れないでください。

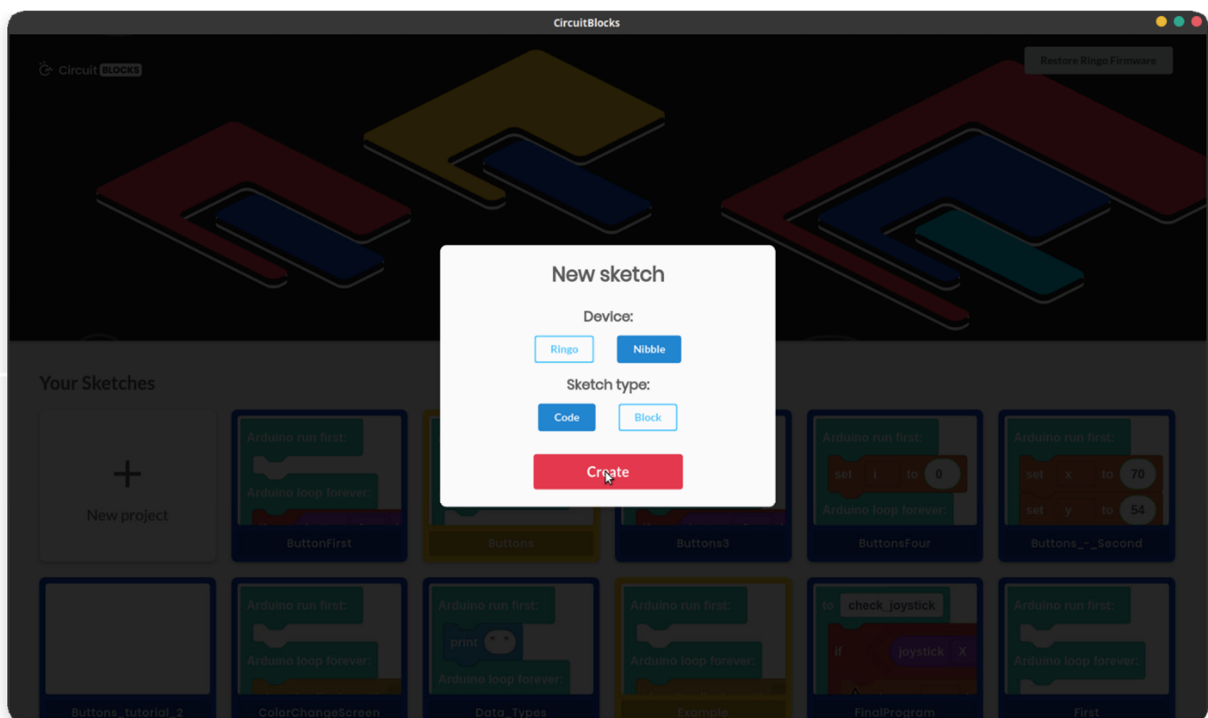


次に、コードを選択してマウスの右ボタンを押し、「コピー」をクリックするか、Ctrl+C/Cmd+Cを押してコード全体をコピーしてください。

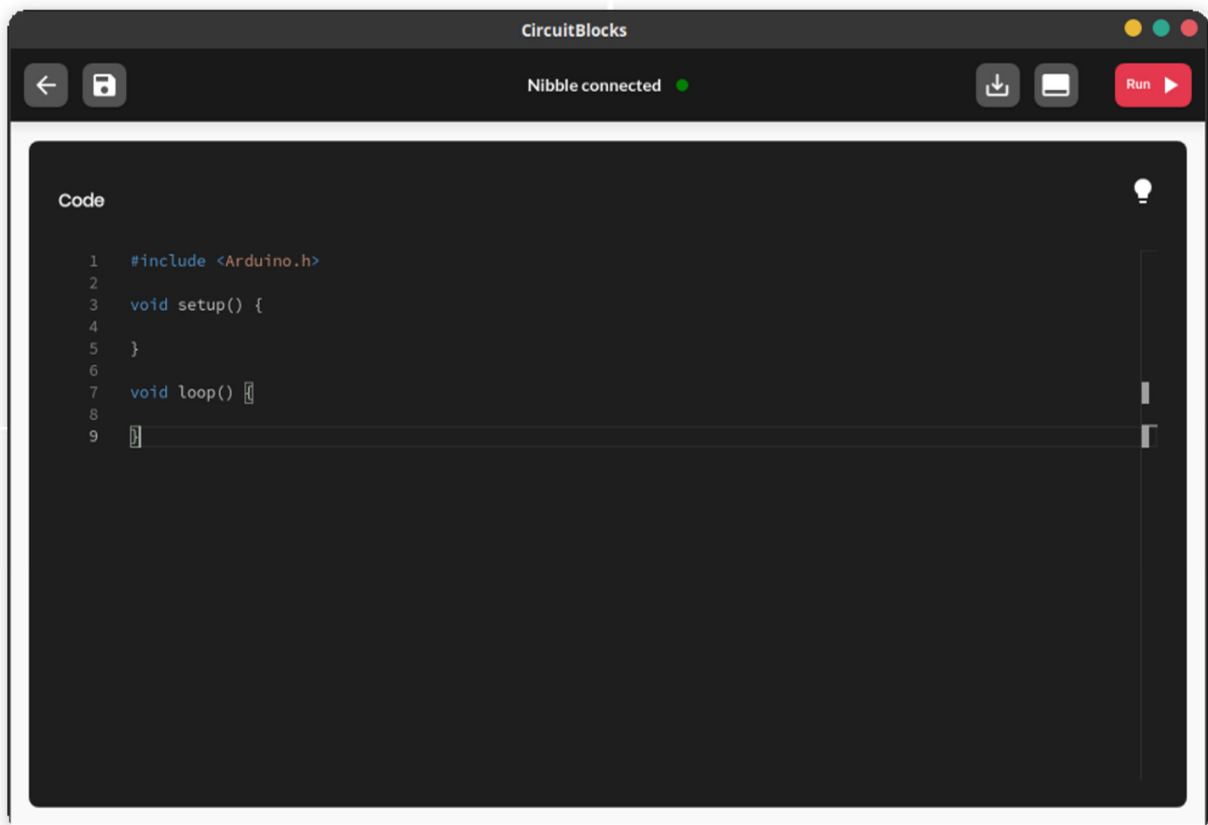


メインメニューに戻り、新しいプロジェクトを開きます。

プロジェクトの種類にNibbleとコードが選択されていることを確認してください。



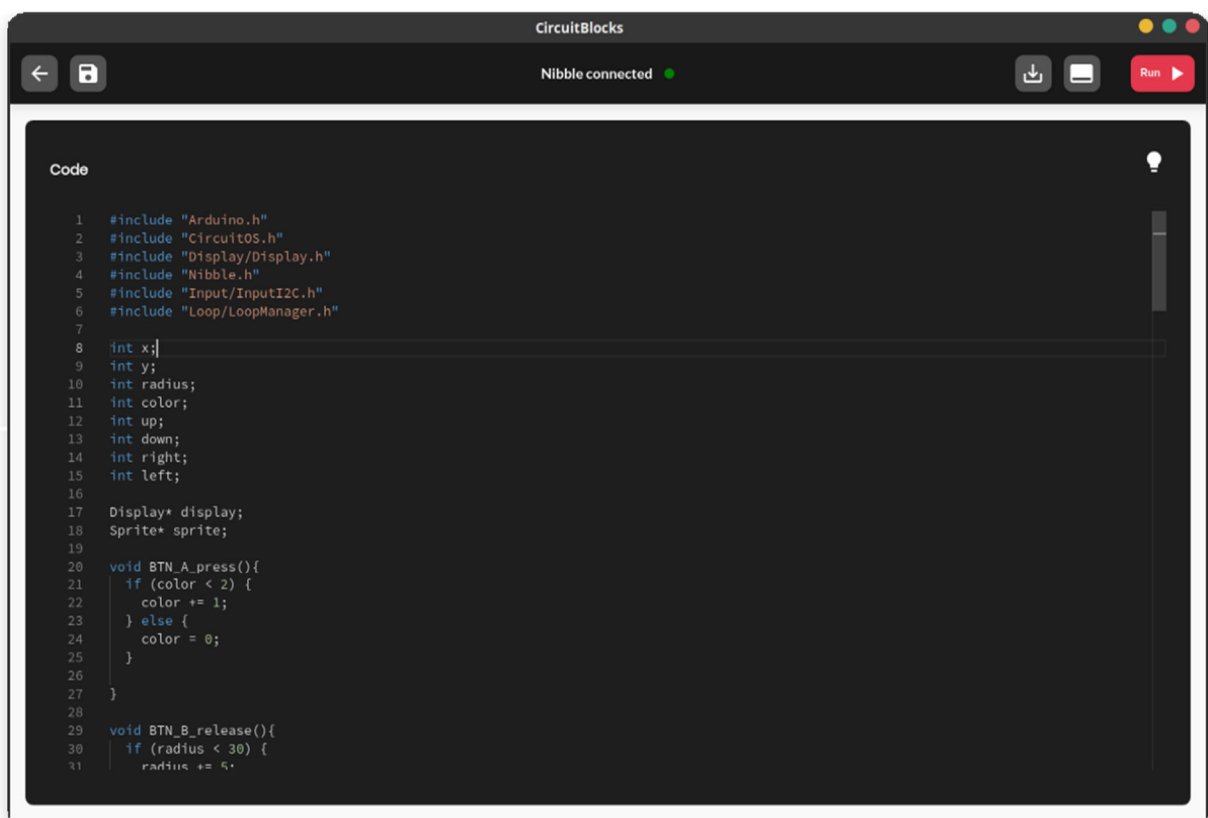
画面は次のようになります。



```
Code
1  #include <Arduino.h>
2
3  void setup() {
4
5  }
6
7  void loop() {
8
9  }
```

内部にあるコードを削除し、コピーしたコードを空の画面に貼り付けてください。

[Run] をクリックしてコードをアップロードすると、Nibbleは以前と同じゲームになるはずですが。



```
Code
1  #include "Arduino.h"
2  #include "CircuitOS.h"
3  #include "Display/Display.h"
4  #include "Nibble.h"
5  #include "Input/InputI2C.h"
6  #include "Loop/LoopManager.h"
7
8  int x;
9  int y;
10 int radius;
11 int color;
12 int up;
13 int down;
14 int right;
15 int left;
16
17 Display* display;
18 Sprite* sprite;
19
20 void BTN_A_press(){
21     if (color < 2) {
22         color += 1;
23     } else {
24         color = 0;
25     }
26 }
27 }
28
29 void BTN_B_release(){
30     if (radius < 30) {
31         radius += 5;
32     }
33 }
```

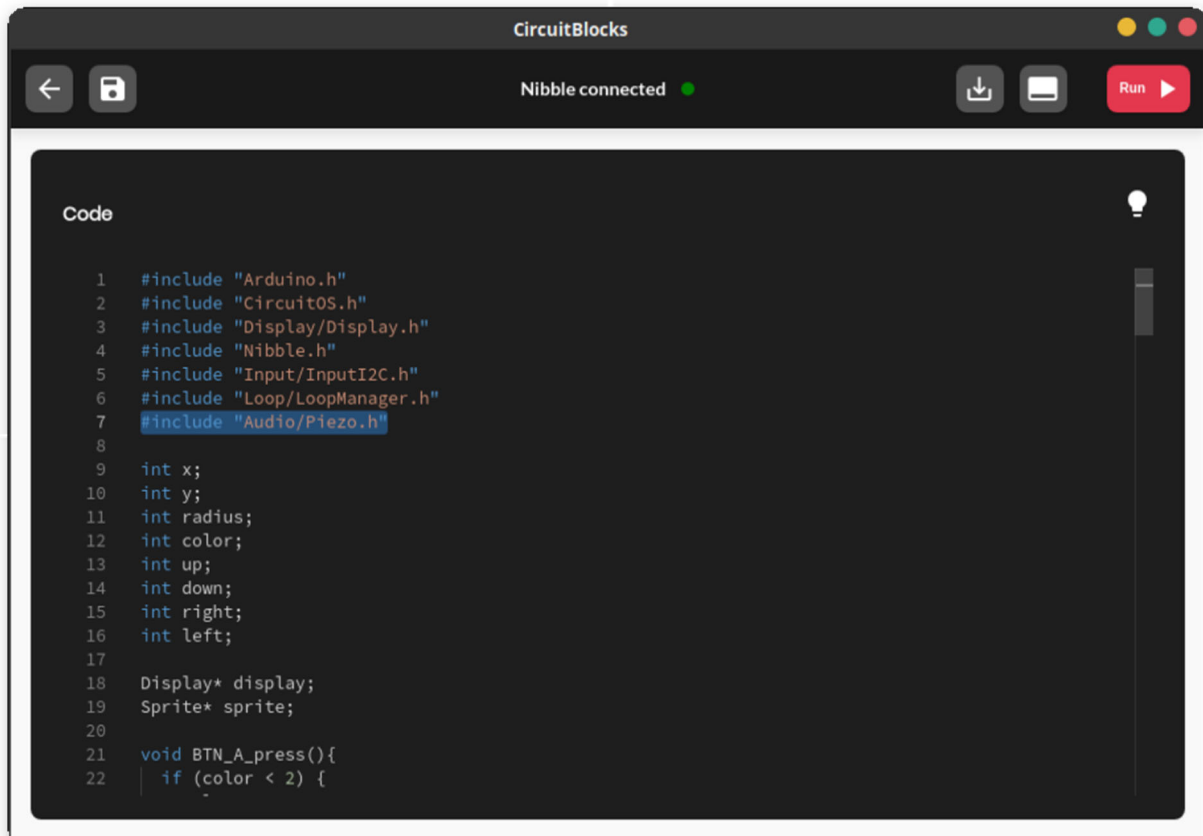
では、サウンドを追加してみましょう。

まず最初に、サウンドライブラリを追加する必要があります。

今回使用するのはPiezoというライブラリで、`#include`セクションの任意の場所に次の行を追加してください。

ARDUINO

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



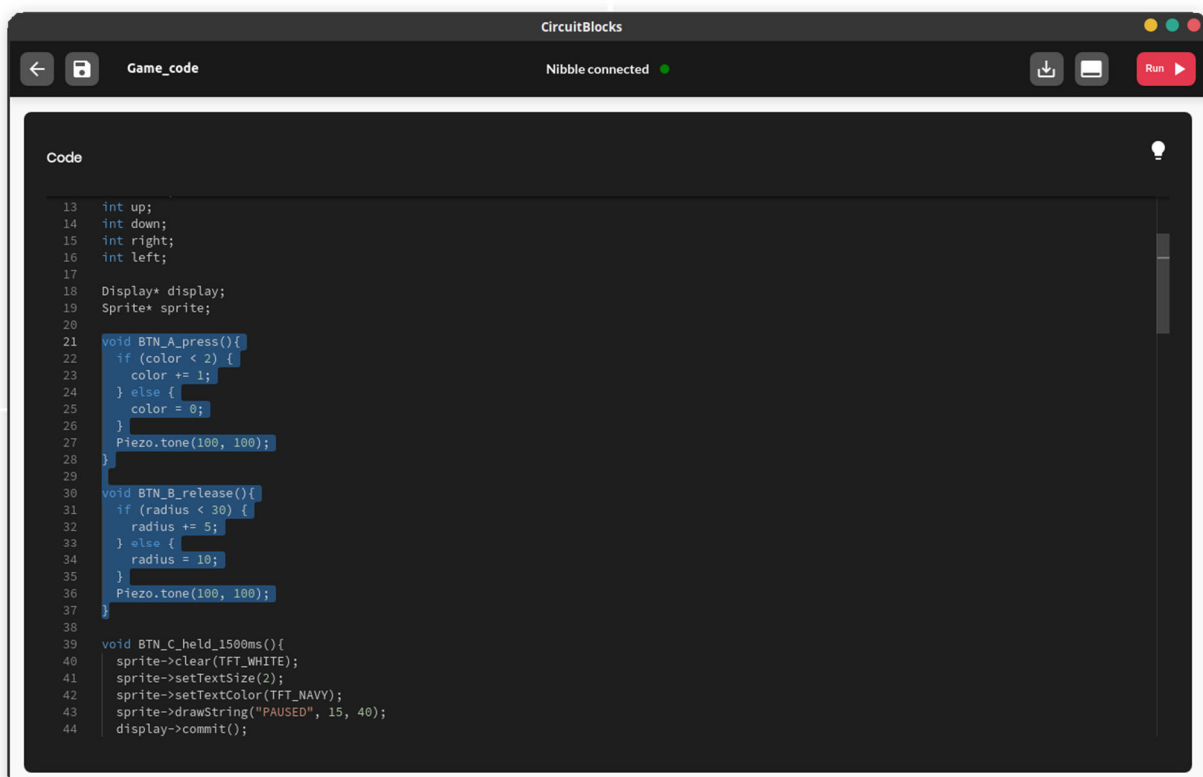
```
CircuitBlocks
Nibble connected
Code
1 #include "Arduino.h"
2 #include "CircuitOS.h"
3 #include "Display/Display.h"
4 #include "Nibble.h"
5 #include "Input/InputI2C.h"
6 #include "Loop/LoopManager.h"
7 #include "Audio/Piezo.h"
8
9 int x;
10 int y;
11 int radius;
12 int color;
13 int up;
14 int down;
15 int right;
16 int left;
17
18 Display* display;
19 Sprite* sprite;
20
21 void BTN_A_press(){
22     if (color < 2) {
```

次に、**Aボタン**または**Bボタン**を押すたびに、ちょっとした音色を加えてみましょう。

`Piezo.tone()`という関数で、括弧の中に2つの数字が入っています。

最初の数値は音の周波数を、2番目の数値は音の長さを決めます。

数値を少し変えて、いろいろな音を出してみましょう。



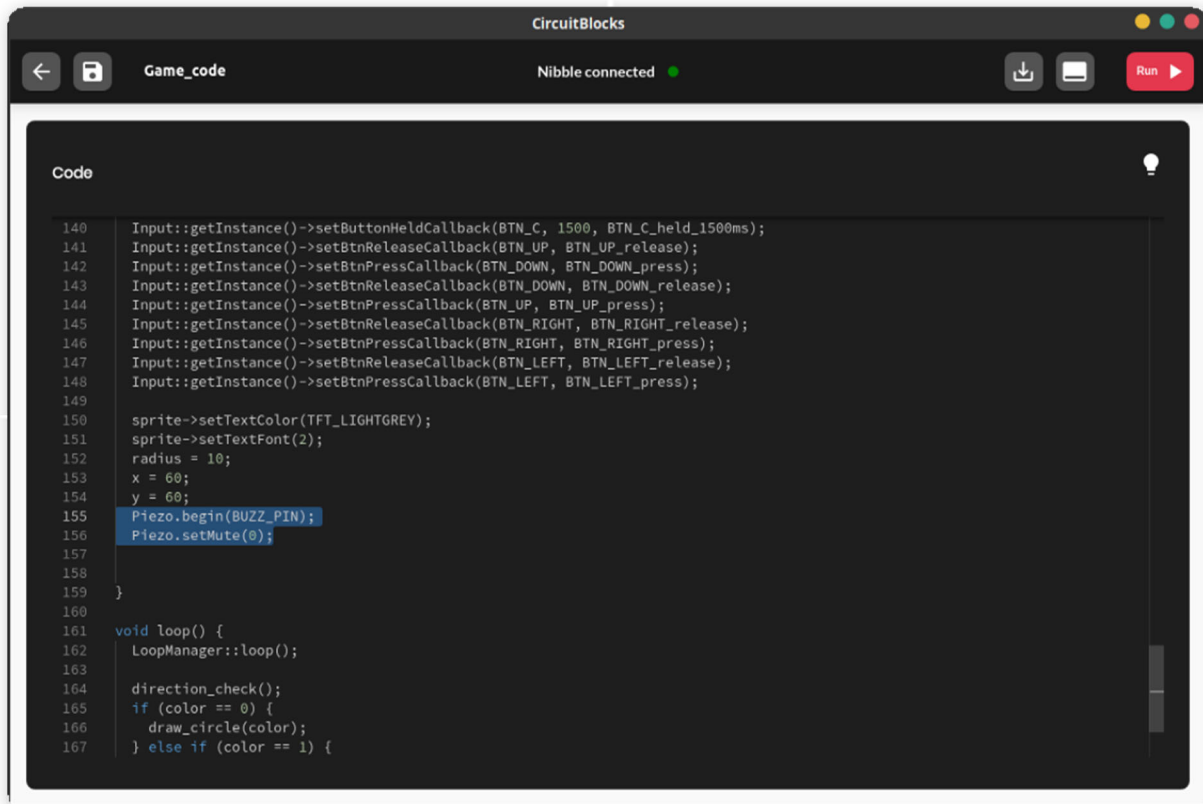
```
CircuitBlocks
Game_code
Nibble connected
Code
13 int up;
14 int down;
15 int right;
16 int left;
17
18 Display* display;
19 Sprite* sprite;
20
21 void BTN_A_press(){
22     if (color < 2) {
23         color += 1;
24     } else {
25         color = 0;
26     }
27     Piezo.tone(100, 100);
28 }
29
30 void BTN_B_release(){
31     if (radius < 30) {
32         radius += 5;
33     } else {
34         radius = 10;
35     }
36     Piezo.tone(100, 100);
37 }
38
39 void BTN_C_held_1500ms(){
40     sprite->clear(TFT_WHITE);
41     sprite->setTextSize(2);
42     sprite->setTextColor(TFT_NAVY);
43     sprite->drawString("PAUSED", 15, 40);
44     display->commit();
```

仕上げの前にもうひとつ、サウンドをオンにしてください。

メインのvoid setup() 関数を探し、一番下に次の行を追加してください。

ARDUINO

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



```
140 Input::getInstance()->setButtonHeldCallback(BTN_C, 1500, BTN_C_held_1500ms);  
141 Input::getInstance()->setBtnReleaseCallback(BTN_UP, BTN_UP_release);  
142 Input::getInstance()->setBtnPressCallback(BTN_DOWN, BTN_DOWN_press);  
143 Input::getInstance()->setBtnReleaseCallback(BTN_DOWN, BTN_DOWN_release);  
144 Input::getInstance()->setBtnPressCallback(BTN_UP, BTN_UP_press);  
145 Input::getInstance()->setBtnReleaseCallback(BTN_RIGHT, BTN_RIGHT_release);  
146 Input::getInstance()->setBtnPressCallback(BTN_RIGHT, BTN_RIGHT_press);  
147 Input::getInstance()->setBtnReleaseCallback(BTN_LEFT, BTN_LEFT_release);  
148 Input::getInstance()->setBtnPressCallback(BTN_LEFT, BTN_LEFT_press);  
149  
150 sprite->setTextColor(TFT_LIGHTGREY);  
151 sprite->setTextFont(2);  
152 radius = 10;  
153 x = 60;  
154 y = 60;  
155 Piezo.begin(BUZZ_PIN);  
156 Piezo.setMute(0);  
157  
158  
159 }  
160  
161 void loop() {  
162   LoopManager::loop();  
163  
164   direction_check();  
165   if (color == 0) {  
166     draw_circle(color);  
167   } else if (color == 1) {
```

赤い [Run] ボタンを押してゲームを再アップロードすると、ゲームにサウンドが追加されました。

必要に応じて、さらに多くのサウンドを追加し、ゲームをさらにクールにすることができます。

ゲームのアイデア

[CircuitMess GitHubのpage](#) Nibbleだけでなく他のコンソール用のゲームも多数掲載されています。

これらのコードを学習すると、これらのゲームがどのように記述されているか、さまざまなアクションにどのような方法が使用されているかがわかります。また、最も重要なこととして、新しいゲームのアイデアを得ることができます。

また、[CircuitMessコミュニティフォーラム](#)など、さまざまなオンラインフォーラムにスクロールして、世界中の人々と知識やアイデアを交換することもできます。

可能性は無限大です。さっそくゲームのコードを書いてみましょう。