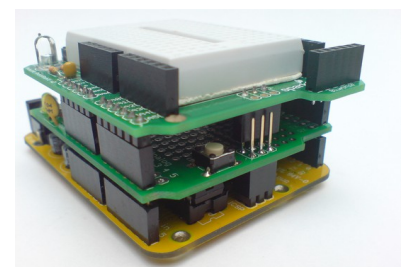
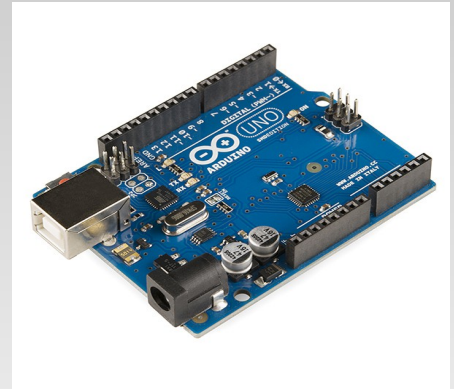


# Arduino

- *Single-board microcontroller*
- Microcontrolador
  - CPU, Memória, Serial, I/O
- Placa
  - Conectores, Fonte, USB, LEDs
- IDE
  - Compilador, Bibliotecas, Editor, *Burner*
  - <http://arduino.cc/en/Reference/HomePage>
- Shields
  - Display, Ethernet, Sensores, etc.

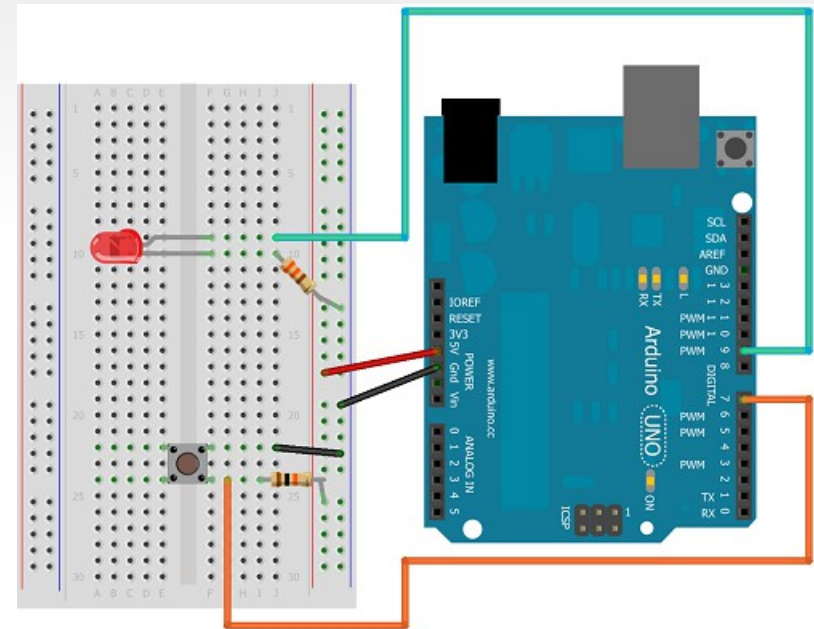


# I/O básico

```
// configura pino para I/O
pinMode(7, INPUT);
pinMode(9, OUTPUT);

// lê o pino
int val = digitalRead(7);

// escreve no pino
digitalWrite(9, HIGH);
```



# Hello World: output

- Piscar o LED a cada 1 segundo
- `reativos/code/arduino/00_blink.ino`

```
#define LED_PIN 13

void setup () {
    pinMode(LED_PIN, OUTPUT);    // Enable pin 13 for digital output
}

void loop () {
    digitalWrite(LED_PIN, HIGH); // Turn on the LED
    delay(1000);                 // Wait one second (1000 milliseconds)
    digitalWrite(LED_PIN, LOW);  // Turn off the LED
    delay(1000);                 // Wait one second
}
```

# Hello World: input

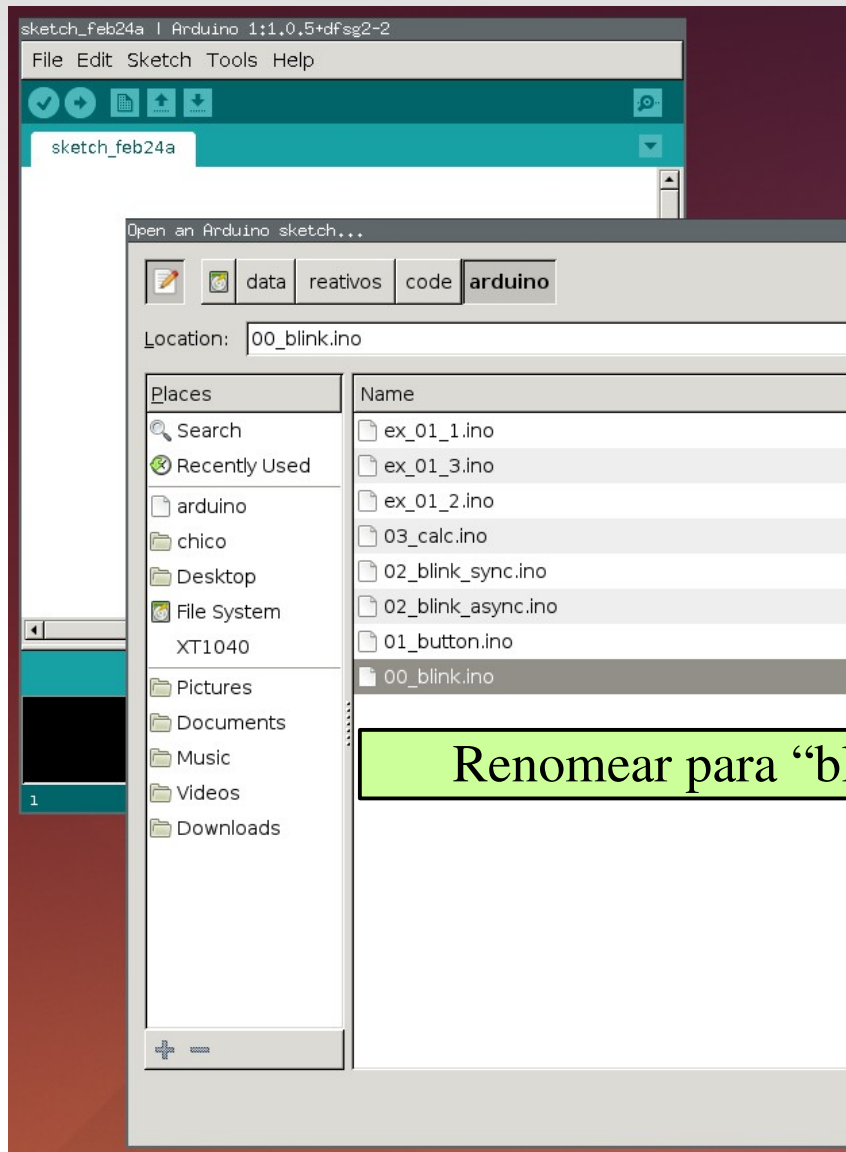
- Fazer o LED acompanhar o estado do botão
- `reativos/code/arduino/01_button.ino`

```
#define LED_PIN 13
#define BUT_PIN 2

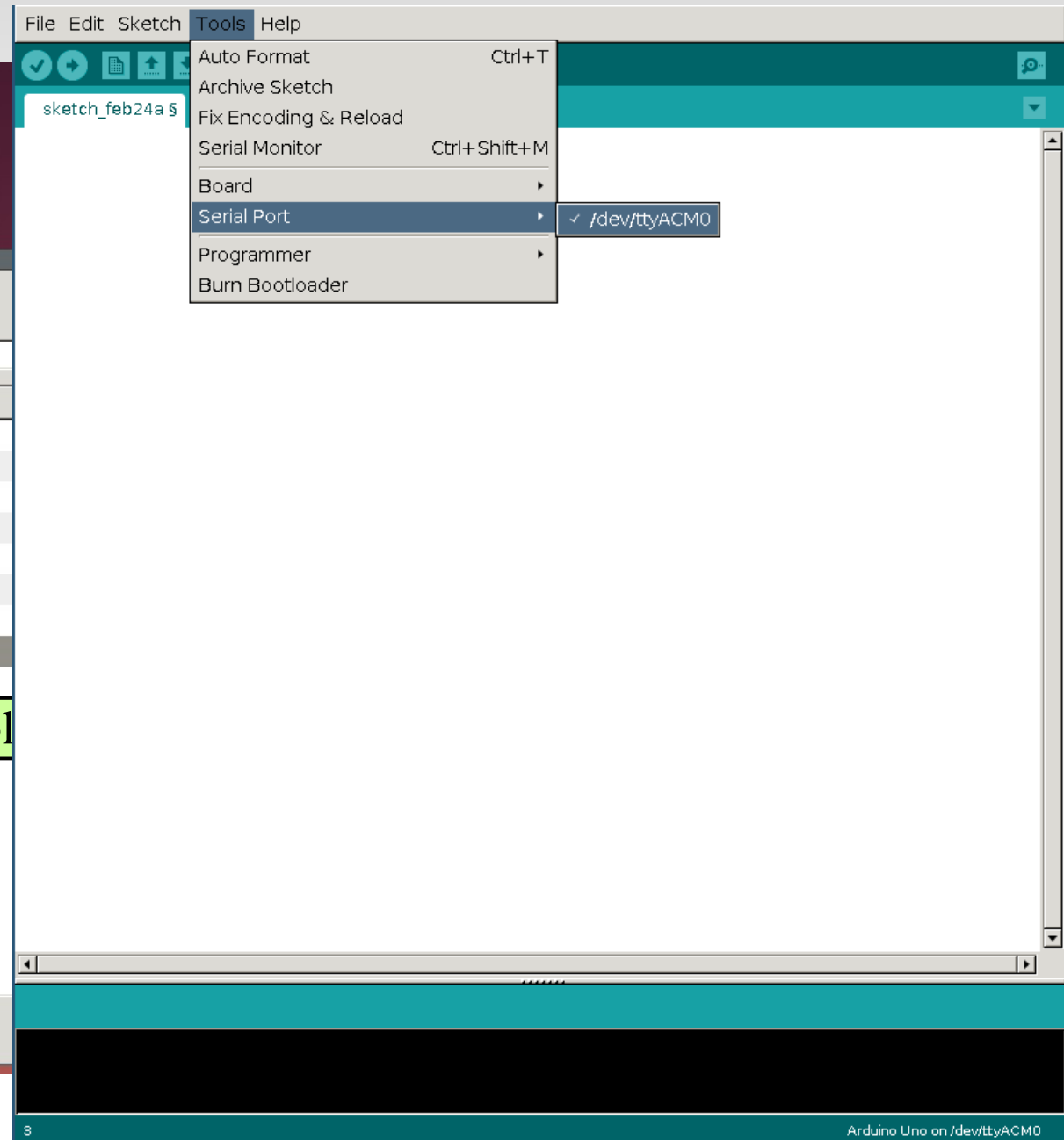
void setup () {
    pinMode(LED_PIN, OUTPUT);           // Enable pin 13 for digital output
    pinMode(BUT_PIN, INPUT);            // Enable pin 2 for digital input
}

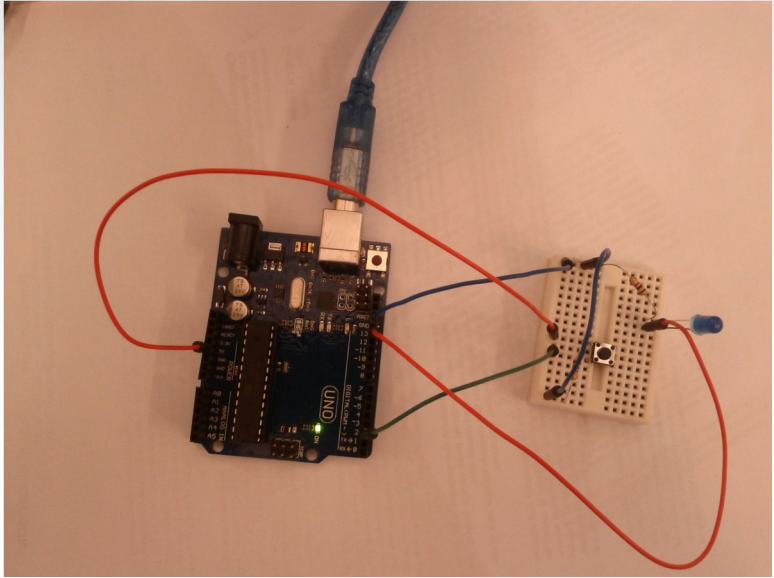
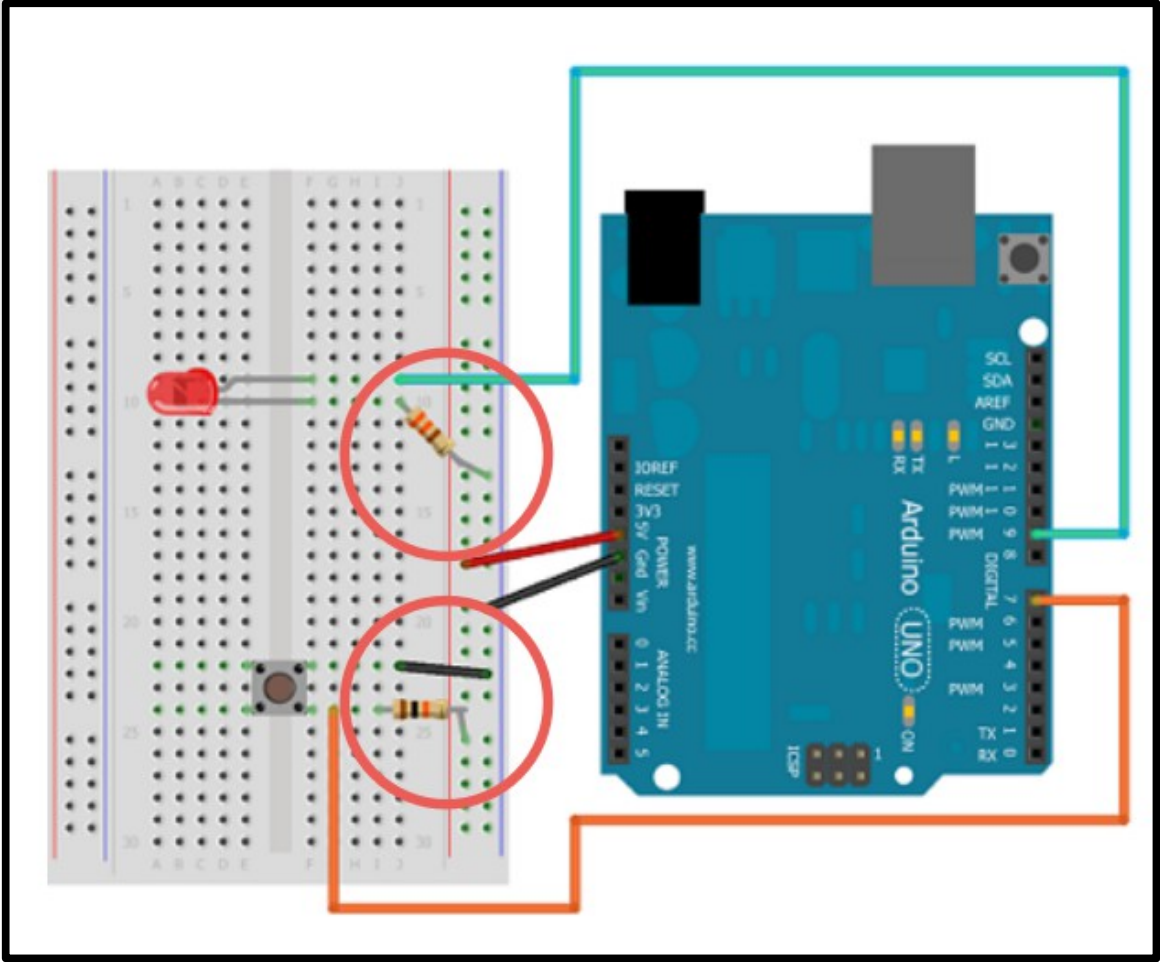
void loop () {
    int but = digitalRead(BUT_PIN);    // Read button state
    digitalWrite(LED_PIN, but);        // Copy state to LED
}
```

# Arduino IDE



Renomear para "bl



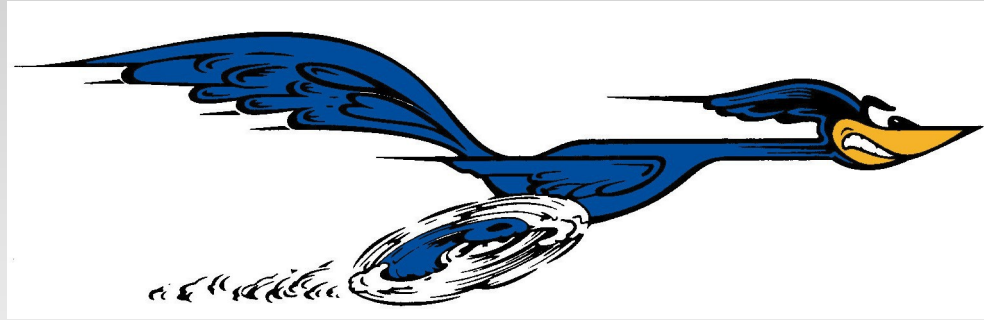


# Exercício 1

- Piscar o LED a cada 1 segundo
- Parar ao pressionar o botão, mantendo o LED aceso para sempre

```
void loop () {  
    digitalWrite(LED_PIN, HIGH);  
    delay(1000);  
    digitalWrite(LED_PIN, LOW);  
    delay(1000);  
  
    int but = digitalRead(BUT_PIN);  
    if (but) {  
        digitalWrite(LED_PIN, HIGH);  
        while(1);  
    }  
}
```

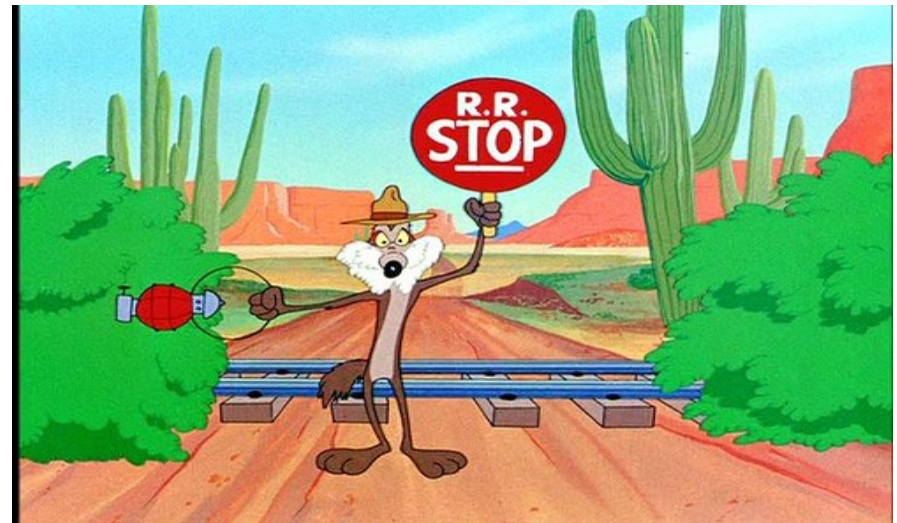
- Programa interativo!



# Programa Reativo

VS

# Chamadas Bloqueantes





# Exercício 1 - Alternativa

- Usar a função `millis()` para contar o tempo, **sem bloquear**.

## `millis()`

### Description

Returns the number of milliseconds since the Arduino board began running the current program. This number will overflow (go back to zero), after approximately 50 days.

### Parameters

None

### Returns

Number of milliseconds since the program started (*unsigned long*)

```
void loop () {  
    millis(); // 1,2,4,5,...  
    delay(1);  
}
```


# Exercício 1 - Reativo

- Guardar *timestamp* da última mudança
- Guardar estado atual do LED

# Inversão de Controle

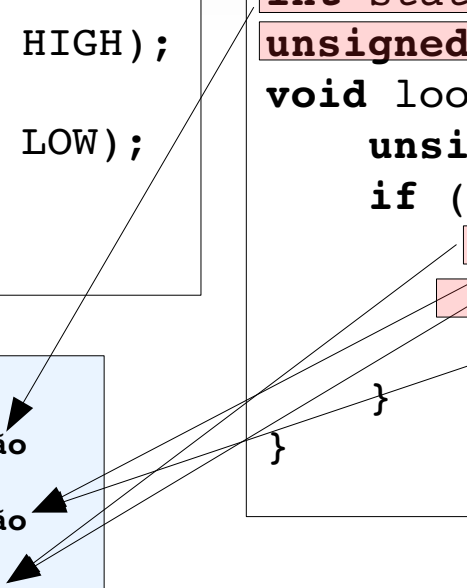
- Aplicação (programador) => Ambiente (dispositivos)
- Programação sequencial => Variáveis globais de estado

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



- inicialização  
- decodificação  
- codificação

```
int state = 1;  
unsigned long old;  
void loop () {  
    unsigned long now = millis();  
    if (now >= old+1000) {  
        old = now;  
        state = !state;  
        digitalWrite(LED_PIN, state);  
    }  
}
```



# Tradeoff

- Execução sequencial com chamadas bloqueantes
  - não reativo
- Inversão de controle e variáveis de estado
  - reativo

# Tarefa-02

(a conferir na próxima aula)

- Piscar o LED a cada 1 segundo
- Botão 1: Acelerar o pisca-pisca a cada pressionamento  
*(somente na transição de LOW->HIGH)*
- Botão 2: Desacelerar a cada pressionamento  
*(somente na transição de LOW->HIGH)*
- Botão 1+2 (em menos de 500ms): Parar



# Modelos de Concorrência

- Modelo Assíncrono
  - ChibiOS: <http://www.chibios.org>
  - Occam-PI:
- Modelo Síncrono
  - Arduino Loop
  - Céu

# Modelo Assíncrono

- Por quê?
  - Como descrever e entender as partes de um sistema concorrente.
  - Vocabulário e semântica
    - execução, composição, comunicação, sincronização
- Modelo Assíncrono
  - Execução independente / Sincronização explícita
    - Threads + locks/mutexes (p-threads, Java Threads)
    - Atores + message passing (erlang, go)
- Modelo Síncrono
  - Execução dependente / Sincronização implícita



# Modelo Assíncrono

- Por quê?
  - Como descrever e entender as partes de um sistema concorrente.
  - Vocabulário e semântica
    - execução, composição, comunicação, sincronização
- Modelo Assíncrono
  - Execução independente / Sincronização explícita
    - Threads + locks/mutexes (p-threads, Java Threads)
    - Atores + message passing (erlang, go)
- Modelo Síncrono
  - Execução dependente / Sincronização implícita
    - Arduino, Circuitos, Game Loops, Padrão Observer



# Mini Arduino

(29 de março e 19 de abril)

- Projeto com sensores, atuadores, e cálculo pesado
  - propostas até 29/03
    - `mini-arduino/PROJETO.md`
- Implementação em C ou Céu
- Extras
  - interrupções
  - threads
- Apresentações em 19/04

# Mini Arduino

(29 de março e 19 de abril)

- Entrada / Sensor
  - Distância, Movimento, Controle IR, RTC, Acelerômetro, Teclado, Umidade, Temperatura, Luz, Botões
- Saída / Atuador
  - LEDs, LCD, Motor, Servo, Buzina
- Entrada e Saída
  - Módulo RF, Bluetooth, Tela touch, Kit carro
- Links
  - <http://www.eletrogate.com/pd-218d8b-kit-big-jack.html>
  - <http://www.eletrogate.com/pd-1c2f01-kit-chassi-2wd-robo-para-arduino.html>
  - <http://excamera.com/sphinx/gameduino2/>