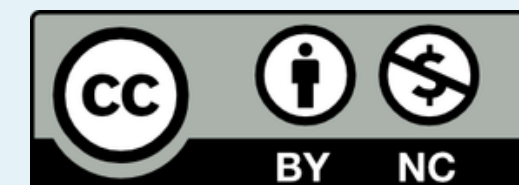


コードとデザイン

東京藝術大学 芸術情報センター開設科目 金曜4-5限 第9週

2024.06.14 松浦知也 (matsura.tomoya@noc.geidai.ac.jp teach@matsuuratomoya.com)



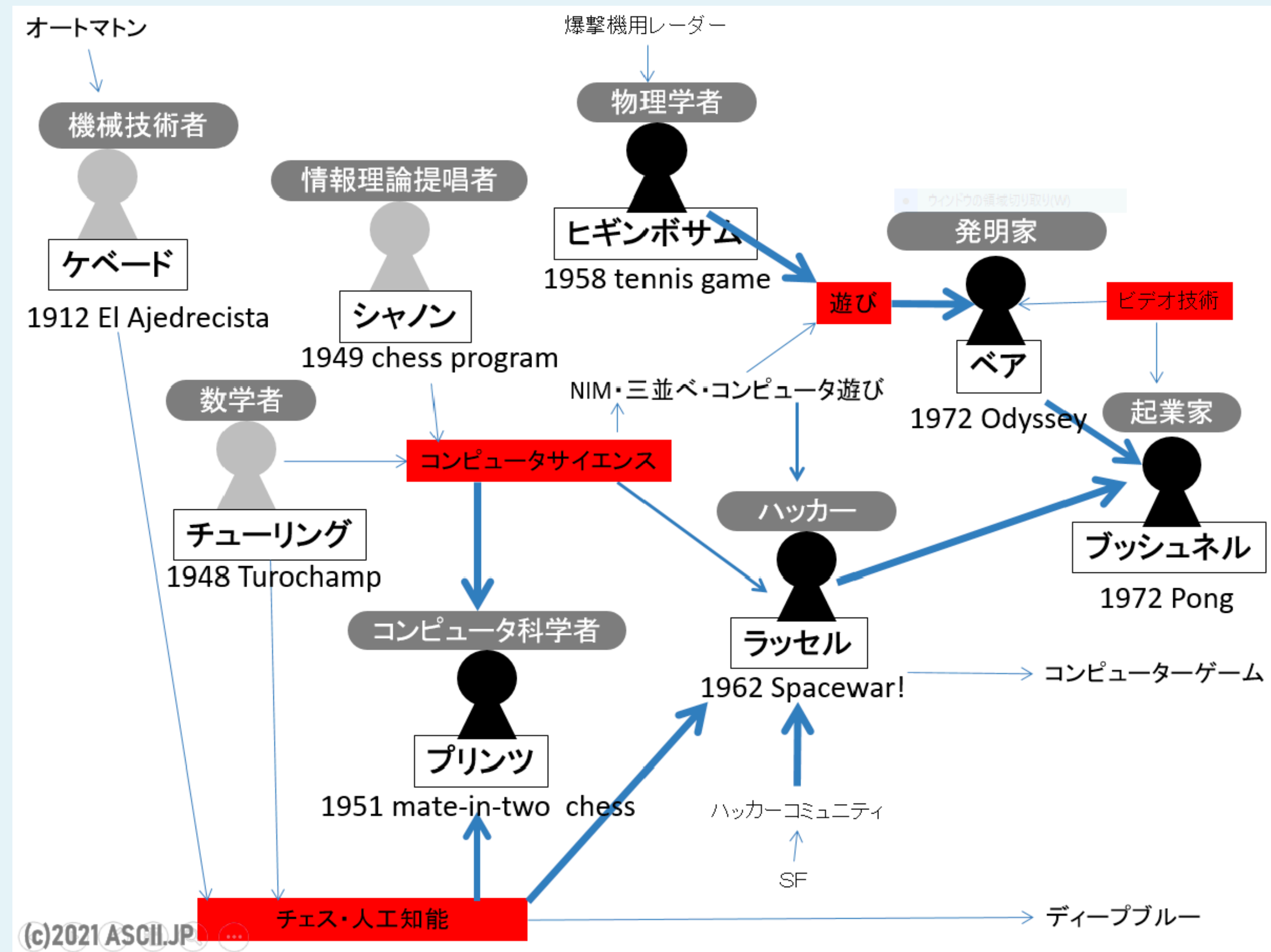
本日のスケジュール

- Processingでオブジェクト指向を学ぼう
- 1ボタン1スイッチでできるゲーム
- ProcessingとArduinoの連携方法（Firmata）
- 可変抵抗(ポテンシオメータ)を使った入力

コンピューターで遊ぶ



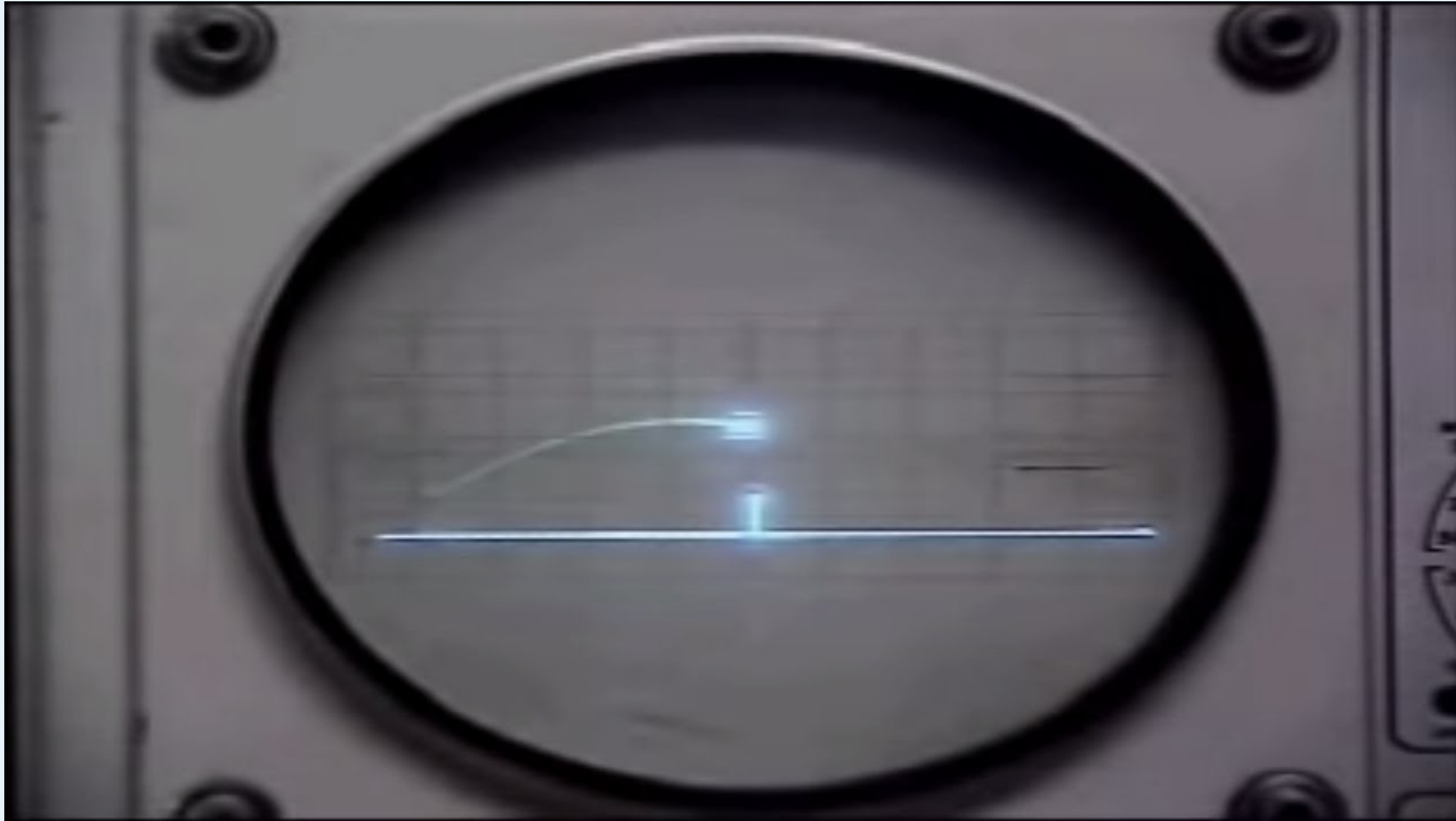
遊びのためのコンピューター



“テレビゲーム誕生の系統図- B-29爆撃機とテレビゲームの原点アタリ
PONGの関係”(2021), 遠藤諭 (角川アスキー総合研究所), 週刊アスキー

<https://weekly.ascii.jp/elem/000/004/078/4078025/>

Tennis For Two(1958)



Tennis for Two - The Original Video Game(2007), The Dot Eaters, CC-BY 3.0
https://www.youtube.com/watch?v=6PG2mdU_i8k

Tennis For Two(1958)



Pre-PONG - Tennis For Two - Precursors Higinbotham/BNL, 1958 - THE DOT EATERS,
<https://thedoteaters.com/?bitstory=bitstory-article-1%2Ftennis-for-two>

Tennis For Two(1958)



マンハッタン計画の一環の、軍事用レーダー開発のためのアナログコンピュータで作られた。
研究所の一般公開で原子力に不安を抱く市民へのアウトリーチの一環として制作された。
(実際の研究内容は軍事機密が多く公にしづらいことも影響)

Pre-PONG - Tennis For Two - Precursors Higinbotham/BNL, 1958 - THE DOT EATERS,
<https://thedoteaters.com/?bitstory=bitstory-article-1%2Ftennis-for-two>

Spacewar!(1962)



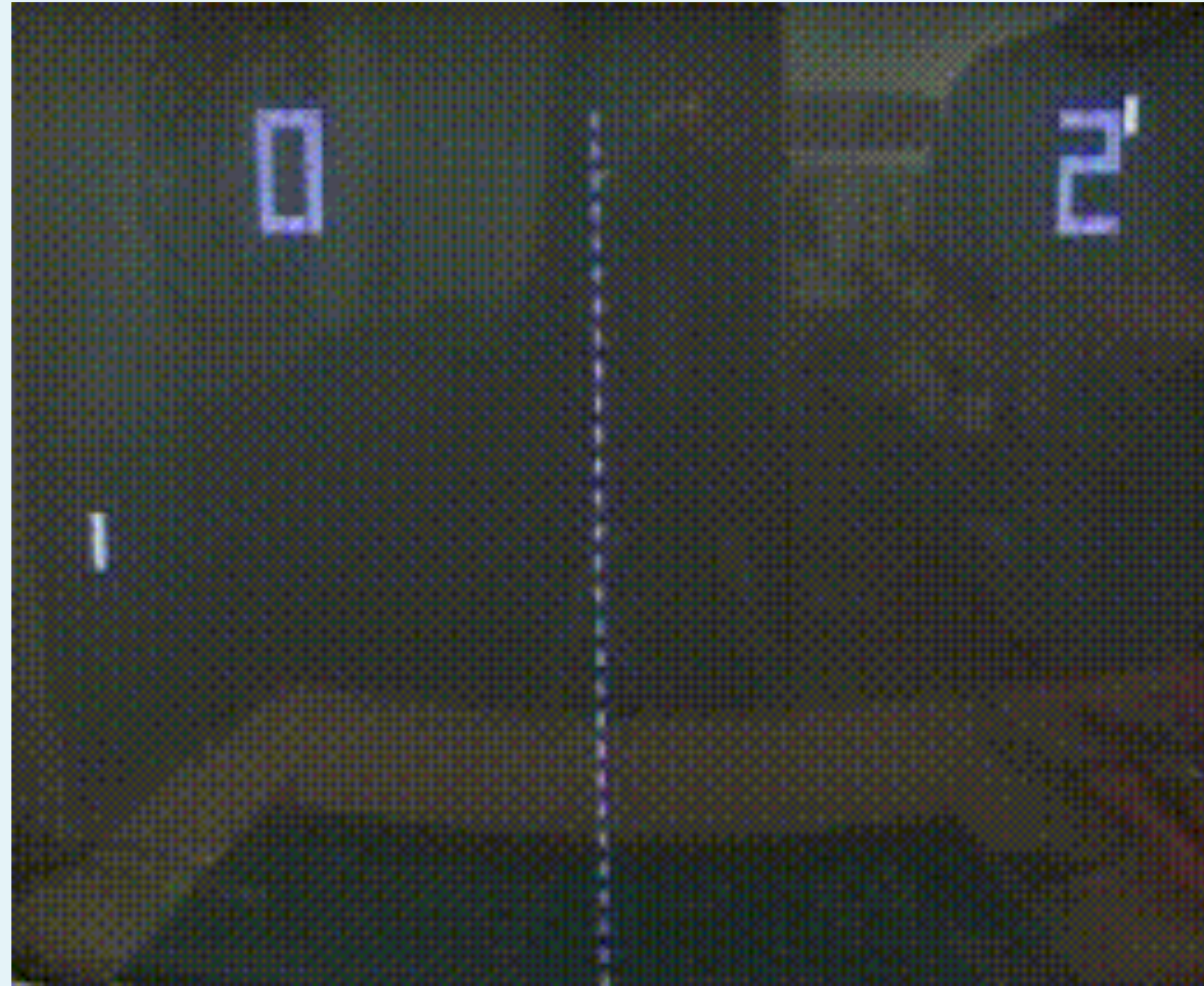
Joi Ito from Inbamura, Japan - Spacewar running on PDP-1, CC 表示 2.0,
<https://commons.wikimedia.org/w/index.php?curid=2099696>

Spacewar!(1962)



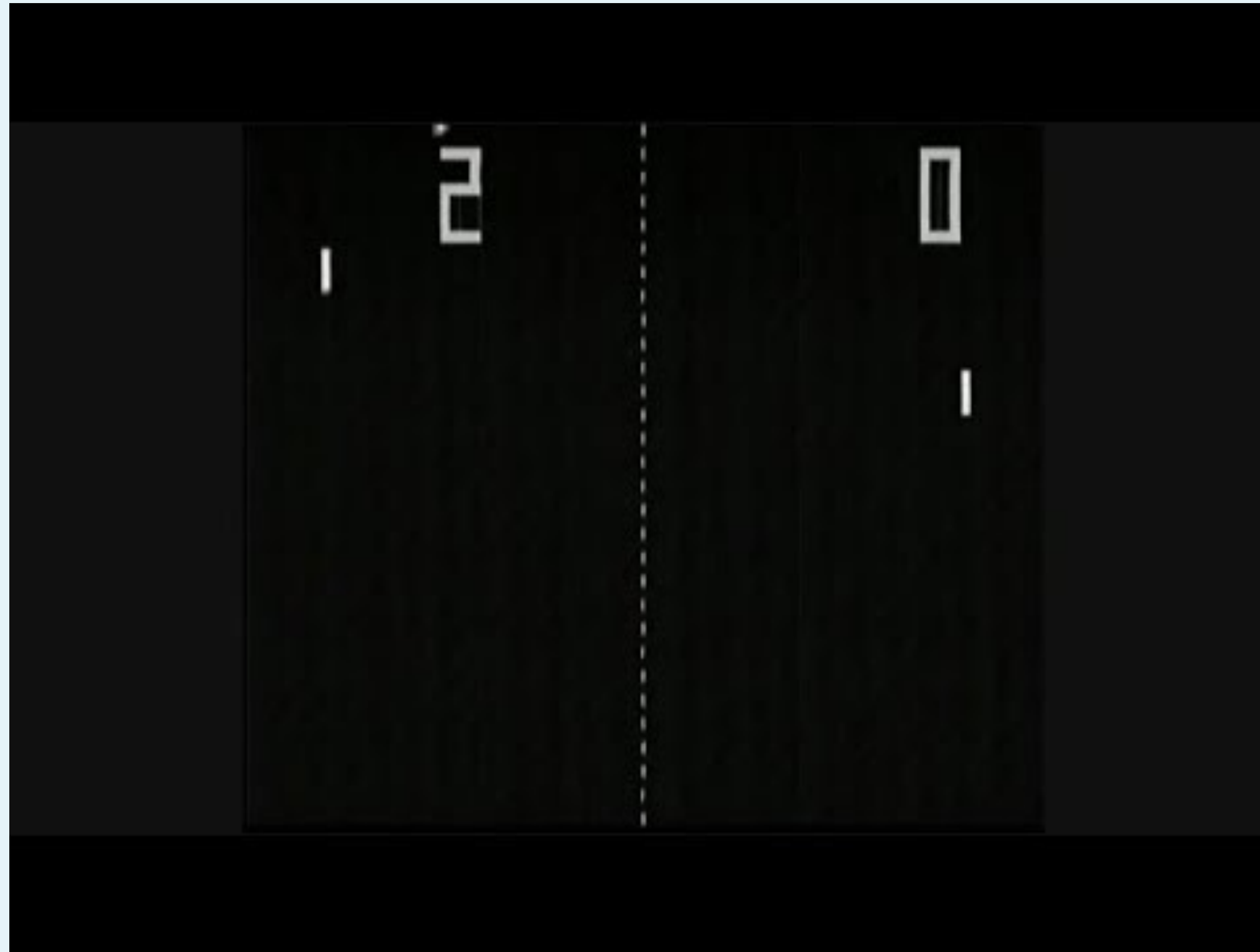
Spacewar! - PDP-1 - One of the First Video Games (MIT 1962) - (2007) The Dot Eaters, CC-BY 3.0, <https://www.youtube.com/watch?v=Rmvp4Hktv7U>

Atari PONG(1972)



pong74ls, CC 表示 3.0, <https://commons.wikimedia.org/w/index.php?curid=79121297>

Atari PONG(1972)



<https://www.youtube.com/watch?v=fhd7FfGCdCo>

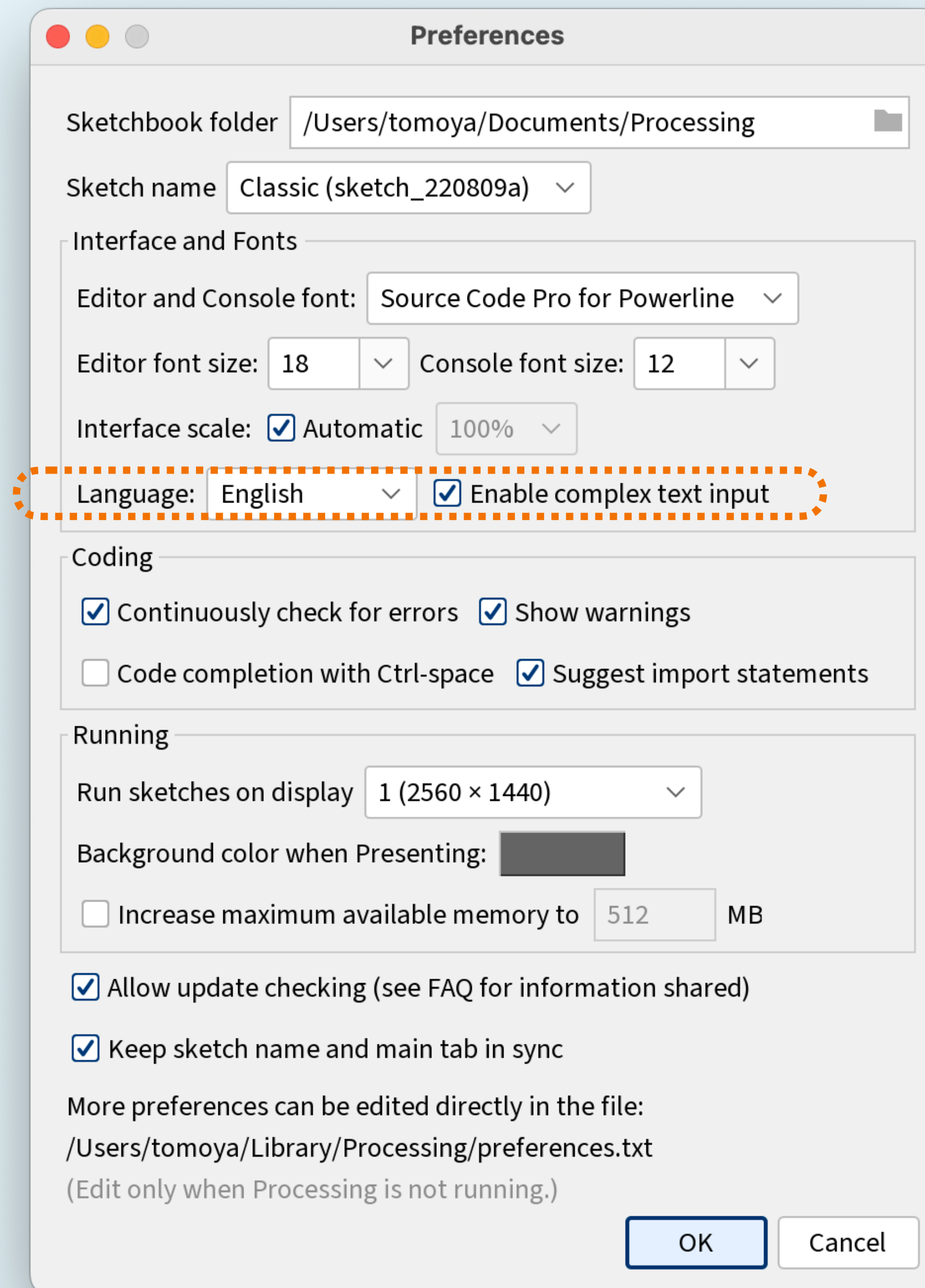
Processingで作る 一人用PONG

Arduinoでのコーディングとの違い

- Arduinoで使う言語（C、 C++）とProcessingで使う言語(Java)はよく似ていますが、基本的にはぜんぜん別の言語です

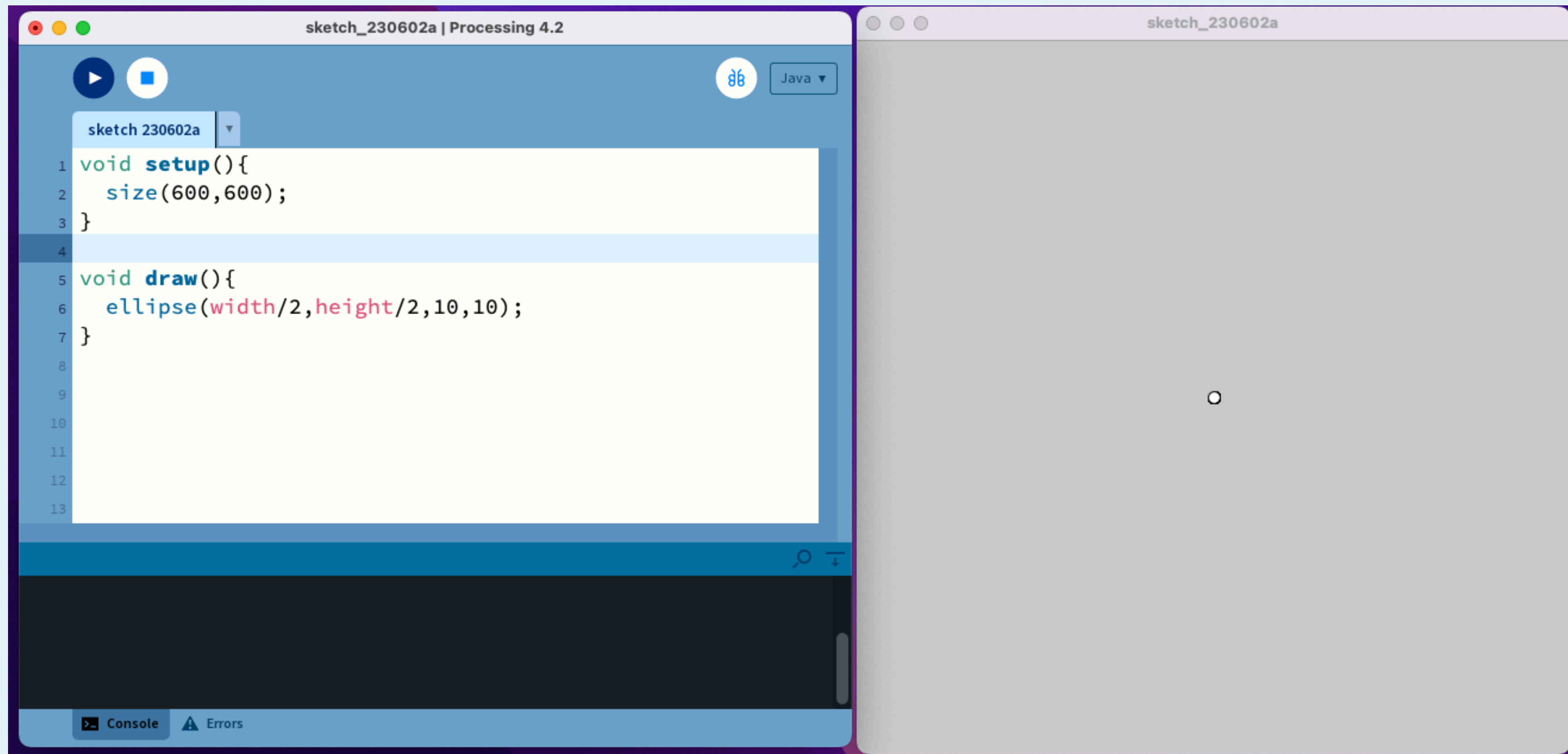
	C++	Java
ライブラリの読み込み	<code>#include <Library.h></code>	<code>import library.*;</code>
定数の定義	<code>const float pi = 3.14;</code>	<code>final float pi = 3.14;</code>
ブーリアン型の 型名	<code>bool answer = true;</code>	<code>boolean answer = true;</code>

個人的に混乱しがちな違い3選



日本語フォント表示に対応するために “Enable complex text input” をオンにしてアプリ再起動

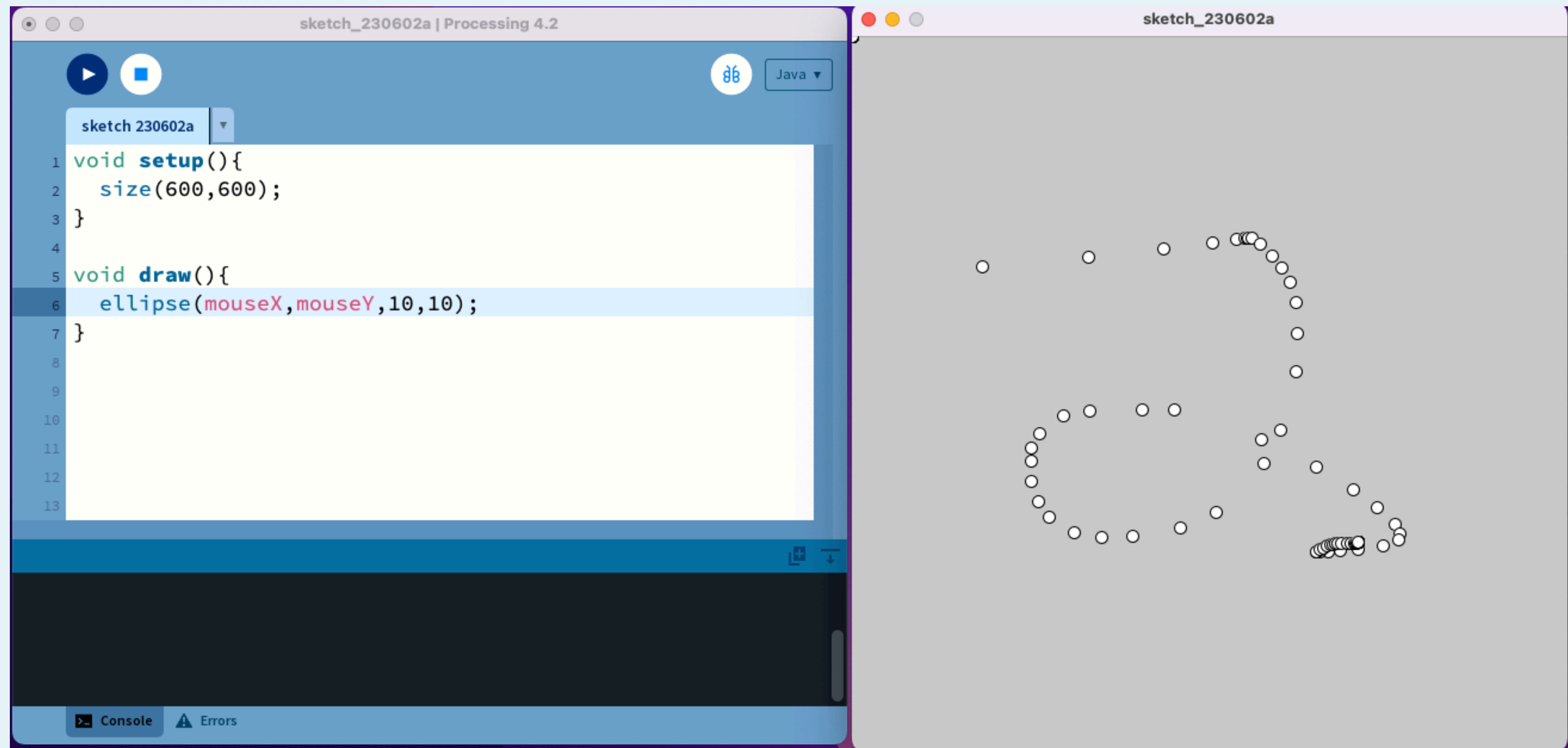
Processingの初歩



sizeでアプリケーションのサイズを決める（ピクセル単位）

ellipse(x,y,w,h)で円を描く

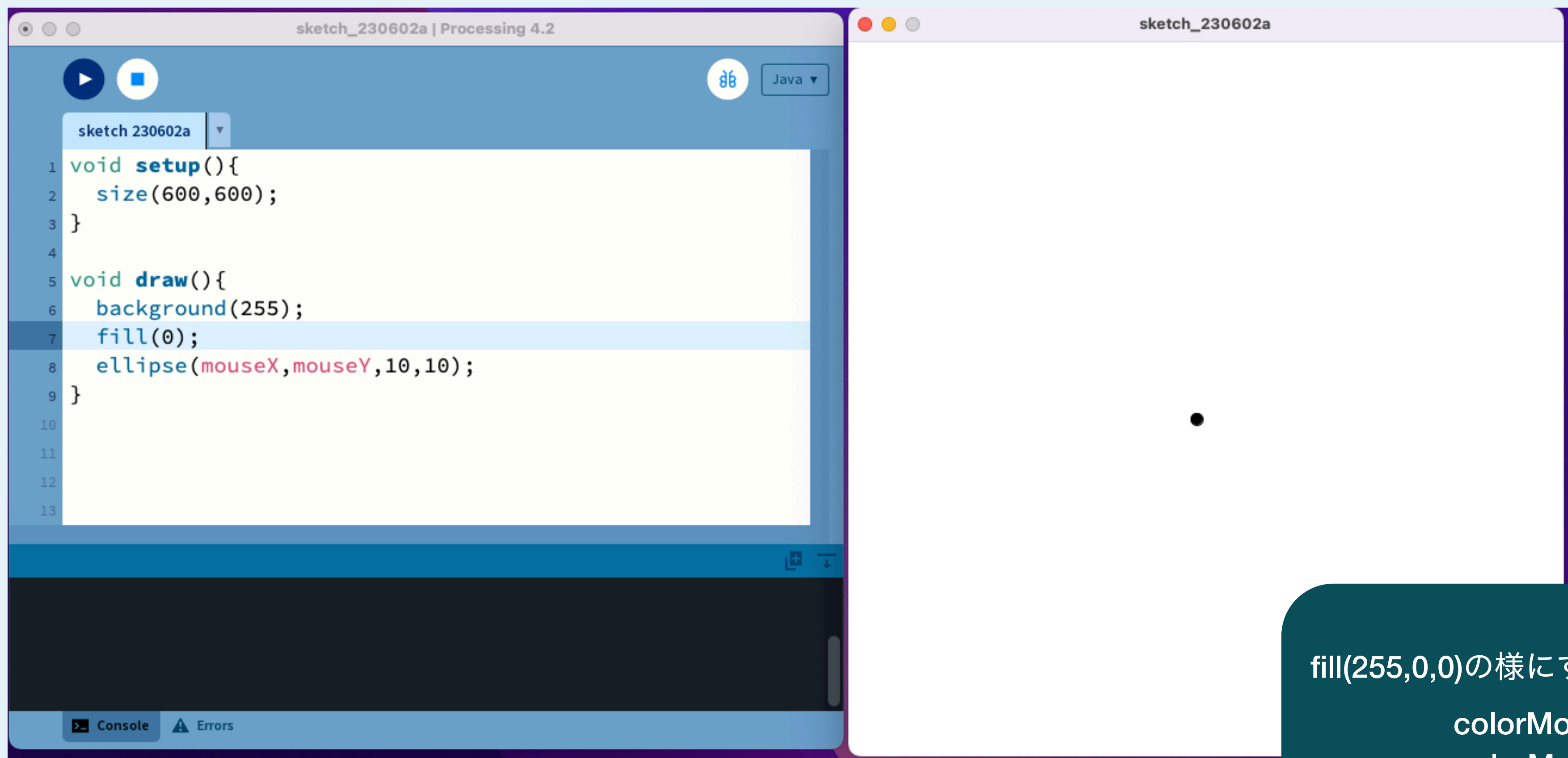
Processingの初歩



width,height,mouseX,mouseYなど組み込みで使える値がいくつかある

前のフレームに描いたものがそのまま残ってしまう

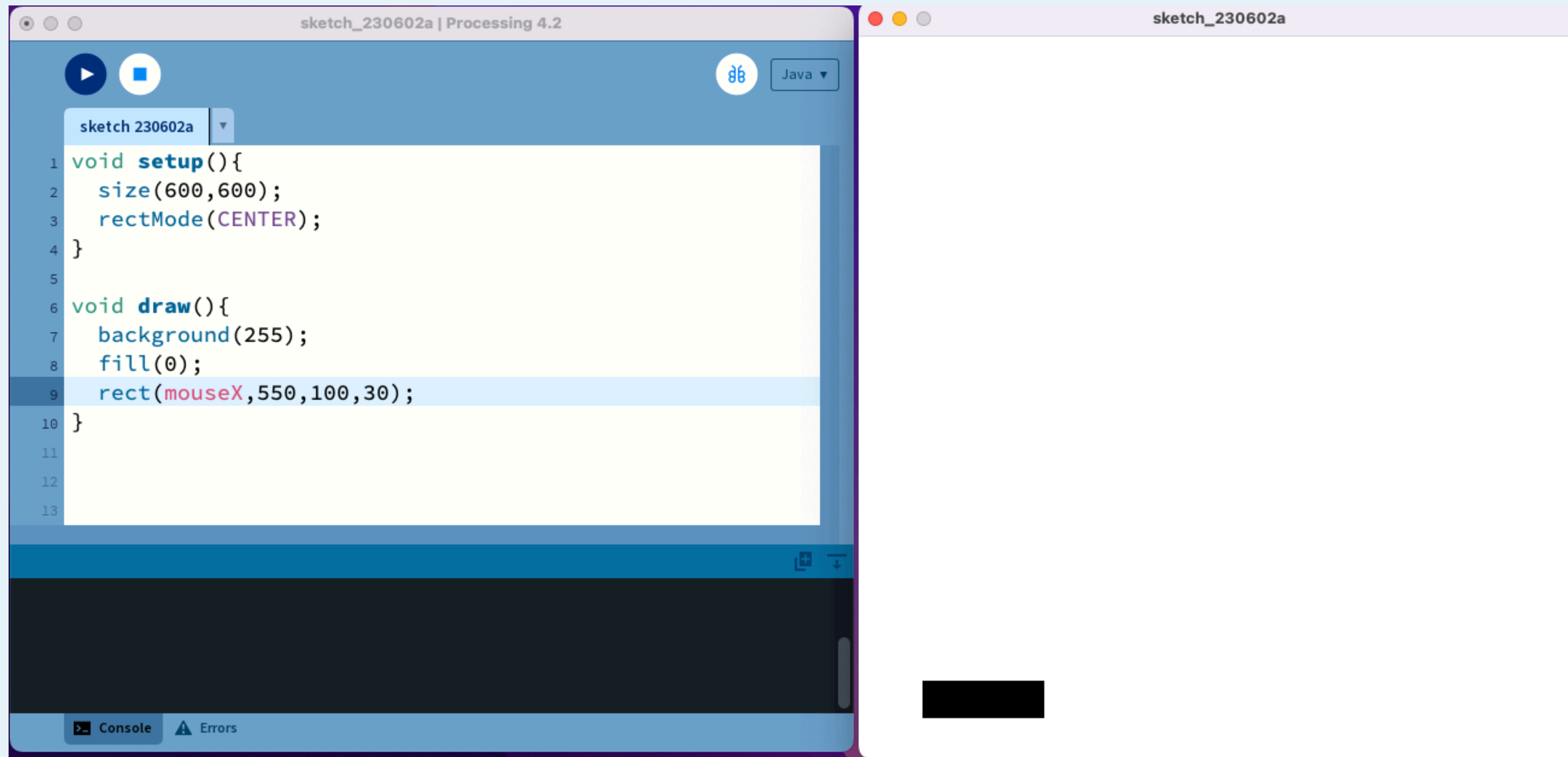
Processingの初歩



background(c)は背景の塗りつぶし実行
fill(c)は以後のシェイプの塗りつぶし色指定
0が黒、128でグレー、255で白

fill(255,0,0)の様にすれば色の指定も可
colorMode(RGB)
colorMode(HSB)
など意味はモードによって違う

マウスに沿って動くバー



rectMode(CENTER)を呼んだ後は
rect(center_x,center_y,width,height)

オブジェクト指向プログラミング

データの抽象化

- 整数(int)、実数(float)や真理値(bool)など、バラバラのデータを扱うには限界がある
- あるデータ型を意味のある単位でひとまとめにして名前をつける：構造体、合成型

```
struct Position{  
    float x;  
    float y;  
}  
void main(){  
    Position pos = {0.0,0.0};  
    draw_circle(pos.x,pos.y);  
}
```

C++での疑似コード

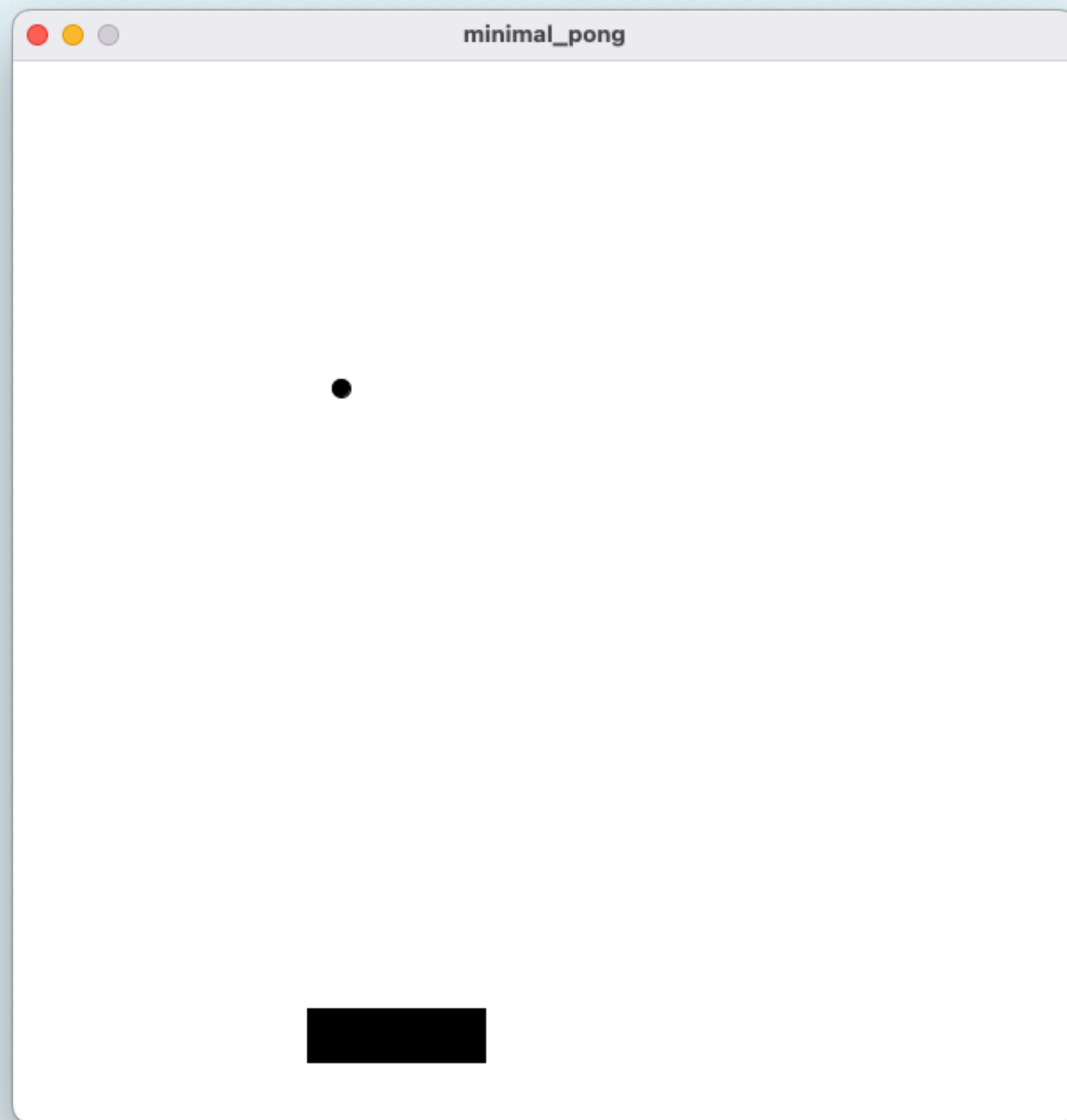
(今回使うProcessingのJava言語にはstructキーワードは存在しない)

オブジェクト指向

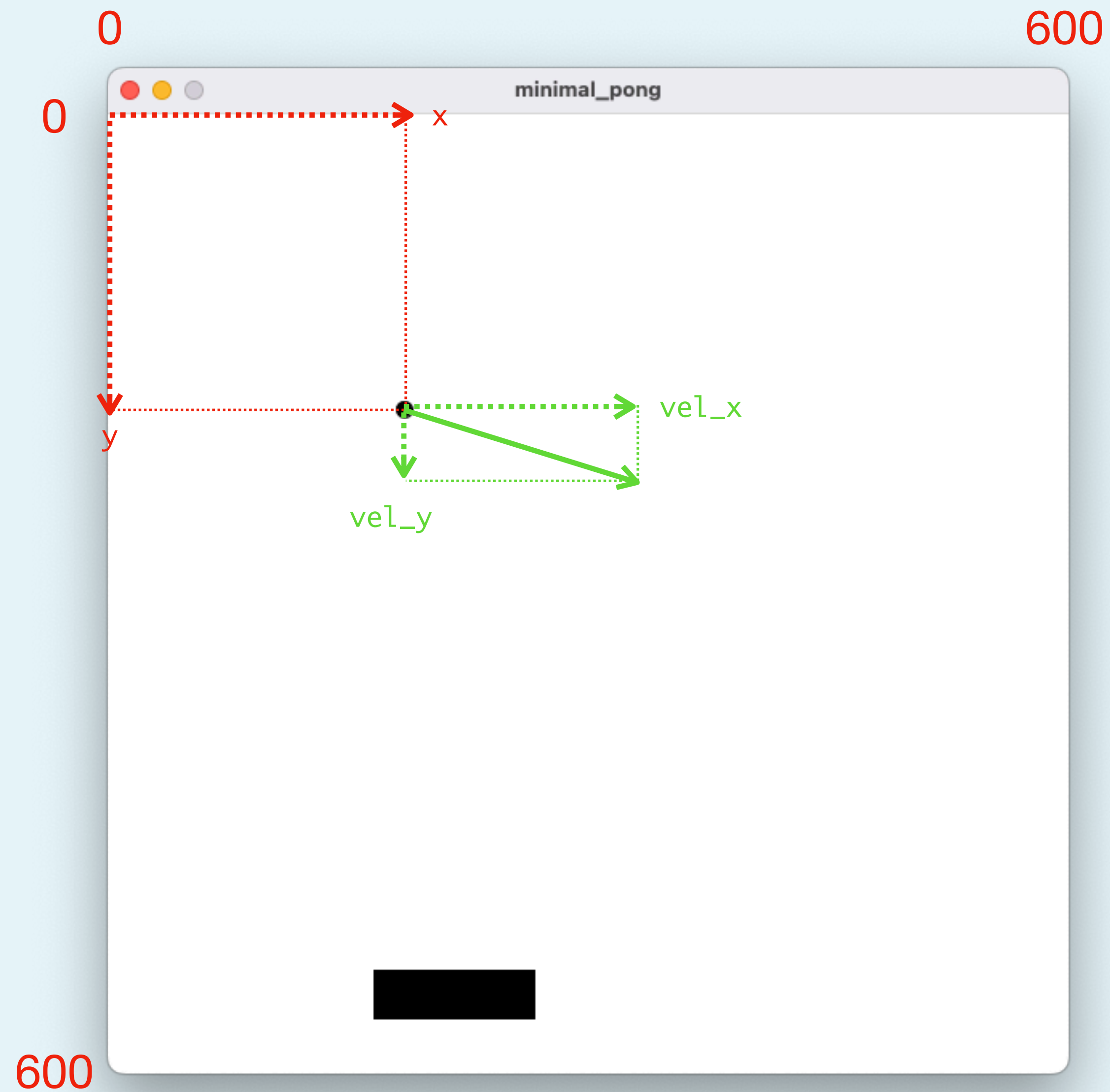
- ある時期以後のプログラミング言語では、**データの集合とそれに紐づく関数をペアにしたデータ構造**が重要視されてきた
 - 現実の物事のふるまいを記述しやすくなる
- 例えば、オブジェクト、クロージャ、モナド、等々。
- C++やJavaでは、オブジェクト指向の1つ、クラス(class)という仕組みを使う
- 今回は、「跳ね返って動くボール」を例に考えよう

用語集

- クラス(`Ball`) : 含まれるデータの集合と、それを参照する関数定義の集合
- インスタンス(`ball1, ball2...`) : クラスを実際に使うときに実体化したもの
- メンバ変数 (`x, y, _vel_x, vel_y`) クラスが持つデータの中身
- メンバ関数/メソッド (`update_pos, reflect_horizontal`) クラスが使える関数
- コンストラクタ : インスタンスを作った初期化時に最初に呼ばれる関数
- デストラクタ : インスタンスを消去するときに呼ばれる関数



- x, y : 現在位置
- vel_x, vel_y : 次のフレームでそれぞれ何ピクセル移動するか (速度)



Ballクラス

```
class Ball {  
    float x;  
    float y;  
    private float vel_x;  
    private float vel_y;  
    Ball() {  
        this.x = width/2;  
        this.y = height/2;  
        this.vel_x = 1;  
        this.vel_y = 1;  
    }  
    public void update_pos() {  
        this.x += this.vel_x;  
        this.y += this.vel_y;  
    }  
}
```

```
Ball ball = new Ball();  
void setup(){  
    size(600,600);  
    rectMode(CENTER);  
}  
  
void draw(){  
    ball.update_pos();  
    background(255);  
    fill(0);  
    rect(mouseX,550,100,30);  
    ellipse(ball.x,ball.y,10,10);  
}
```

Ballクラス

```
class Ball {                                     クラス定義
    float x;                                     メンバ変数宣言
    float y;
    private float vel_x;
    private float vel_y;
    Ball() {                                     コンストラクタ
        this.x = width/2;
        this.y = height/2;
        this.vel_x = 1;
        this.vel_y = 1;
    }
    public void update_pos() {                  メンバ関数宣言
        this.x += this.vel_x;
        this.y += this.vel_y;
    }
}
```

```
Ball ball = new Ball();
void setup(){
    size(600,600);
    rectMode(CENTER);
}

void draw(){
    ball.update_pos();
    background(255);
    fill(0);
    rect(mouseX,550,100,30);
    ellipse(ball.x,ball.y,10,10);
}
```

Ballクラス

privateをつけると、
クラスの中でのみ参照できる

インスタンスから呼ぶメンバ
関数はpublicを付ける

```
class Ball {  
    float x;  
    float y;  
    private float vel_x;  
    private float vel_y;  
    Ball() {  
        this.x = width/2;  
        this.y = height/2;  
        this.vel_x = 1;  
        this.vel_y = 1;  
    }  
    public void update_pos() {  
        this.x += this.vel_x;  
        this.y += this.vel_y;  
    }  
}
```

```
Ball ball = new Ball();  
void setup(){  
    size(600,600);  
    rectMode(CENTER);  
}
```

new Classname()でインスタンス
を生成(コンストラクタ実行)

```
void draw(){  
    ball.update_pos();  
    background(255);  
    fill(0);  
    rect(mouseX,550,100,30);  
  
    ellipse(ball.x,ball.y,10,10);  
}
```

インスタンスに対してメンバ
関数を実行

メンバ変数をもとに情報を
取得する

Ballクラス

```
class Ball {
    float x;
    float y;
    private float vel_x;
    private float vel_y;
    Ball() {
        this.x = width/2;
        this.y = height/2;
        this.vel_x = random(-2,2);
        this.vel_y = random(-2,2);
    }
    public void update_pos() {
        this.x += this.vel_x;
        this.y += this.vel_y;
    }
}
Ball ball = new Ball();
Ball ball2 = new Ball();

void setup(){
    size(600,600);
    rectMode(CENTER);
}

void draw(){
    ball.update_pos();
    ball2.update_pos();
    background(255);
    fill(0);
    rect(mouseX,550,100,30);

    ellipse(ball.x,ball.y,10,10);
    ellipse(ball2.x,ball2.y,10,10);
}
```

Ballを2個インスタンス化してみる、初期速度をランダムにしてみる

Ballクラス

```
class Ball {
    float x;
    float y;
    private float vel_x;
    private float vel_y;
    Ball() {
        this.x = width/2;
        this.y = height/2;
        this.vel_x = random(-2,2);
        this.vel_y = random(-2,2);
    }
    public void update_pos() {
        this.x += this.vel_x;
        this.y += this.vel_y;
        if (this.x < 0 || this.x > width){
            reflect_horizontal();
        }
    }
    void reflect_horizontal(){
        this.vel_x = -this.vel_x;
    }
}

Ball ball = new Ball();
Ball ball2 = new Ball();

void setup(){
    size(600,600);
    rectMode(CENTER);
}

void draw(){
    ball.update_pos();
    ball2.update_pos();
    background(255);
    fill(0);
    rect(mouseX,550,100,30);

    ellipse(ball.x,ball.y,10,10);
    ellipse(ball2.x,ball2.y,10,10);
}
```

跳ね返り：x方向にはみ出したら速度を正負反転させる

やってみよう

- 一旦Ballは1個に戻して考えよう
- 垂直方向の反射（reflect_vertical）を実装しよう
 - 上側の壁にぶつかったときに反射するようにupdate_posを変えよう
 - マウス操作のブロックにぶつかったときにも反射する様にしよう
- ボールがブロックをすり抜けたらゲームオーバーになるようにしよう
- マウスをクリックしたらゲームをリセットできるようにしよう

ゲーム性を追求しよう

- 反射するたびにBallの速度が上がるようにしよう
- バーに当たった位置によって反射の角度が変わるようにしよう
- 2人対戦バージョンにしてみよう

```

class Ball {
  float x;
  float y;
  float vel_x;
  float vel_y;
  Ball() {
    this.x = width/2;
    this.y = height/2;
    this.vel_x = random(0, 1.0) > 0.5 ? -3 : 3;
    this.vel_y = random(0, 1.0) > 0.5 ? -3 : 3;
  }
  public void update_pos() {
    this.x += this.vel_x;
    this.y += this.vel_y;
    if (this.x < 0 || this.x > width) {
      this.reflect_horizontal();
    }
    if (this.y < 0 ) {
      this.reflect_vertical();
    }
  }
  void reflect_horizontal() {
    this.vel_x = -this.vel_x*1.1;
  }
  public void reflect_vertical() {
    this.vel_y = -this.vel_y*1.1;
  }
}

final float bar_width = 100;
final float bar_y = 550;
float bar_pos = 0.;
boolean gameover = false;
Ball ball = new Ball();

```

```

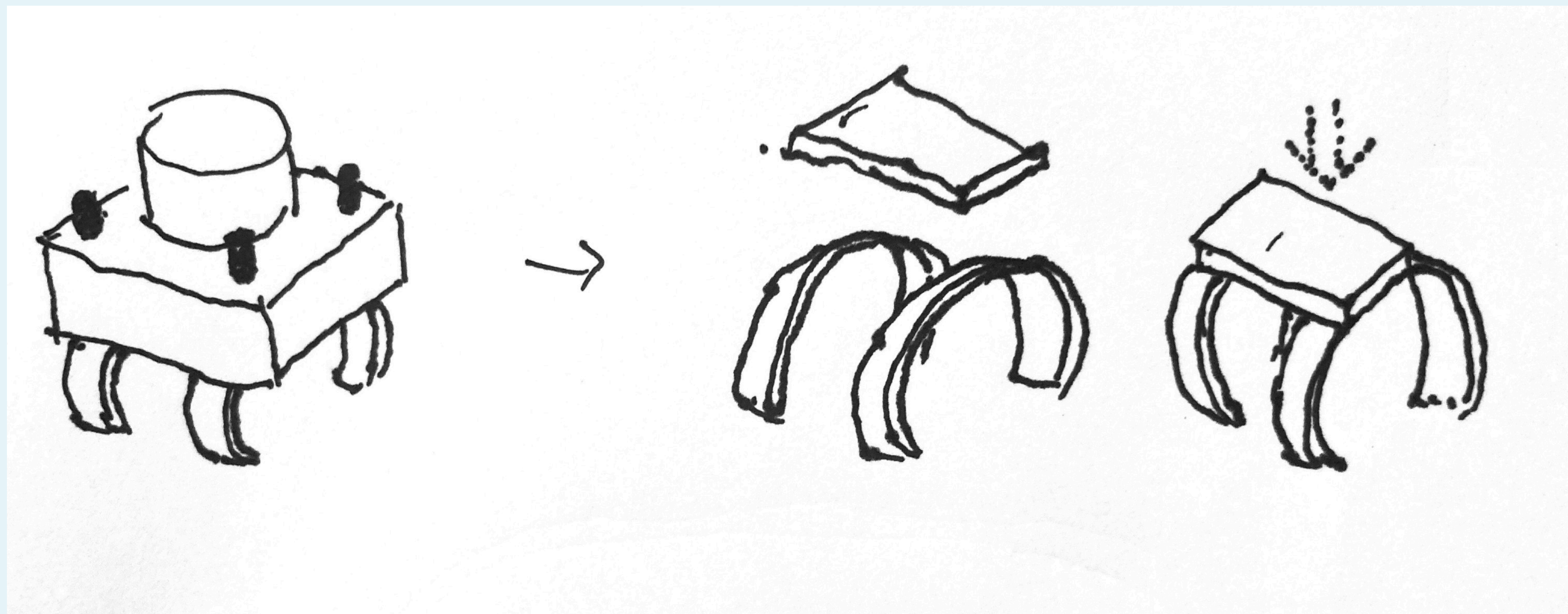
void setup() {
  size(600, 600);
  rectMode(CENTER);
}

void draw() {
  ball.update_pos();
  if (ball.y > height) {
    gameover = true;
  }
  if (gameover) {
    textSize(24);
    text("game over", width/2, height/2);
  } else {
    bar_pos = mouseX;
    if (ball.y > bar_y &&
        ball.x > bar_pos - bar_width/2 &&
        ball.x < bar_pos + bar_width/2) {
      ball.reflect_vertical();
    }
    background(255);
    fill(0);
    rect(bar_pos, bar_y, bar_width, 30);
    ellipse(ball.x, ball.y, 10, 10);
  }
}

void mousePressed() {
  if (gameover) {
    gameover = false;
    ball = new Ball();
  }
}

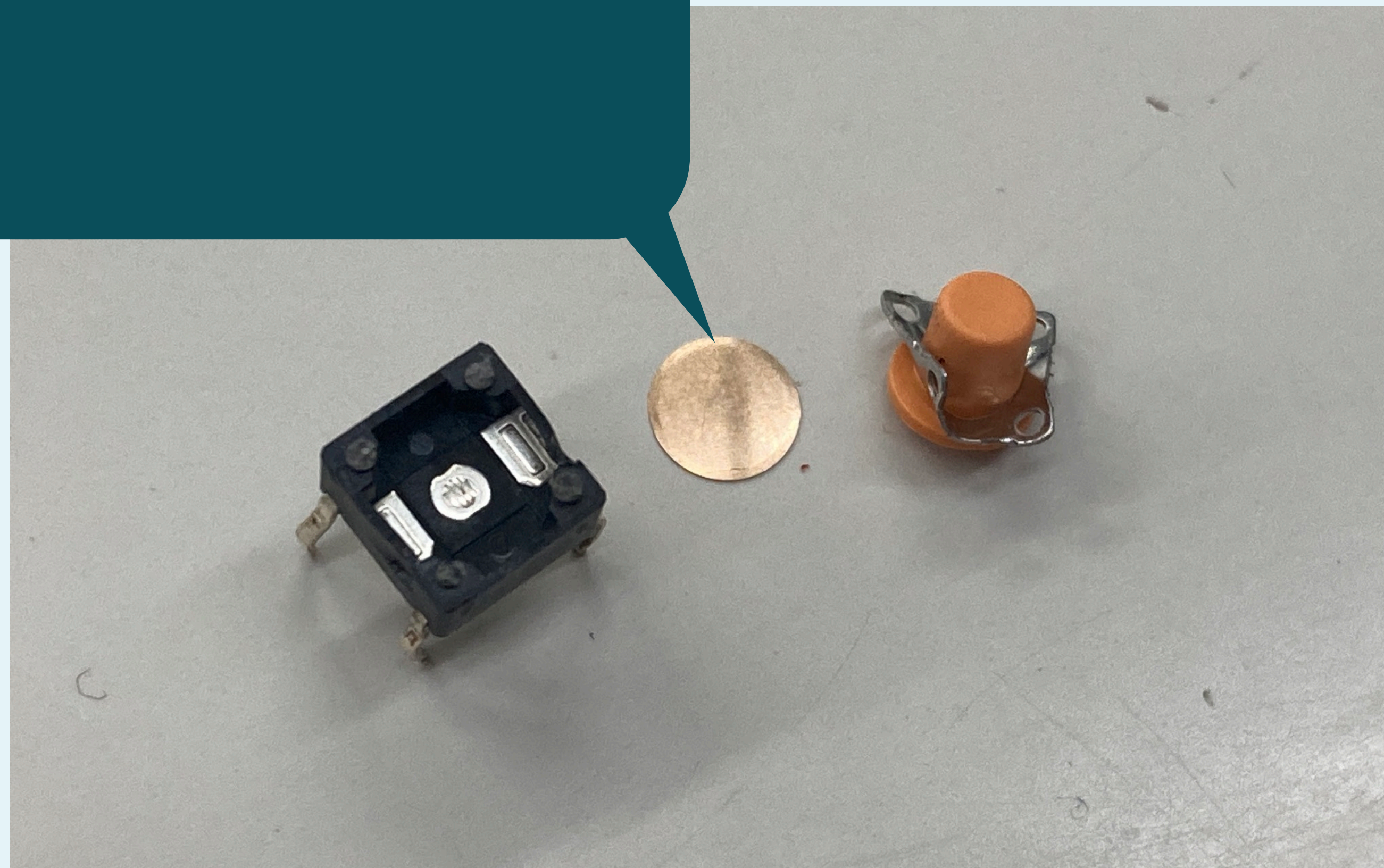
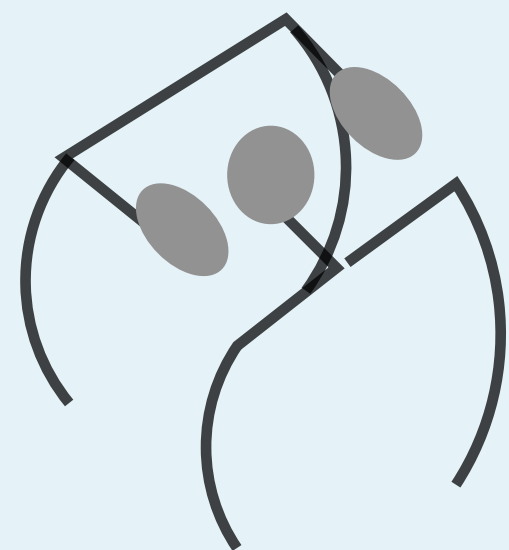
```

可変抵抗とタクトスイッチ

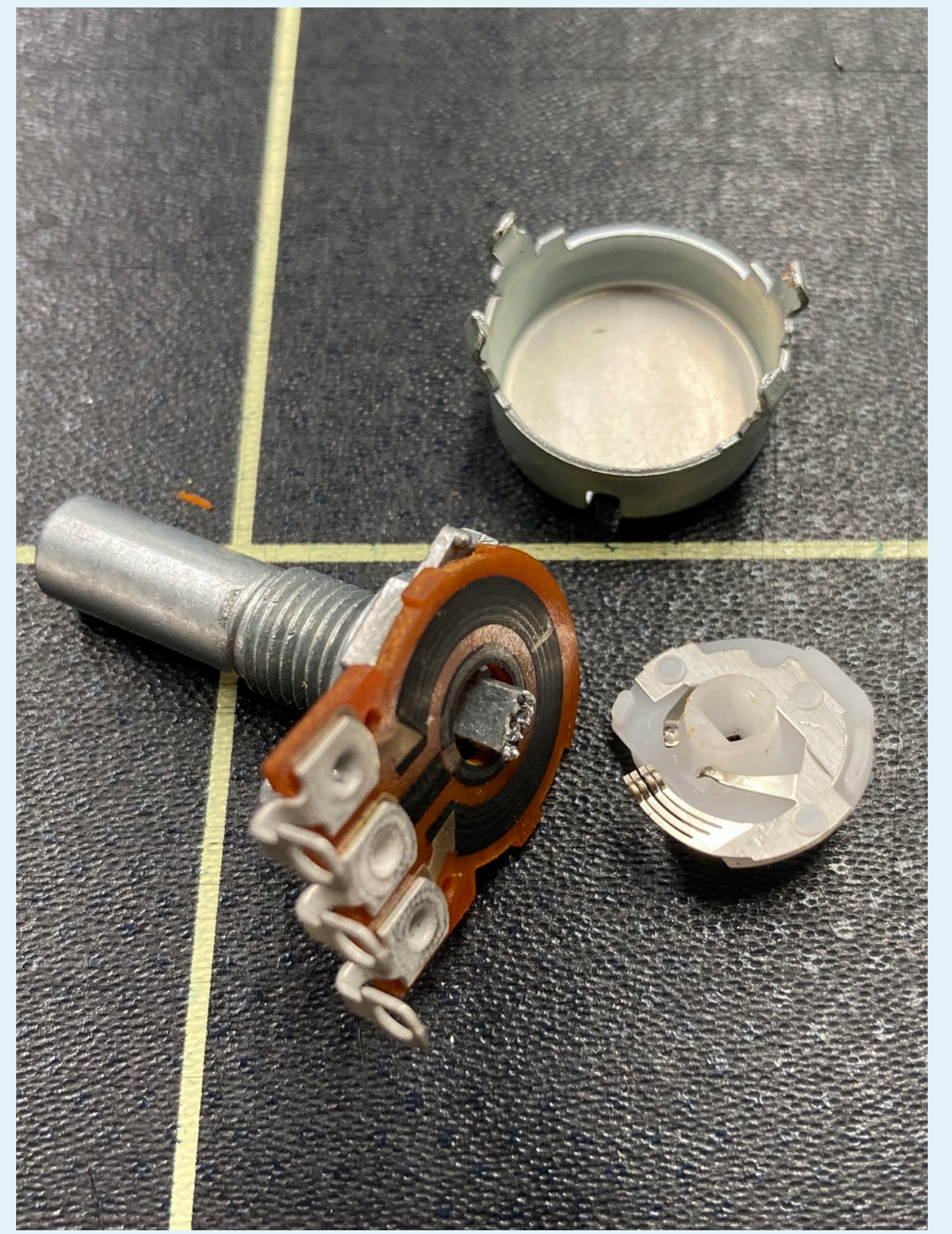
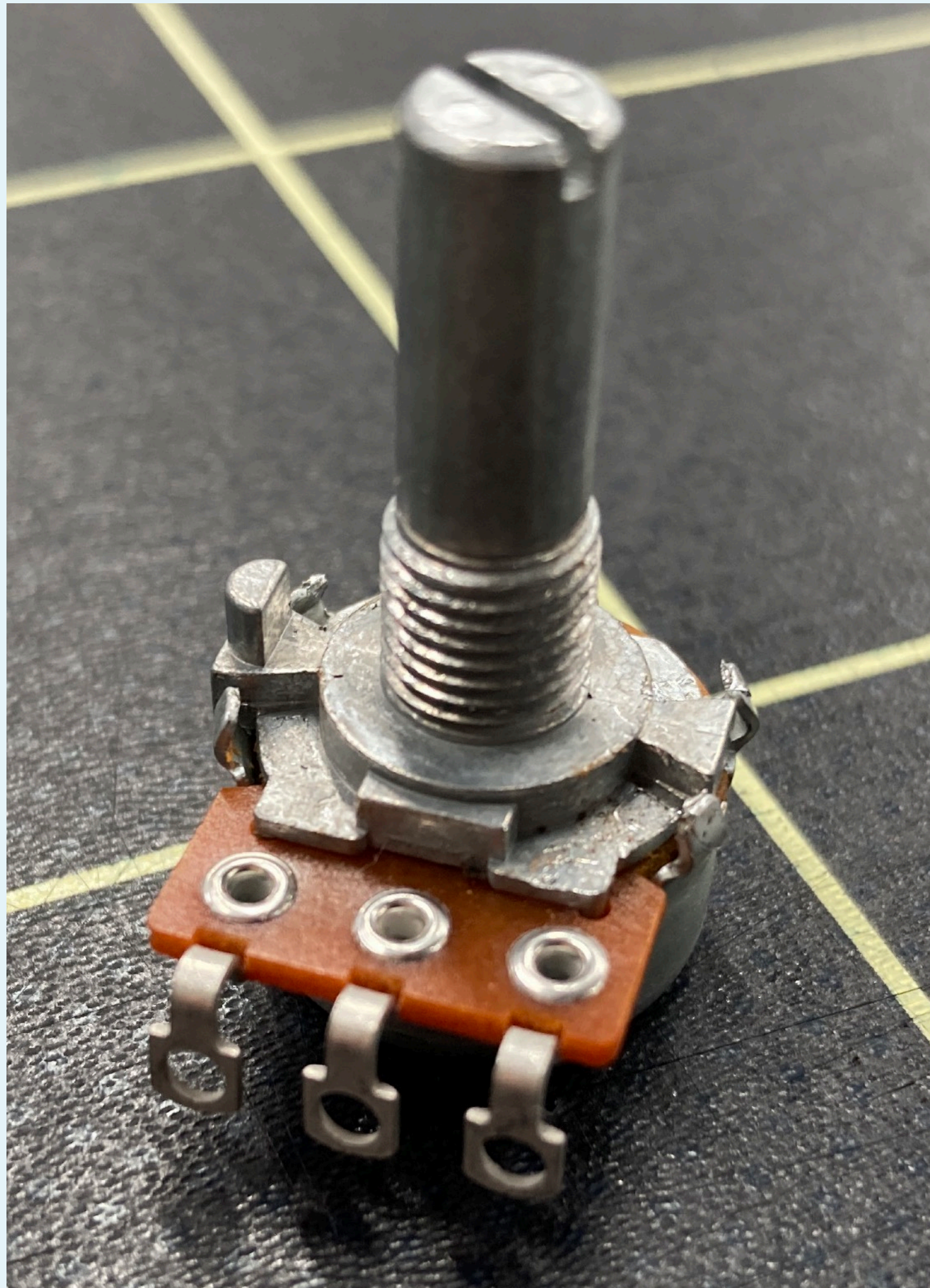


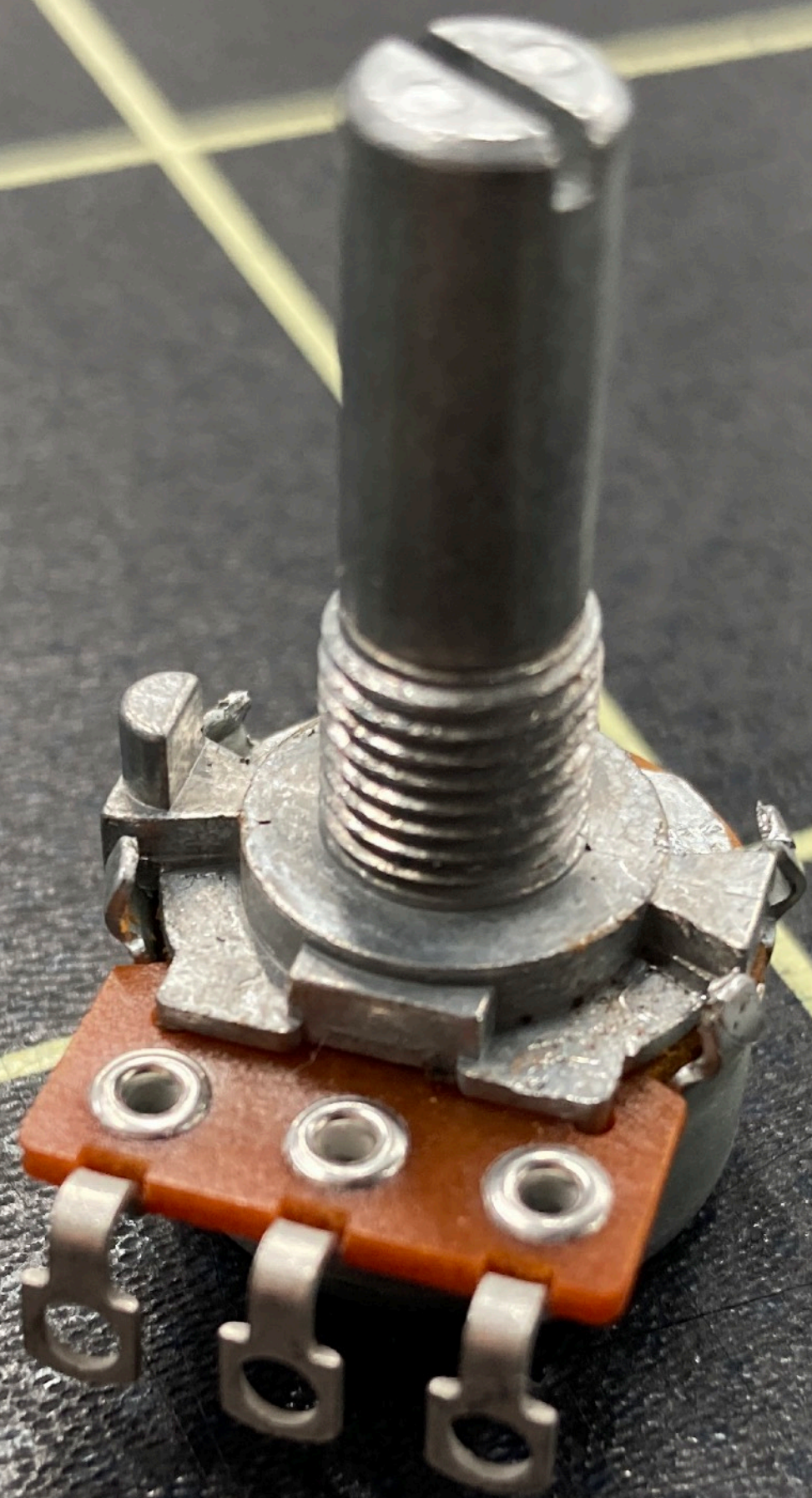
タクトスイッチの構造

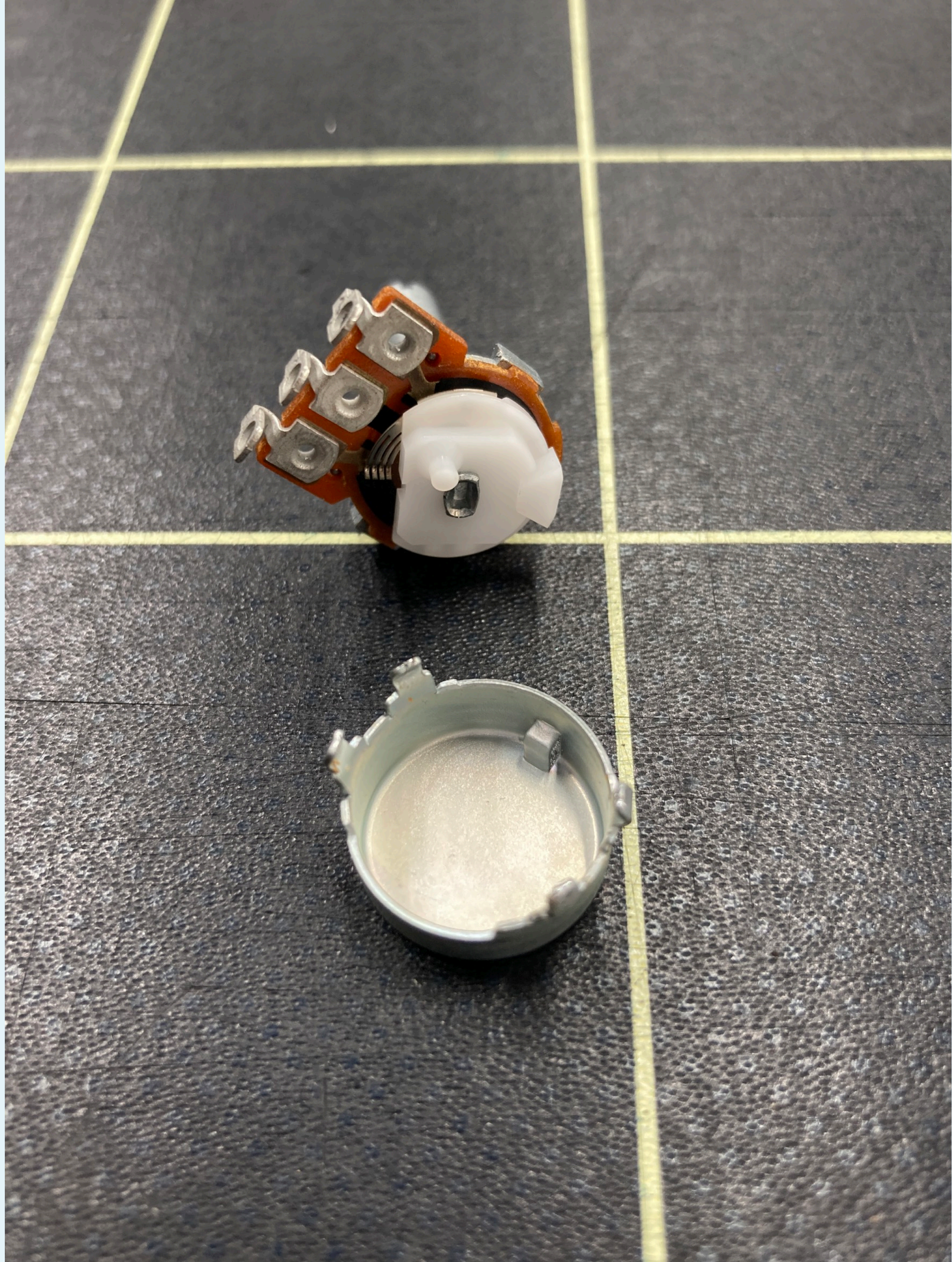
若干膨らんでいて、押すとペコッと凹む



可變抵抗







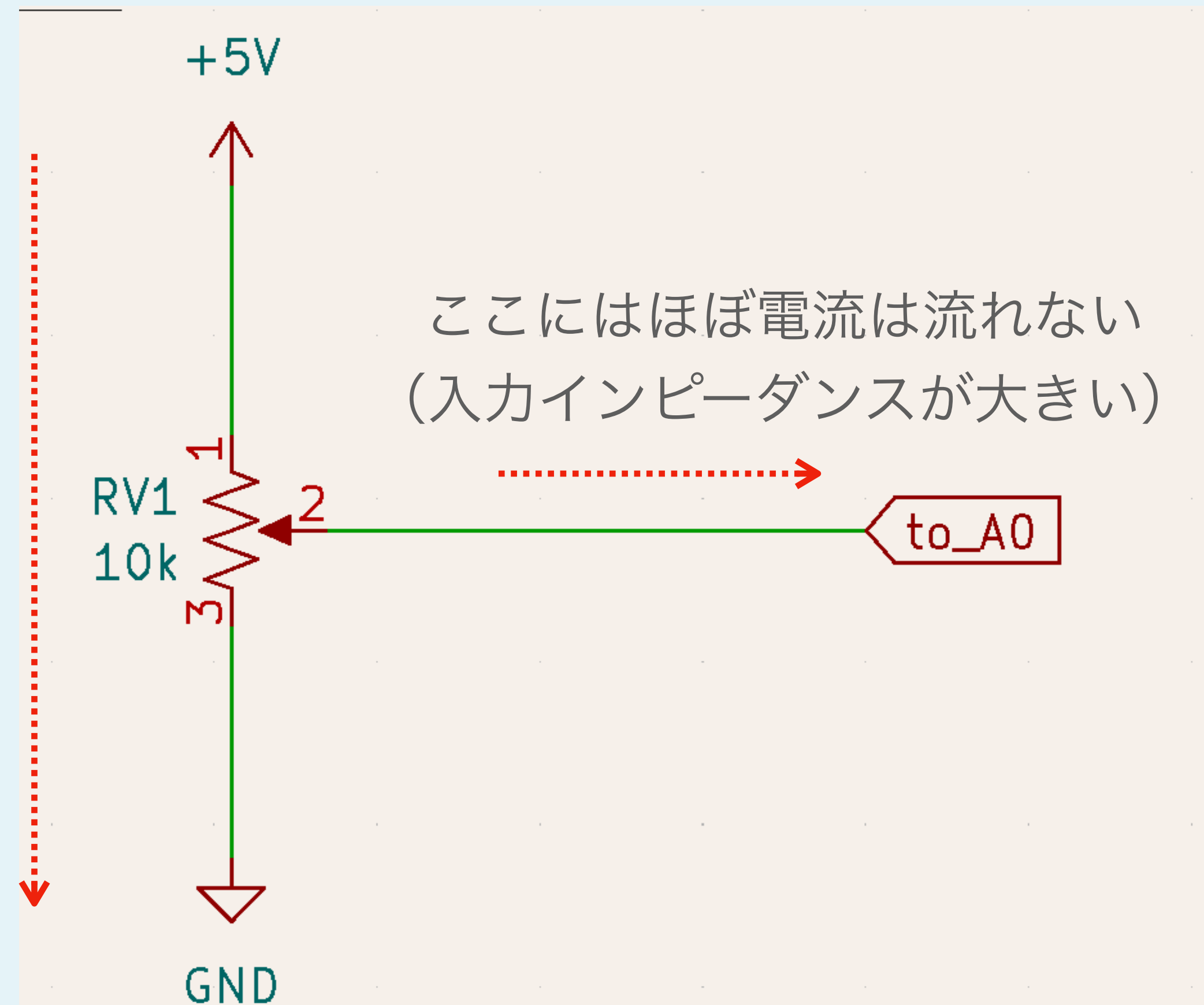


可変抵抗

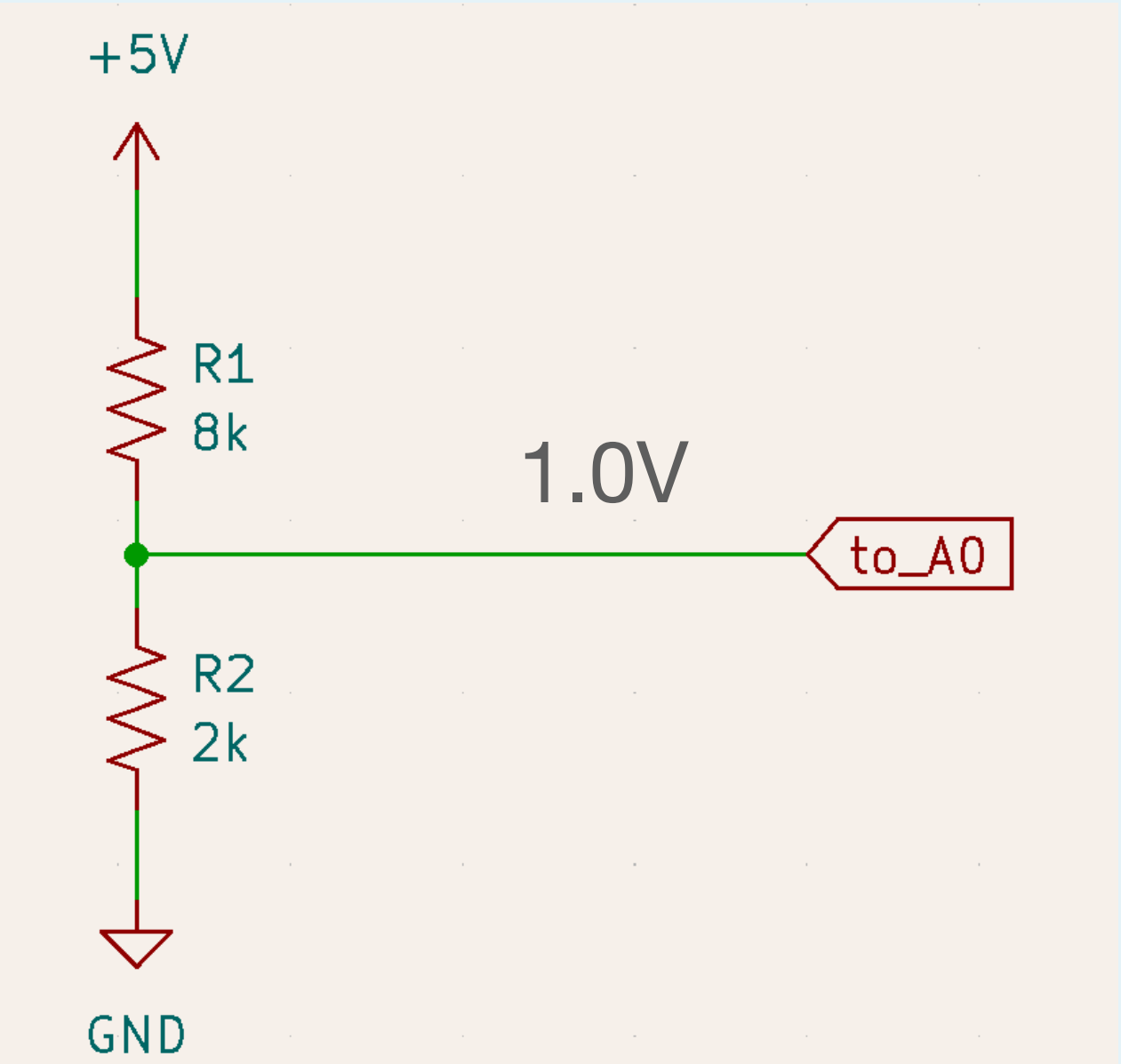
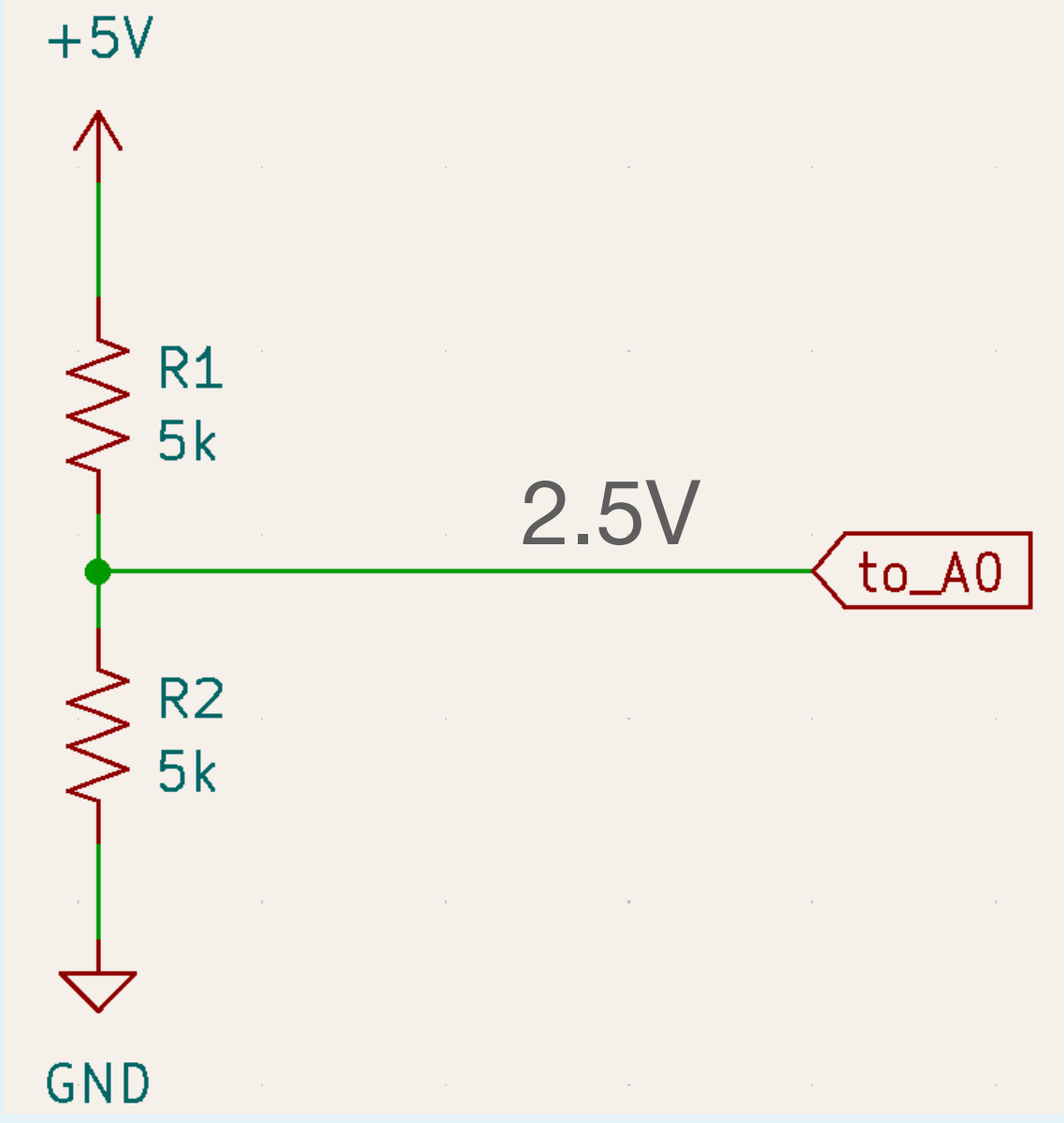
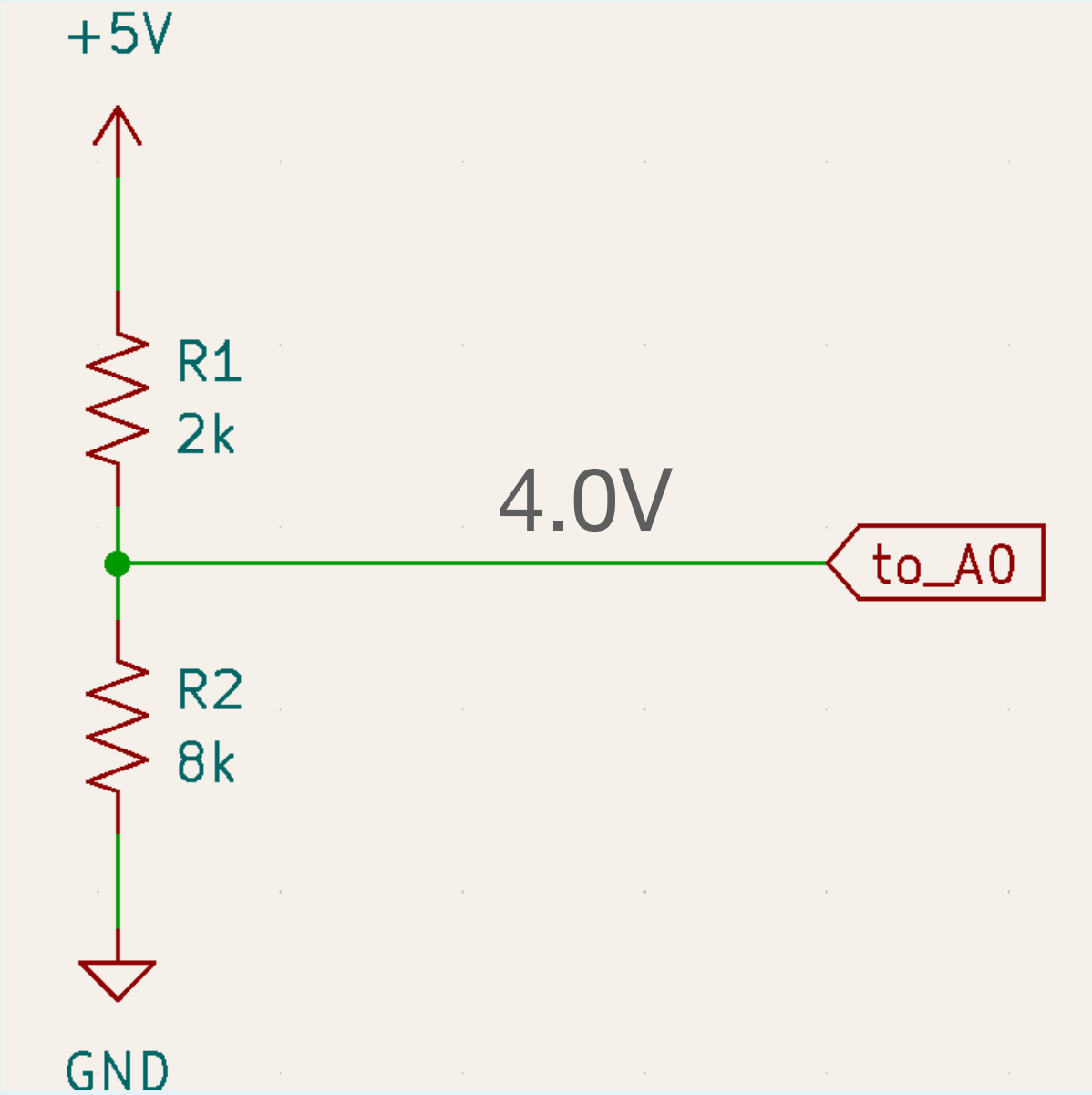
電流はほぼこの経路でのみ流れる

$$V=IR$$

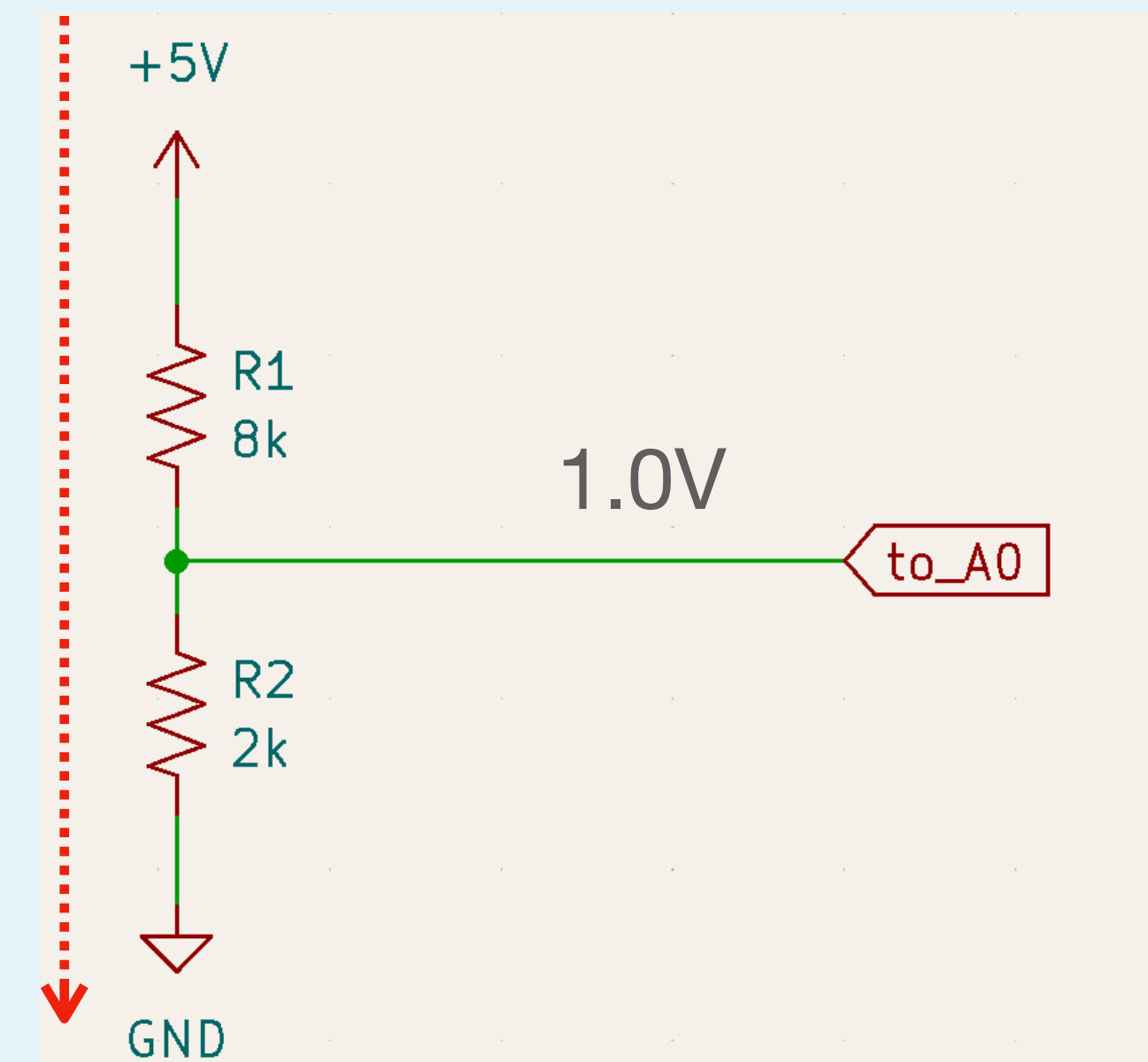
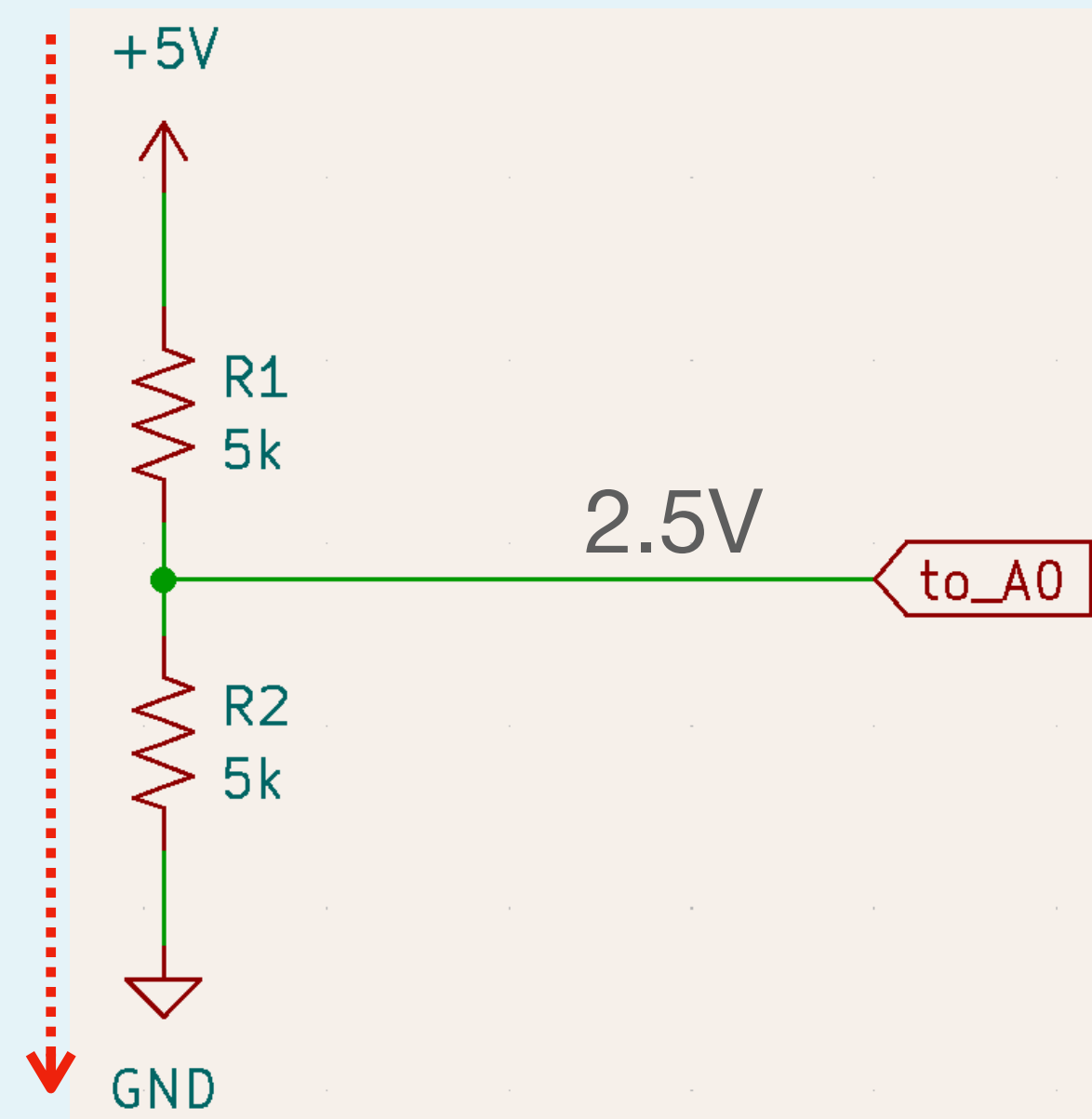
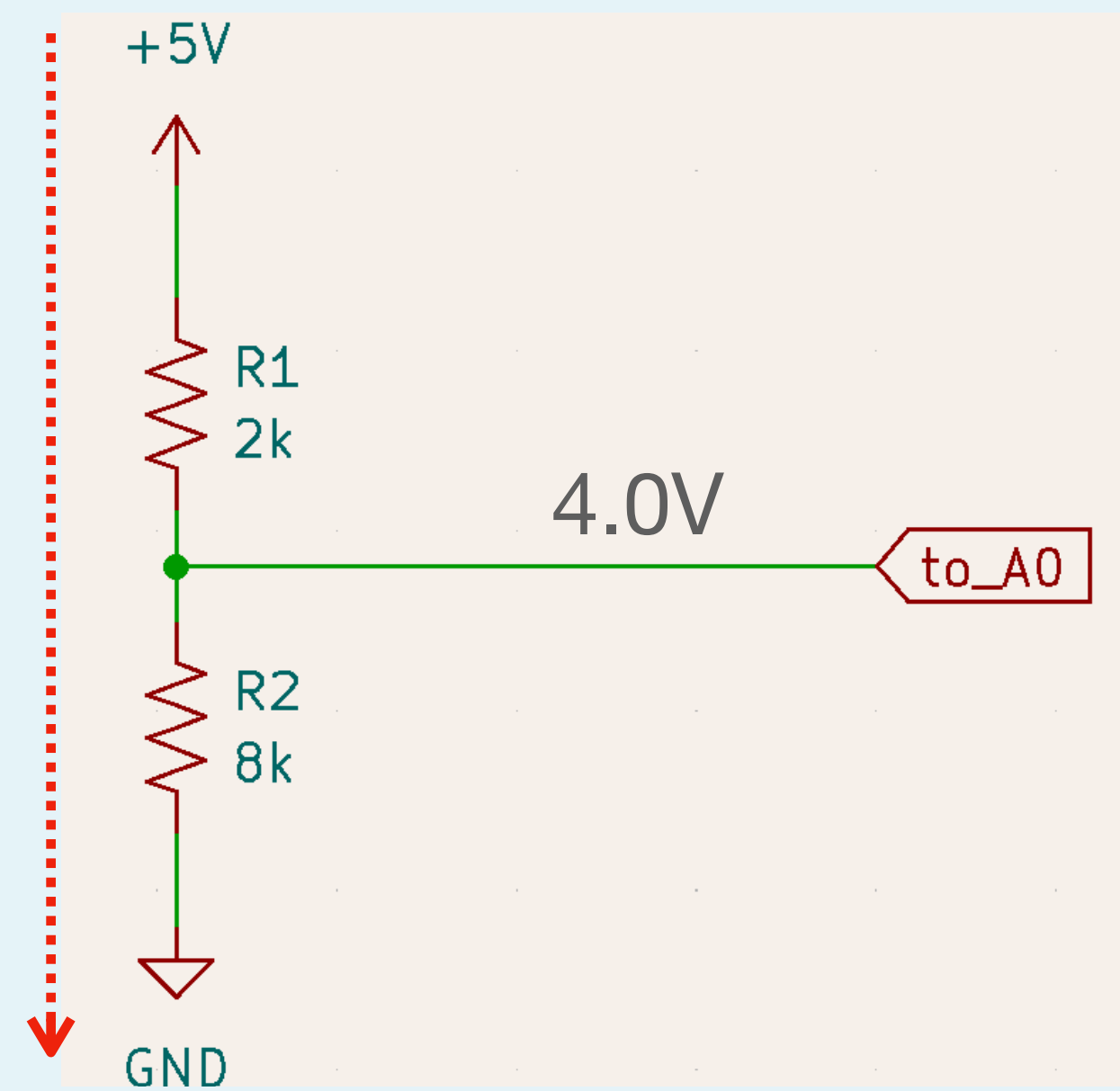
$$I = 5.0\text{V} / 10000\Omega \\ = 0.5\text{mA}$$



可變抵抗

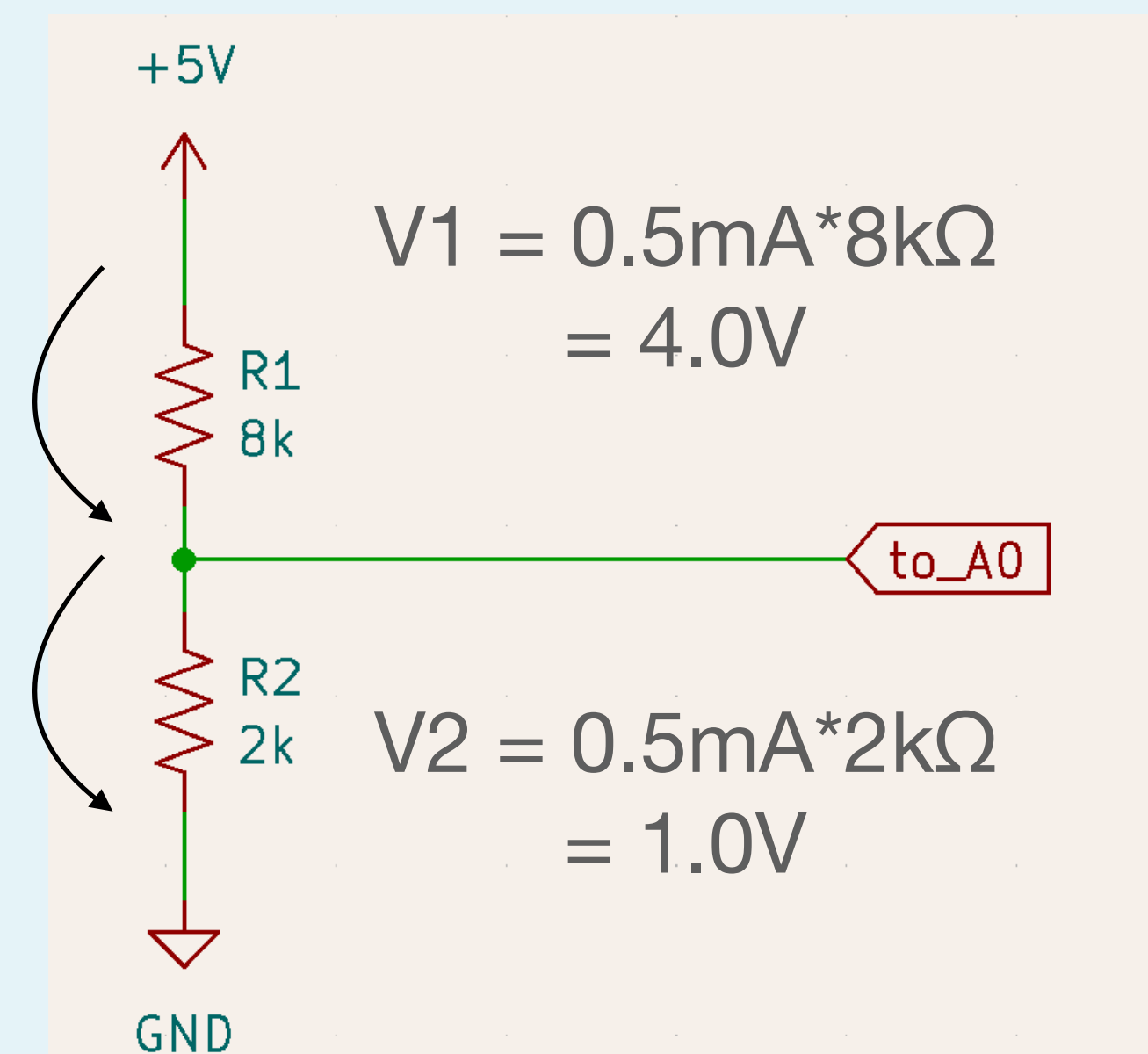
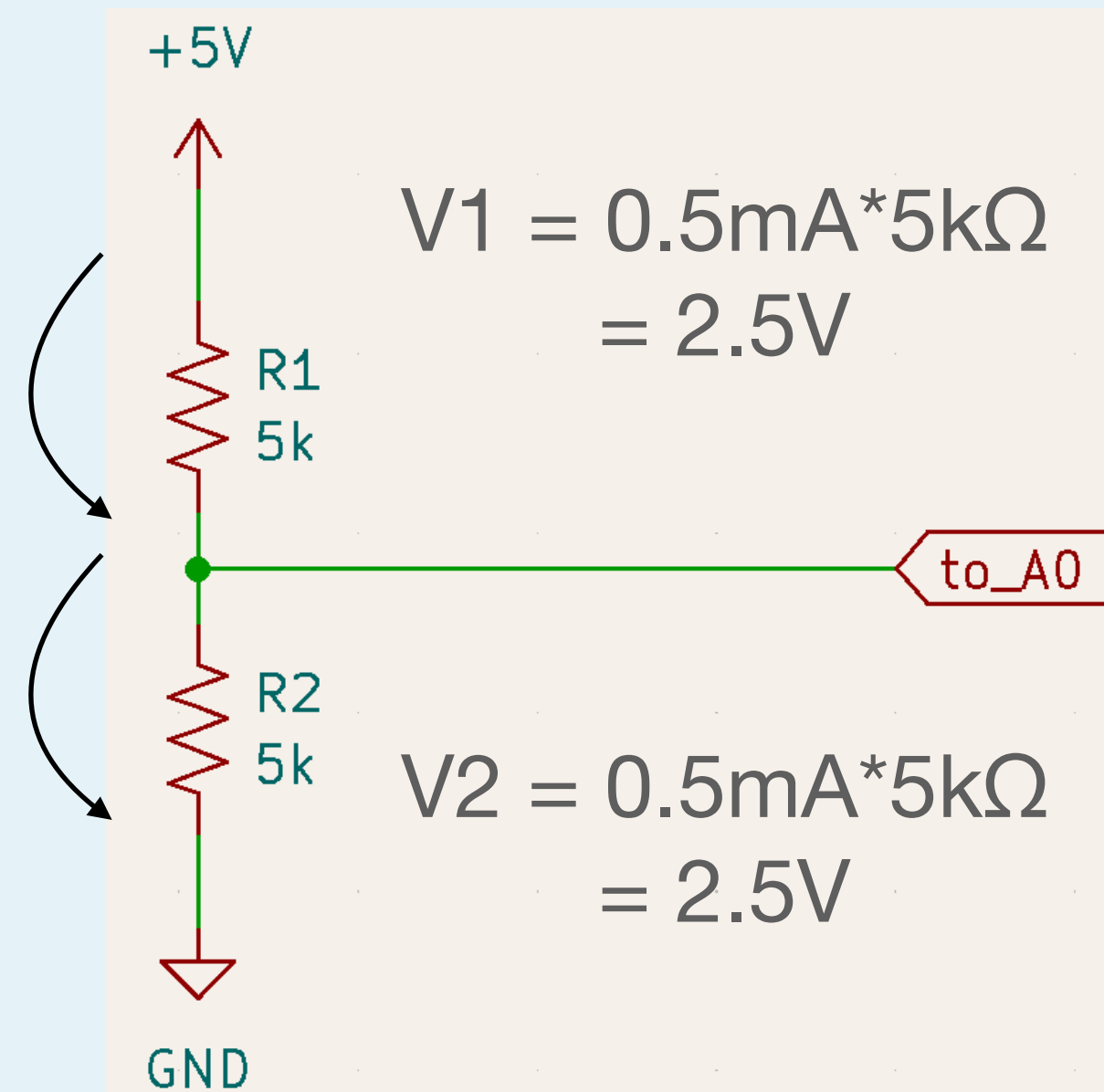
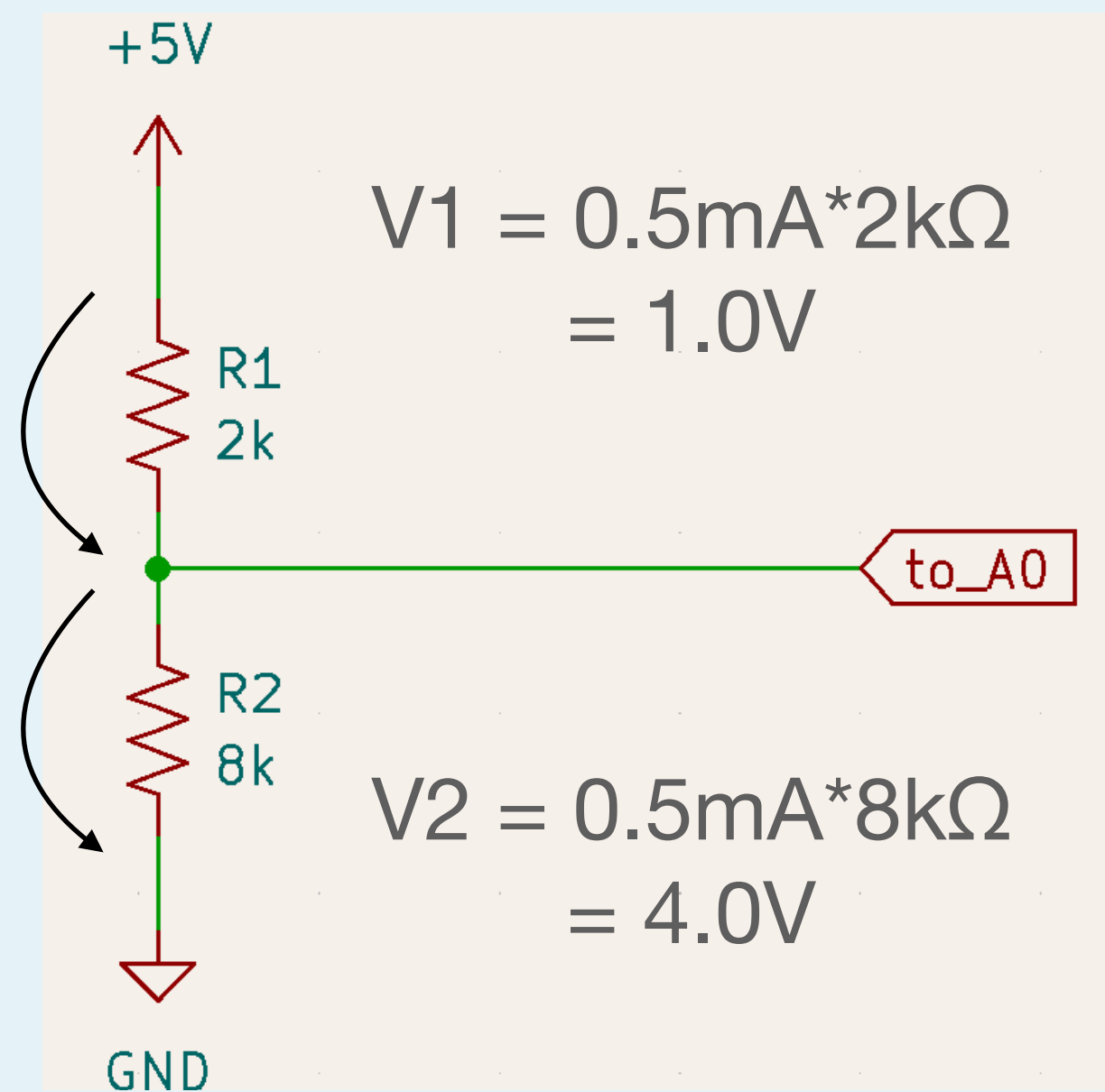


可変抵抗



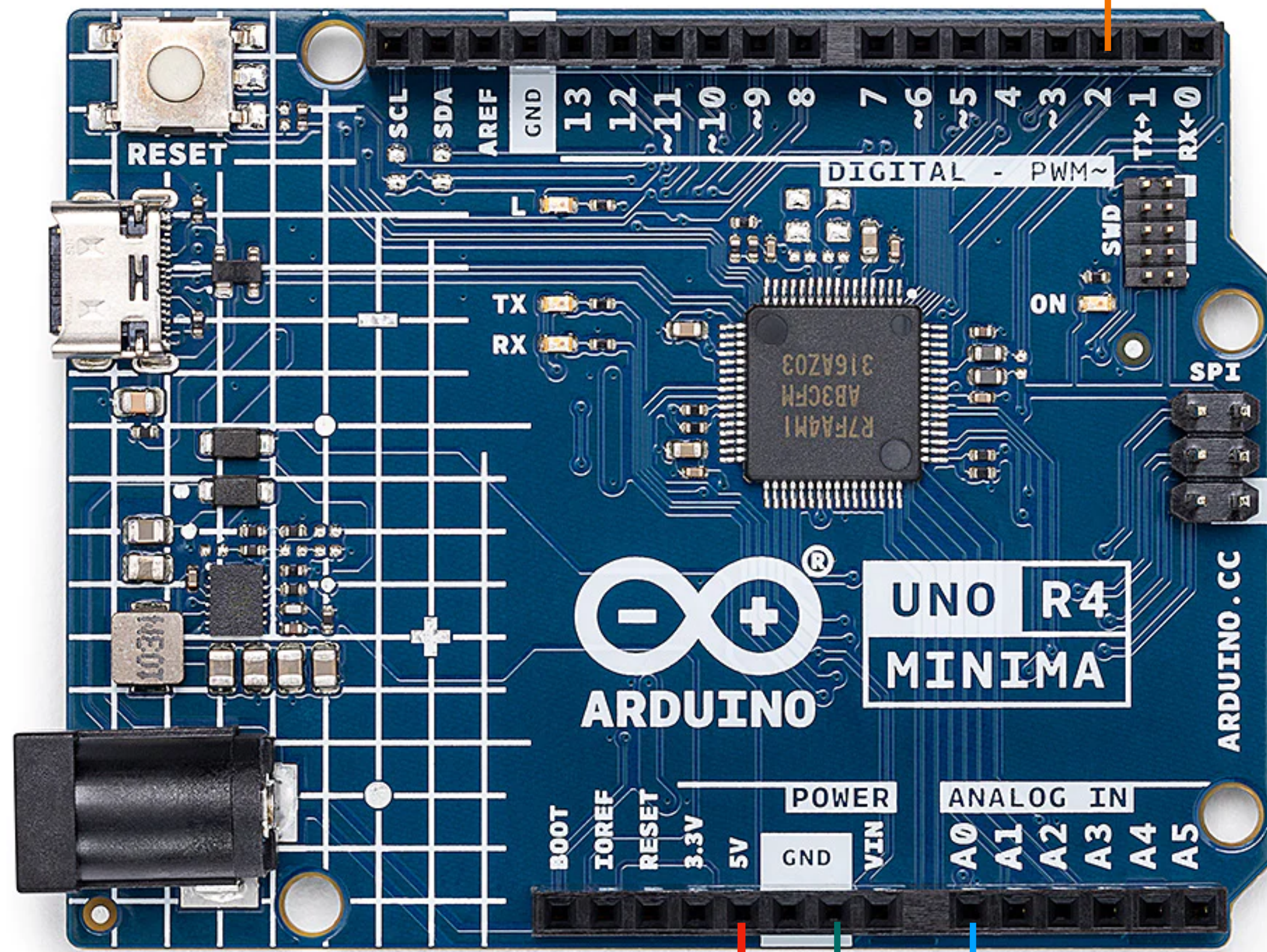
ワイパーがどの位置でも流れる電流はすべて0.5mA

可変抵抗



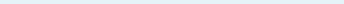
電流*抵抗値で、それぞれの抵抗での電圧降下が求まる

Pin2: INPUT_PULLUP



1k~100k







Digital 2

FirmataでProcessingから Arduinoを直接操作

Processingとの連携の方法

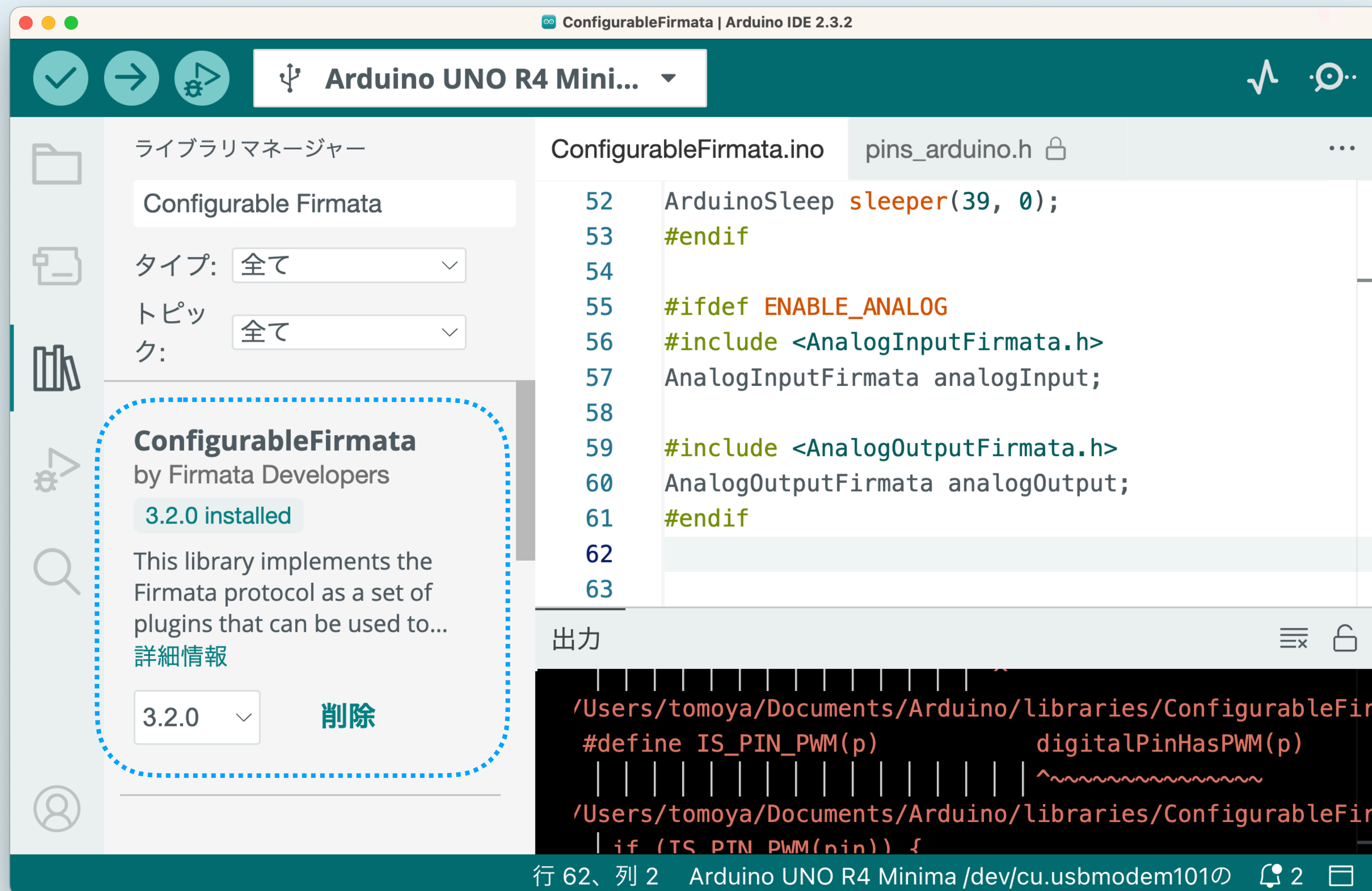


デメリット：前回使ったADCTouchのような、Arduino側のライブラリを使うのが難しい

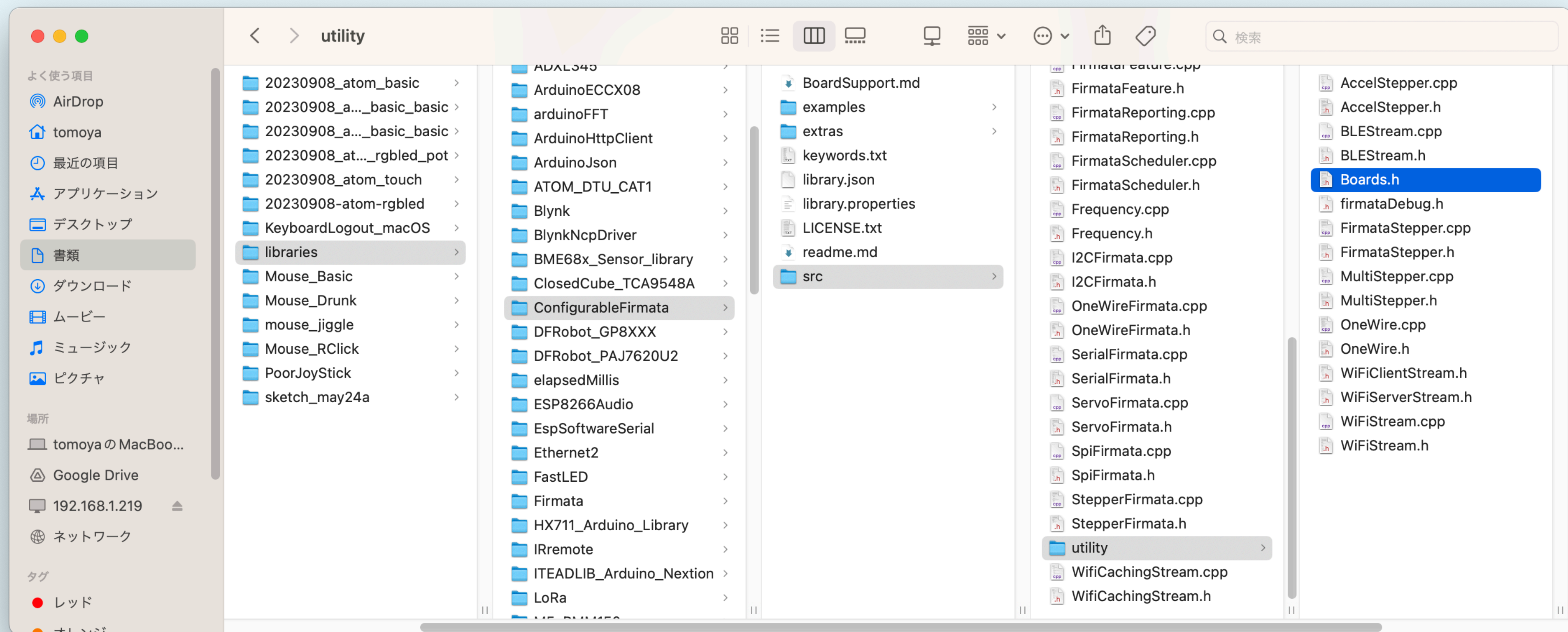
Processingとの連携の方法



同じFirmataのスケッチで、Cycling'74 Max (maxuino) や、Pure Data(pduino)でも同様に制御できる



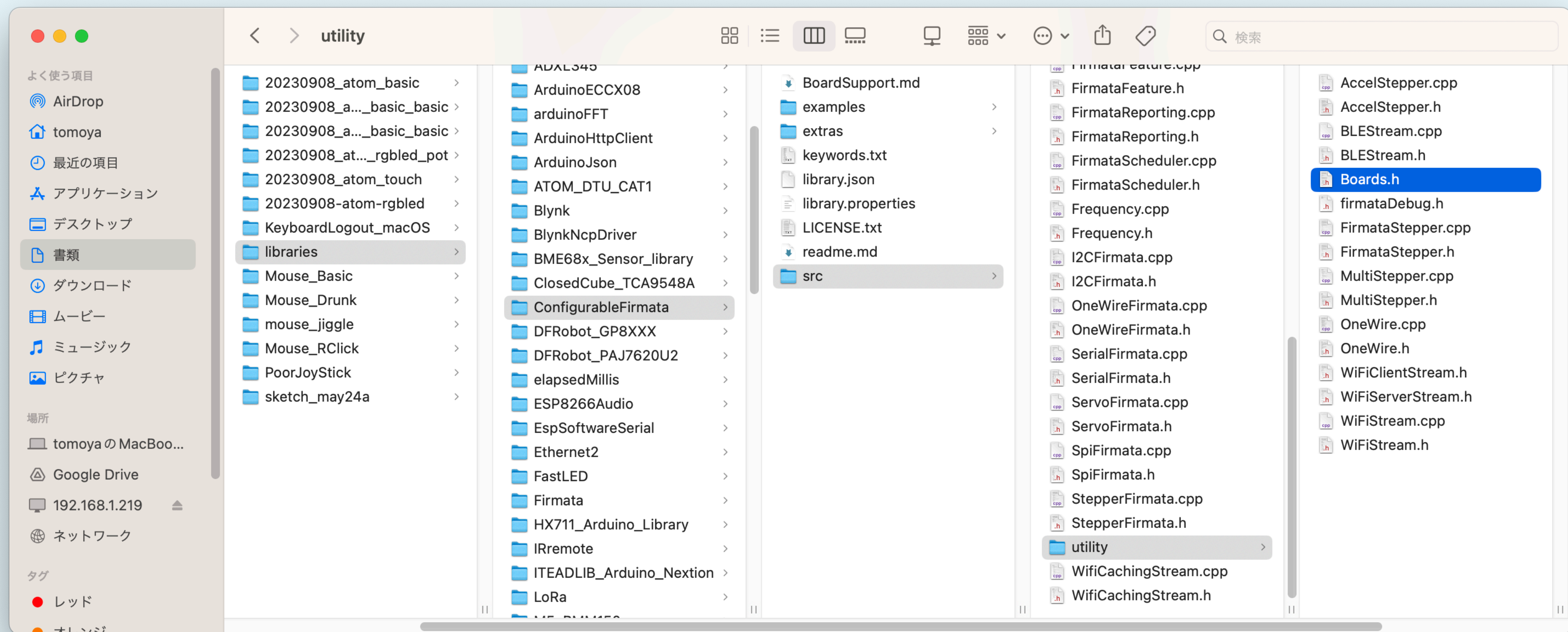
ライブラリマネージャーから "Configurable Firmata" をインストール



書類/Arduino/libralies/ConfigurableFirmata/src/utility/Boards.hを差し替えます

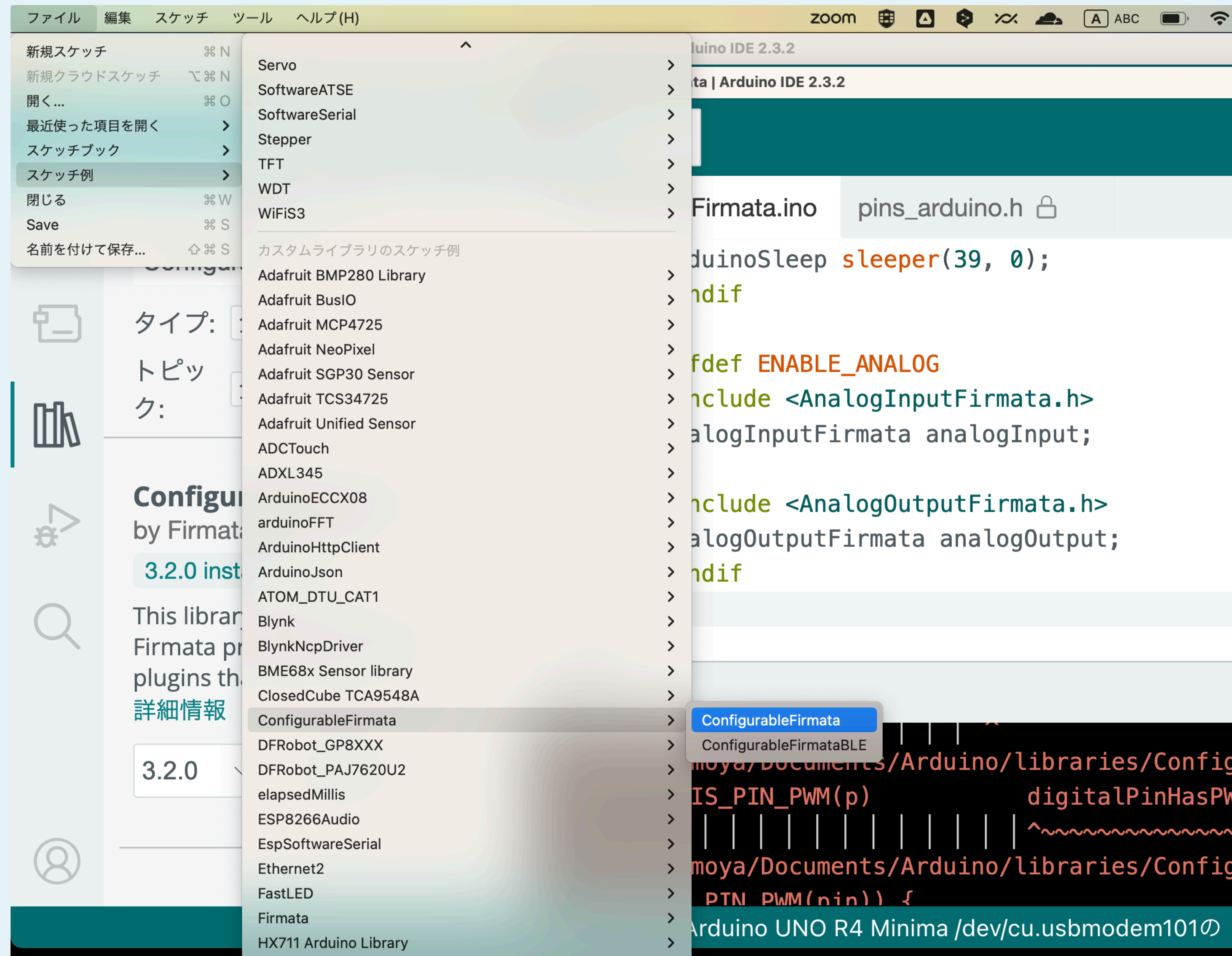
(Classroomでも配布)

[https://gist.githubusercontent.com/tomoyanonymou/
9efdd3e6063b9eeb0fb7527137a55c93/raw/
7c938abed36eb7b4b912f257b177e4e7ce96bed2/Boards.h](https://gist.githubusercontent.com/tomoyanonymou/9efdd3e6063b9eeb0fb7527137a55c93/raw/7c938abed36eb7b4b912f257b177e4e7ce96bed2/Boards.h)

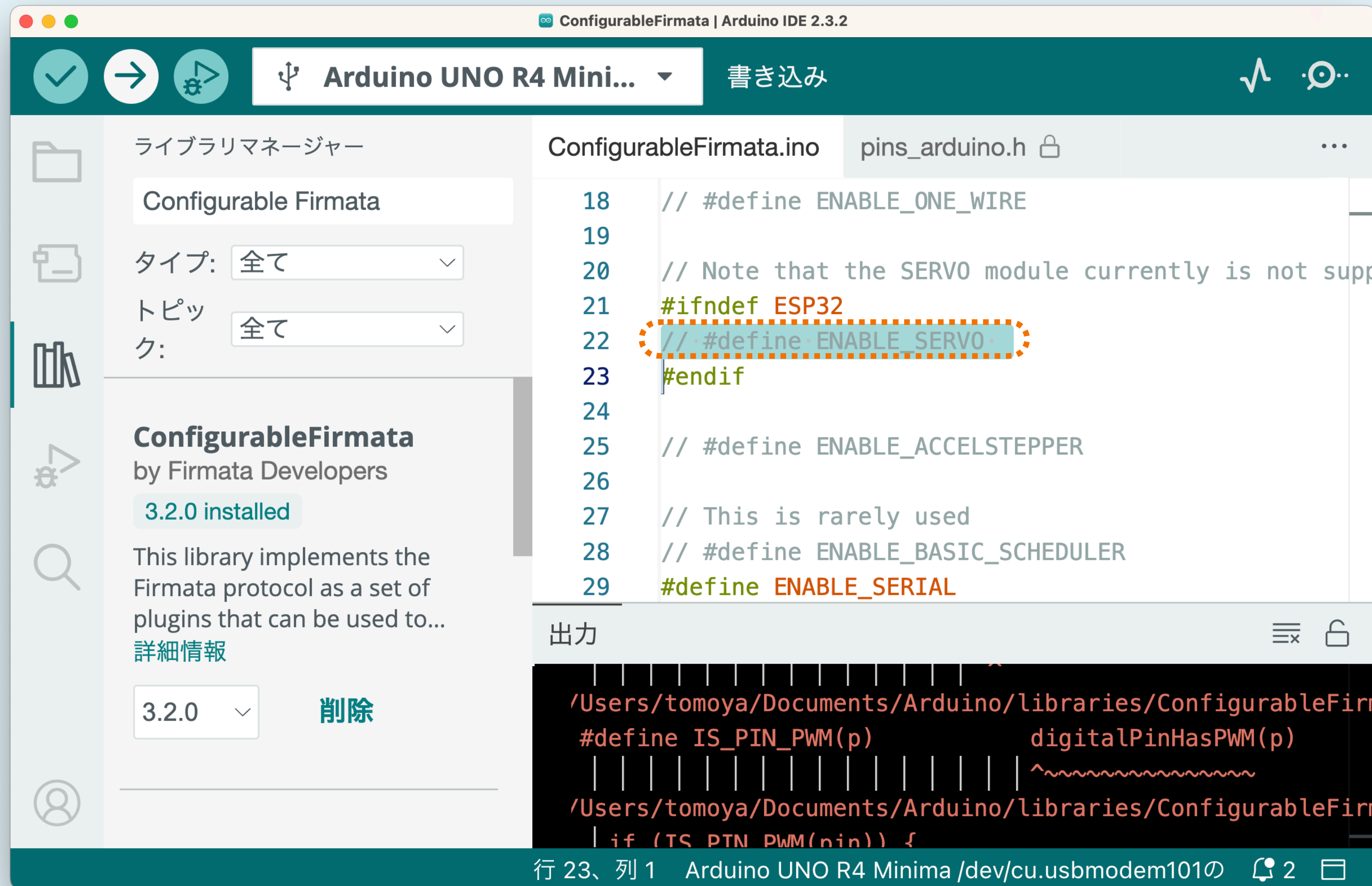


ボードの定義がバグっているため、
書類/Arduino/libraries/ConfigurableFirmata/src/utility/Boards.hを差し替えます

[https://gist.githubusercontent.com/tomoyanonymous/
9efdd3e6063b9eeb0fb7527137a55c93/raw/
7c938abed36eb7b4b912f257b177e4e7ce96bed2/Boards.h](https://gist.githubusercontent.com/tomoyanonymous/9efdd3e6063b9eeb0fb7527137a55c93/raw/7c938abed36eb7b4b912f257b177e4e7ce96bed2/Boards.h)



Exampleから、カスタムライブラリのスケッチ例→ConfigurableFirmata
→ConfigurableFirmataを開く

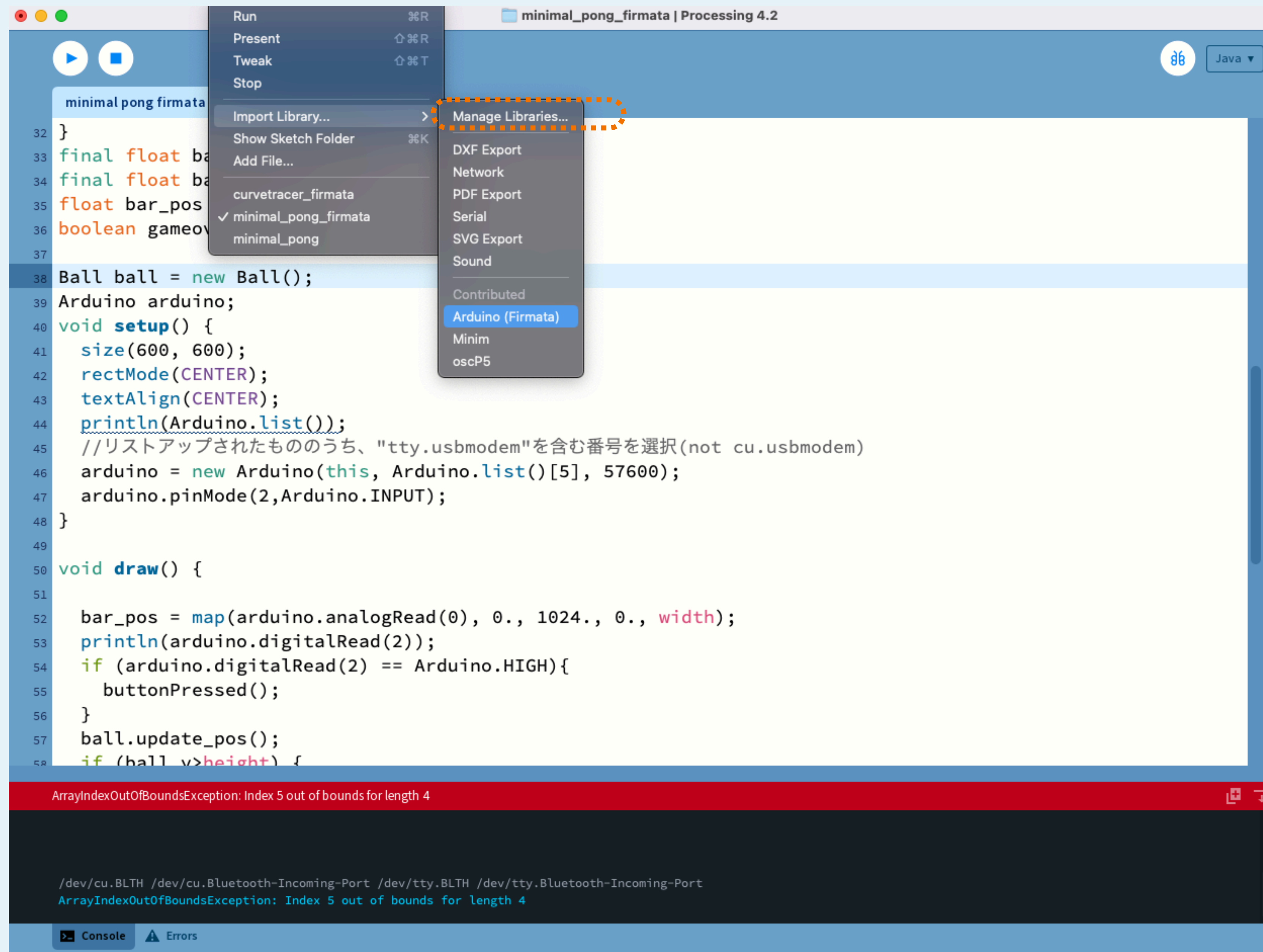


22行目の `#define ENABLE_SERVO` の行頭に `//` をつけてコメントアウト → 書き込み

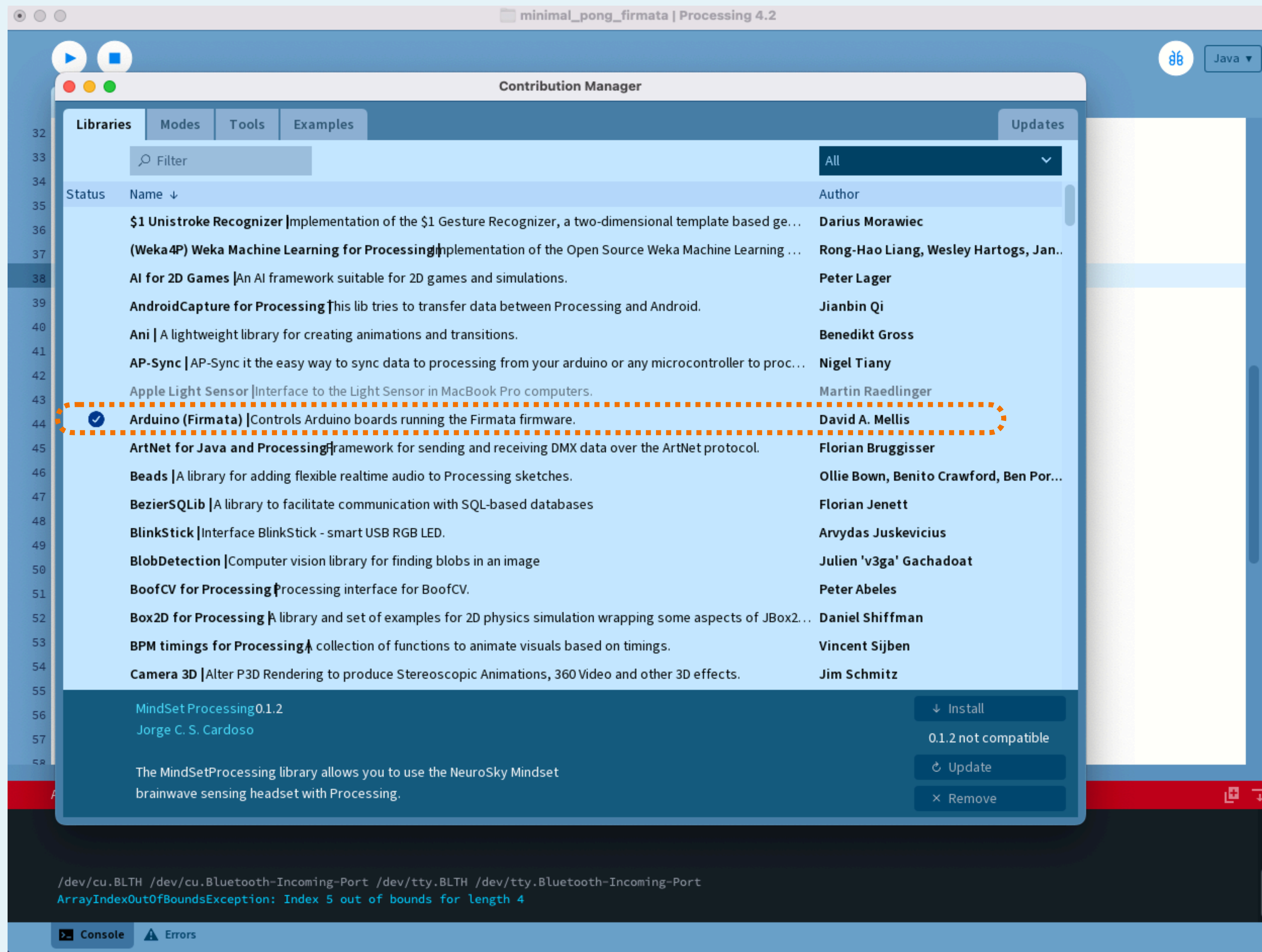
※Arduino Uno R3以前やAVRベースの互換ボードを使用するときは、"Firmata"ライブラリを入れて、
Exampleの"StandardFirmata"をそのまま書き込むだけでOK

昨年度の授業資料を参照

<https://teach.matsuuratomoya.com/docs/2023/code-design/8/>



“Manage Libraries”からArduinoライブラリを入れる



“Arduino(Firmata)”を探してインストール

やってみよう

- mouseXをanalogReadで置き換えよう
 - map関数で入力範囲を0~1024から、0~widthになるように
- マウスクリックをdigitalReadで置き換えよう
 - mousePressed()関数の代わりにbuttonClicked()関数を用意しよう

```

import processing.serial.*;
import cc.arduino.*;
import org.firmata.*;

class Ball {
  float x;
  float y;
  float vel_x;
  float vel_y;
  Ball() {
    this.x = width/2;
    this.y = height/2;
    this.vel_x = random(0, 1.0)> 0.5 ? -5:5;
    this.vel_y = random(0, 1.0)> 0.5 ? -5:5;
  }
  public void update_pos() {
    this.x += this.vel_x;
    this.y += this.vel_y;
    if (this.x < 0 || this.x > width) {
      this.reflect_horizontal();
    }
    if (this.y < 0 ) {
      this.reflect_vertical();
    }
  }
  void reflect_horizontal() {
    this.vel_x = -this.vel_x*1.1;
  }
  public void reflect_vertical() {
    this.vel_y = -this.vel_y*1.1;
  }
}

final float bar_width = 100;
final float bar_y = 550;
float bar_pos = 0.;
boolean gameover = false;

Ball ball = new Ball();
Arduino arduino;

```

```

void setup() {
  size(600, 600);
  rectMode(CENTER);
  textAlign(CENTER);
  println(Arduino.list());
  arduino = new Arduino(this, Arduino.list()[1], 57600);
  arduino.pinMode(2, Arduino.INPUT);
}

void draw() {
  bar_pos = map(arduino.analogRead(0), 0., 1024., 0.,
width);
  println(arduino.digitalRead(2));
  if (arduino.digitalRead(2) == Arduino.LOW){
    buttonPressed();
  }
  ball.update_pos();
  if (ball.y>height) {
    gameover = true;
  }
  if (gameover) {
    textSize(24);
    text("game over", width/2, height/2);
  } else {
    if (ball.y > bar_y &&
        ball.x > bar_pos - bar_width/2 &&
        ball.x < bar_pos + bar_width/2) {
      ball.reflect_vertical();
    }
    background(255);
    fill(0);
    rect(bar_pos, bar_y, bar_width, 30);
    ellipse(ball.x, ball.y, 10, 10);
  }
}

void buttonPressed() {
  if (gameover) {
    gameover = false;
    ball = new Ball();
  }
}

```


“Import Libraries”から
Arduinoを入れても、
serialは別途importする
必要あり

```
import processing.serial.*;
import cc.arduino.*;
import org.firmata.*;

class Ball {
  float x;
  float y;
  float vel_x;
  float vel_y;
  Ball() {
    this.x = width/2;
    this.y = height/2;
    this.vel_x = random(0, 1.0)> 0.5 ? -5:5;
    this.vel_y = random(0, 1.0)> 0.5 ? -5:5;
  }
  public void update_pos() {
    this.x += this.vel_x;
    this.y += this.vel_y;
    if (this.x < 0 || this.x > width) {
      this.reflect_horizontal();
    }
    if (this.y < 0 ) {
      this.reflect_vertical();
    }
  }
  void reflect_horizontal() {
    this.vel_x = -this.vel_x*1.1;
  }
  public void reflect_vertical() {
    this.vel_y = -this.vel_y*1.1;
  }
}

final float bar_width = 100;
final float bar_y = 550;
float bar_pos = 0.;
boolean gameover = false;

Ball ball = new Ball();
Arduino arduino;
```

```
void setup() {
  size(600, 600);
  rectMode(CENTER);
  textAlign(CENTER);
  println(Arduino.list());
  arduino = new Arduino(this, Arduino.list()[5], 57600);
  arduino.pinMode(2, Arduino.INPUT_PULLUP);
}

void draw() {
  bar_pos = map(arduino.analogRead(0), 0., 1024., 0.,
width);
  println(arduino.digitalRead(2));
  if (arduino.digitalRead(2) == Arduino.HIGH){
    buttonPressed();
  }
  ball.update_pos();
  if (ball.y>height) {
    gameover = true;
  }
  if (gameover) {
    textSize(24);
    text("game over", width/2, height/2);
  } else {
    if (ball.y > bar_y &&
        ball.x > bar_pos - bar_width/2 &&
        ball.x < bar_pos + bar_width/2) {
      ball.reflect_vertical();
    }
    background(255);
    fill(0);
    rect(bar_pos, bar_y, bar_width, 30);
    ellipse(ball.x, ball.y, 10, 10);
  }
}

void buttonPressed() {
  if (gameover) {
    gameover = false;
    ball = new Ball();
  }
}
```

コンソールに現れた
USBデバイス一覧か
ら、“tty.usbmodem”を含む物
を0スタートのインデックス
で指定
(cu.usbmodemだとうまくい
かないかも)

`pinMode(2,INPUT_PULLUP)` \longrightarrow `arduino.pinMode(2,Arduino.INPUT_PULLUP)`

`analogRead(A0)` \longrightarrow `arduino.analogRead(0)`

`digitalRead(2)` \longrightarrow `arduino.digitalRead(2)`

`HIGH,LOW` \longrightarrow `Arduino.HIGH,Arduino.LOW`

Arduino.でアクセスする（クラス）か、arduino.でアクセスする(インスタンス)かは、
仮にArduinoを2つ同時に使いたいときに意味が変わるかどうか？
で考えよう。analogReadなどの値はボードごとに異なるが、HIGHやLOWの定義は
インスタンス間で共通（静的メンバ変数）

やってみよう

- 同じ1ノブ1ボタンのインターフェースで、どんな操作の違いを考えられるか。
 - 例えば、ノブを自動で往復するバーの速度を変える役割にするとか
 - ボタンをゲームリセットの代わりに、押すたびにボールの数が増えるボタンにするとか
- 可変抵抗の代わりに、最初に使った赤外線距離センサーを使うとどうなるか