S4A (Scratch for Arduino) est un programme qui permet d'utiliser une copie de Scratch pour programmer un Arduino.

right|thumb| Icône du logiciel S4A

Merci les développeur de S4A !

Vous trouverez tout sur le site de [http://s4a.cat/ S4A]

Allez sur '“download”' et chargez la version du programme qui convient à votre ordinateur.

Après l'installation, il faut préparer votre carte Arduino !

Préparez votre carte Arduino

Le [https://fr.wikipedia.org/wiki/Firmware firmware] est un logiciel chargé sur un matériel informatique pour qu'il puisse fonctionner.

Dans la cas de S4A, il faut doter l'Arduino de ce programme pour qu'il interagisse avec le logiciel S4A.

Copiez le programme firmware en bas de cet article et collez le comme nouveau programme dans le logiciel Arduino(IDE). Téléversez-le sur votre Arduino.

200px

Votre Arduino est prêt à fonctionner avec S4A !

Comment fonctionne S4A ?

Comme scratch ! Avec cependant quelques différences !

Par exemple, les programme de scratch 2 ne fonctionnent pas avec S4A. Vous devrez donc créer vos programme pour S4A. _ On y trouve aussi des fonctions différentes, celles qui permettent d'interagir avec l'Arduino.

En voici l'allure générale :

S4Ageneral.jpeg|600px

Quand lance S4A, il attend une carte Arduino équipée du firmware :

S4Aattendlacarte.jpeg|300px

Ici lorsqu'on écrit un programme ou qu'on le charge.

S4Aavecprogramme.jpeg|600px

On crée un programme par glisser-déposer :

S4Adelacerblocs.jpeg|600px

On y retrouve le sélecteur de grand types de fonctions : mouvement, son, … :

Selecteurdetypesdefonctions.jpeg

Avec des fonctions liées à Arduino :

180px 180px180px180px

180px180px180px180px

Le nom du programme, n'est pas très visible :

Nomduprog.jpeg|300px

Le firmware

<pre> *NEW IN VERSION 1.6c (by Jorge Gomez):* Fixed variable type in pin structure: pin.state should be int, not byte *Optimized speed of execution while receiving data from computer in readSerialPort()* NEW IN VERSION 1.6b (by Jorge Gomez): *Added new structure arduinoPins to hold the pins information:* - This makes the code easier to read and modify (IMHO) - *Allows to change the type of pin more easily to meet non standard use of S4A* - Eliminates the need of having to deal with different kind of index access (ie: states[pin-4]) - *By using an enum to hold all the possible output pin states the code is now more readable* Changed all functions using old style pin access: configurePins(), resetPins(), readSerialPort(), updateActuator() and sendUpdateActuator() *Fixed possible overflow every 70 minutes (2e32 us) in pulse() while using micros(). Changed for delayMicroseconds()* Some minor coding style fixes

NEW IN VERSION 1.6a (by Jorge Gomez): Fixed compatibility with Arduino Leonardo by avoiding the use of timers *readSerialPort() optimized:* - created state machine for reading the two bytes of the S4A message - *updateActuator() is only called if the state is changed* Memory use optimization *Cleaning some parts of code* Avoid using some global variables

NEW IN VERSION 1.6: Refactored reset pins *Merged code for standard and CR servos* Merged patch for Leonardo from Peter Mueller (many thanks for this!)

NEW IN VERSION 1.5: Changed pin 8 from standard servo to normal digital output

NEW IN VERSION 1.4: Changed Serial.print() for Serial.write() in ScratchBoardSensorReport function to make it compatible with latest Arduino IDE (1.0)

NEW IN VERSION 1.3: Now it works on GNU/Linux. Also tested with MacOS and Windows 7. *timer2 set to 20ms, fixing a glitch that made this period unstable in previous versions.* readSerialport() function optimized. *pulse() modified so that it receives pulse width as a parameter instead using a global variable.* updateServoMotors changes its name as a global variable had the same name. *Some minor fixes. typedef enum { input, servomotor, pwm, digital } pinType; typedef struct pin { pinType type;* Type of pin

```
int state;         //State of an output
//byte value;       //Value of an input. Not used by now. TODO
```

};

pin arduinoPins[14]; *Array of struct holding 0-13 pins information unsigned long lastDataReceivedTime = millis(); void setup() { Serial.begin(38400); Serial.flush(); configurePins(); resetPins(); } void loop() { static unsigned long timerCheckUpdate = millis(); if (millis()-timerCheckUpdate>=20) { sendUpdateServomotors(); sendSensorValues(); timerCheckUpdate=millis(); } readSerialPort(); } void configurePins() { arduinoPins[0].type=input; arduinoPins[1].type=input; arduinoPins[2].type=input; arduinoPins[3].type=input; arduinoPins[4].type=servomotor; arduinoPins[5].type=pwm; arduinoPins[6].type=pwm; arduinoPins[7].type=servomotor; arduinoPins[8].type=servomotor; arduinoPins[9].type=pwm; arduinoPins[10].type=digital; arduinoPins[11].type=digital; arduinoPins[12].type=digital; arduinoPins[13].type=digital; } void resetPins() { for (byte index=0; index <=13; index++) { if (arduinoPins[index].type!=input) { pinMode(index, OUTPUT); if (arduinoPins[index].type==servomotor) { arduinoPins[index].state = 255; servo (index, 255); } else { arduinoPins[index].state=0; digitalWrite(index,LOW); } } } } void sendSensorValues() { unsigned int sensorValues[6], readings[5]; byte sensorIndex; for (sensorIndex = 0; sensorIndex < 6; sensorIndex++)* for analog sensors, calculate the median of 5 sensor readings in order to avoid variability and power surges

```
{
  for (byte p = 0; p < 5; p++)
    readings[p] = analogRead(sensorIndex);
  insertionSort(readings, 5); //sort readings
  sensorValues[sensorIndex] = readings[2]; //select median reading
}
```

```
//send analog sensor values
for (sensorIndex = 0; sensorIndex < 6; sensorIndex++)
  ScratchBoardSensorReport(sensorIndex, sensorValues[sensorIndex]);
```

```
//send digital sensor values
ScratchBoardSensorReport(6, digitalRead(2)?1023:0);
ScratchBoardSensorReport(7, digitalRead(3)?1023:0);
```

}

void insertionSort(unsigned int* array, unsigned int n) {

```
for (int i = 1; i < n; i++)
  for (int j = i; (j > 0) && ( array[j] < array[j-1] ); j--)
    swap( array, j, j-1 );
```

}

void swap(unsigned int* array, unsigned int a, unsigned int b) {

```
unsigned int temp = array[a];
array[a] = array[b];
array[b] = temp;
```

}

void ScratchBoardSensorReport(byte sensor, int value) *PicoBoard protocol, 2 bytes per sensor { Serial.write(B10000000 |* [1]

[1]

sensor & B1111)«3)

```
  | ((value>>7) & B111));
Serial.write( value & B1111111);
```

} void readSerialPort() {

```
byte pin;
int newVal;
static byte actuatorHighByte, actuatorLowByte;
static byte readingSM = 0;
```

```
if (Serial.available())
{
  if (readingSM == 0)
  {
    actuatorHighByte = Serial.read();
    if (actuatorHighByte >= 128) readingSM = 1;
  }
  else if (readingSM == 1)
  {
    actuatorLowByte = Serial.read();
    if (actuatorLowByte < 128) readingSM = 2;
    else readingSM = 0;
  }
```

```
  if (readingSM == 2)
  {
    lastDataReceivedTime = millis();
    pin = ((actuatorHighByte >> 3) & 0x0F);
    newVal = ((actuatorHighByte & 0x07) << 7) | (actuatorLowByte & 0x7F);
```

```
    if(arduinoPins[pin].state != newVal)
    {
      arduinoPins[pin].state = newVal;
      updateActuator(pin);
    }
    readingSM = 0;
  }
}
else checkScratchDisconnection();
```

} void reset() *with xbee module, we need to simulate the setup execution that occurs when a usb connection is opened or closed without this module { resetPins();* reset pins

```
sendSensorValues(); // protocol handshaking
```

```
lastDataReceivedTime = millis();
```

} void updateActuator(byte pinNumber) {

```
if (arduinoPins[pinNumber].type==digital) digitalWrite(pinNumber,
arduinoPins[pinNumber].state);
else if (arduinoPins[pinNumber].type==pwm) analogWrite(pinNumber,
arduinoPins[pinNumber].state);
```

} void sendUpdateServomotors() {

```
for (byte p = 0; p < 10; p++)
  if (arduinoPins[p].type == servomotor) servo(p, arduinoPins[p].state);
```

} void servo (byte pinNumber, byte angle) {

```
if (angle != 255)
  pulse(pinNumber, (angle * 10) + 600);
```

} void pulse (byte pinNumber, unsigned int pulseWidth) {

```
digitalWrite(pinNumber, HIGH);
delayMicroseconds(pulseWidth);
digitalWrite(pinNumber, LOW);
```

} void checkScratchDisconnection() *the reset is necessary when using an wireless arduino board (because we need to ensure that arduino isn't waiting the actuators state from Scratch) or when scratch isn't sending information (because is how serial port close is detected) { if (millis() - lastDataReceivedTime > 1000) reset();* reset state if actuators reception timeout = one second } S4AFirmware16.inoS4AFirmware16.ino </pre>