

Document technique

Arduino et capteurs

Sommaire

Introduction	3
Structure d'un code Arduino générique	3
Inclusion des Bibliothèques	5
Déclaration des Constantes et des Variables Globales	5
Section de Configuration - setup()	5
Boucle Principale - loop()	6
Meilleures Pratiques	6
Les différentes broches de l'Arduino (pins)	7
Broches d'alimentation	7
Broches de lecture des capteurs	8
Broches Numérique	8
Broches Analogiques	8
Broches de Communication (à titre informatif)	8
Communication série	8
Communication sans fil	9
Les Bonnes Pratiques	9
Communication Arduino à Godot (et vis-versa)	11
Tutoriel : Mise en place de l'environnement pour la communication série	12
1. Vérification de l'environnement de développement	12
1.1 Vérification de la version de .NET	12
2. Installation et configuration de System.IO.Ports	12
2.1 Vérification des packages installés	12
2.2 Installation de System.IO.Ports	12
3. Mise à jour du projet vers .NET 8 (si nécessaire)	13
3.1 Modification du fichier .csproj	13
3.2 Régénération des fichiers de compilation	13

—

Introduction

Bienvenue à tous dans notre hackathon, ce document est dédié à l'exploration des capteurs avec l'Arduino UNO R3 et l'envoi des données de ceux-ci vers Godot.

Avant de plonger dans l'univers de l'électronique, il est essentiel de comprendre quelques principes de base pour éviter d'endommager les composants ou la carte Arduino elle-même.

Structure d'un code Arduino générique

Voici un code basique qui lit un capteur sur une broche et qui allume et éteint une led en fonction des valeurs du capteur.

```
// Inclusion des bibliothèques nécessaires
#include <LibraryName.h> // Inclure ici les bibliothèques nécessaires à votre projet

// Déclaration des constantes et des variables globales
const int ledPin = 13; // Exemple de constante pour le numéro de broche
int sensorValue = 0; // Exemple de variable pour stocker la valeur lue par un capteur

// Section de configuration - setup()
// Cette fonction est appelée une fois lorsque l'Arduino démarre
void setup() {
    // Initialisation des broches
    pinMode(ledPin, OUTPUT); // Définir la broche comme une sortie
    pinMode(A0, INPUT); // Définir la broche comme une entrée

    // Initialisation de la communication série (si nécessaire)
    Serial.begin(9600); // Initialiser la communication série à 9600 bauds

    // Initialisation des composants et des bibliothèques
    // Par exemple, initialiser un écran LCD, un capteur, etc.
    // lcd.begin(16, 2);
}
```

```
// Autres configurations initiales
// Par exemple, initialiser les valeurs de variables ou l'état des broches
digitalWrite(ledPin, LOW); // Initialiser la LED à l'état bas
}

// Boucle principale - loop()
// Cette fonction est appelée en boucle après setup()
void loop() {
    // Lire les données des capteurs
    sensorValue = analogRead(A0); // Lire une valeur analogique depuis la broche A0

    // Traitement des données
    // Par exemple, convertir la valeur lue en une autre unité, filtrer les données, etc.
    int processedValue = sensorValue / 4;

    // Contrôle des sorties
    // Par exemple, allumer une LED en fonction de la valeur lue
    if (processedValue > 100) {
        digitalWrite(ledPin, HIGH); // Allumer la LED
    } else {
        digitalWrite(ledPin, LOW); // Éteindre la LED
    }
}
```

```
}

// Communication série (si nécessaire)
// Par exemple, envoyer les données lues ou traitées à l'ordinateur
Serial.println(processedValue);

// Attente avant la prochaine itération de la boucle
delay(1000); // Attendre 1 seconde
}
```

Inclusion des Bibliothèques

```
#include <LibraryName.h>
```

- Incluez toutes les bibliothèques nécessaires pour votre projet en haut du fichier.
- Les bibliothèques fournissent des fonctionnalités supplémentaires et des interfaces pour interagir avec divers composants matériels et logiciels.

Déclaration des Constantes et des Variables Globales

```
const int ledPin = 13;  
int sensorValue = 0;
```

- Utilisez const pour les valeurs qui ne changeront jamais (par exemple, les numéros de broche).
- Déclarez les variables globales en dehors des fonctions pour qu'elles puissent être utilisées dans tout le programme.

Section de Configuration - setup()

```
void setup() {  
    pinMode(ledPin, OUTPUT);  
    pinMode(A0, INPUT);  
    Serial.begin(9600);  
    digitalWrite(ledPin, LOW);  
}
```

- **Initialisation des broches** : Configurez les broches comme entrées ou sorties à l'aide de pinMode().
- **Communication série** : Initialiser la communication série avec Serial.begin(). Il faut que la valeur entre parenthèses soit la même au niveau dans la partie setup(), dans le moniteur série lorsque vous testez sur l'IDE Arduino dans l'initialisation de la communication avec Arduino dans Godot dans le Arduino Manager.

- **Initialisation des composants** : Si vous utilisez des bibliothèques, initialisez vos composants ici (par exemple, écran LCD, capteurs, etc.).
- **Configuration initiale** : Mettez en place l'état initial des broches et des variables.

Boucle Principale - loop()

```
void loop() {  
    sensorValue = analogRead(A0);  
    int processedValue = sensorValue / 4;  
    if (processedValue > 100) {  
        digitalWrite(ledPin, HIGH);  
    } else {  
        digitalWrite(ledPin, LOW);  
    }  
    Serial.println(processedValue);  
    delay(1000);  
}
```

- **Lecture des capteurs** : Utilisez `analogRead()` ou `digitalRead()` pour lire les valeurs des capteurs.
- **Traitement des données** : Effectuez tout traitement nécessaire sur les données lues (par exemple, conversion, filtrage).
- **Contrôle des sorties** : Utilisez `digitalWrite()` pour contrôler les sorties (par exemple, allumer ou éteindre une LED).
- **Communication série** : Envoyez les données traitées à l'ordinateur pour le débogage ou l'enregistrement.
- **Délais** : Utilisez `delay()` pour attendre un certain temps avant la prochaine itération de la boucle (utile pour réduire la fréquence de lecture des capteurs).

Meilleures Pratiques

Commentaires : Utilisez des commentaires pour expliquer ce que fait chaque section du code. Cela facilite la compréhension et la maintenance.

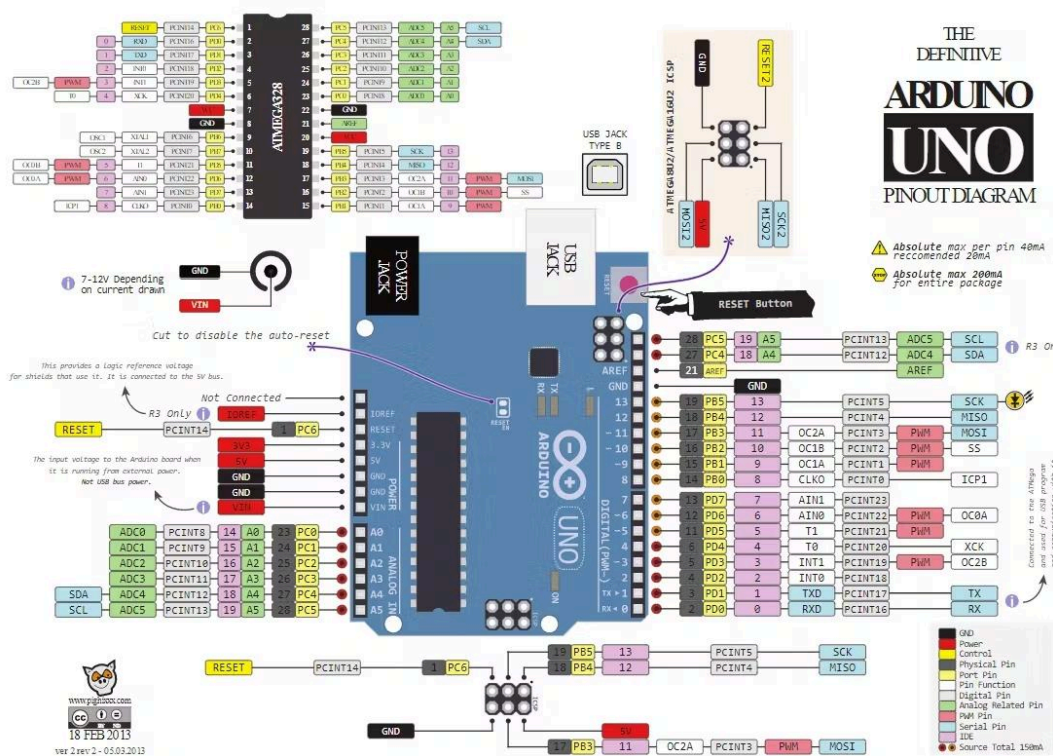
Constantes : utilisez des constantes (`const`) pour les valeurs qui ne changent pas. Cela rend le code plus lisible et plus facile à modifier.

Fonctions : Si votre code devient complexe, décomposez-le en plusieurs fonctions. Cela améliore la lisibilité et la réutilisabilité.

Serial Monitor : utilisez le moniteur série pour le débogage. Cela vous aide à comprendre ce que fait votre programme et à identifier les problèmes. **Lorsque vous lancez le jeu sur Godot, veillez à ce que le moniteur série soit coupé sur l'IDE Arduino sinon ça ne marchera pas.**

Optimisation des Delays : Soyez prudent avec delay(), car il bloque l'exécution du programme. Utilisez plutôt la fonction millis() pour des tâches qui nécessitent une temporisation sans bloquer le programme.


Les différentes broches de l'Arduino (pins)



Broches d'alimentation

5V : Cette broche fournit une tension de 5 volts régulée, qui est souvent utilisée pour alimenter les capteurs et autres composants électroniques, tous les capteurs de ce hackathon fonctionnent en 5V (On peut faire une exception avec .

GND (Ground) : Ces broches sont connectées à la masse de votre circuit. Elles sont utilisées comme référence de tension et pour compléter les circuits (c'est le 0V).



3.3V : Cette broche fournit une tension de 3.3 volts régulée, utile pour alimenter des capteurs et des dispositifs qui nécessitent une tension plus basse que 5 volts (pas besoin dans notre cas).

VIN (Voltage Input) : Cette broche permet de fournir une tension d'alimentation à la carte Arduino. Elle est connectée directement à la source d'alimentation externe, que ce soit une batterie ou un adaptateur secteur (**ne pas utiliser**).

Broches de lecture des capteurs

Broches Numérique

Ce sont des broches qui peuvent être configurées pour envoyer ou recevoir des signaux numériques (logiques 0 ou 1). Elles peuvent être utilisées pour contrôler des LED, des moteurs, des relais mais aussi et surtout lire l'état d'un bouton, d'un capteur ultrason, etc.

Broches Analogiques

Ces broches sont utilisées pour lire des valeurs de tension analogiques (plage de 0 à 5 volts). Elles peuvent être utilisées pour lire les données provenant de capteurs analogiques tels que les différents types de potentiomètres qui vous seront utiles dans ce hackathon.

Broches de Communication (à titre informatif)

Communication série

TX (Transmission) et RX (Réception) :

Ces broches sont utilisées pour la communication série (UART) avec d'autres périphériques, comme un ordinateur ou d'autres cartes Arduino. C'est par celle-ci que la liaison série est effectuée entre le moniteur série de l'IDE arduino (le logiciel arduino).

Broches PWM (Modulation de Largeur d'Impulsion) :

Ces broches sont capables de produire un signal PWM, qui peut être utilisé pour simuler une tension variable. Cela est souvent utilisé pour contrôler la vitesse des moteurs, la luminosité des LED, etc.

Broches I2C et SPI :

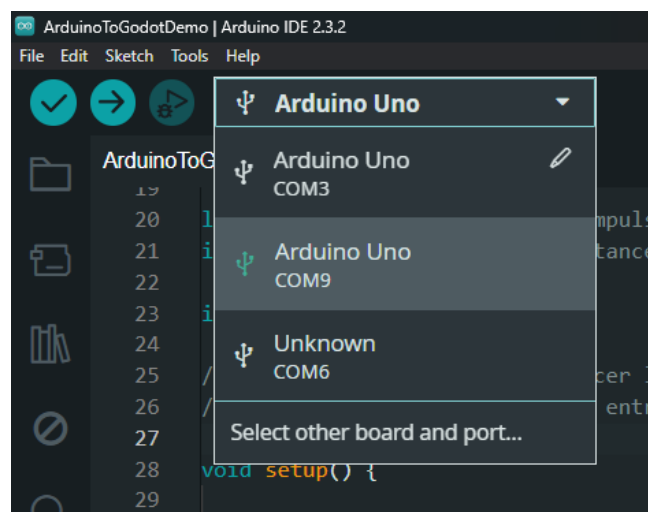
Ces broches sont utilisées pour les protocoles de communication série avancés tels que I2C (Inter-Integrated Circuit) et SPI (Serial Peripheral Interface). Ils permettent la communication avec plusieurs périphériques en utilisant un seul bus de données.

Communication sans fil

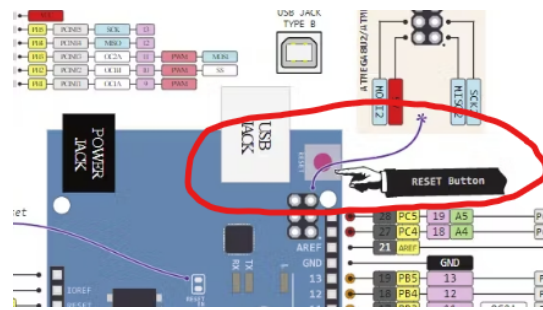
Certains micro contrôleur permettent de communiquer sans fil au travers de protocoles tels que le BLE (Bluetooth Low Energy) ou le Wifi. La carte que vous utilisez aujourd'hui ne le permet pas mais des cartes peu chères telles que l'ESP32 vous le permettrait (Certains d'entre vous ont pu s'amuser avec le système de notification Whatsapp cette année par exemple).

Les Bonnes Pratiques

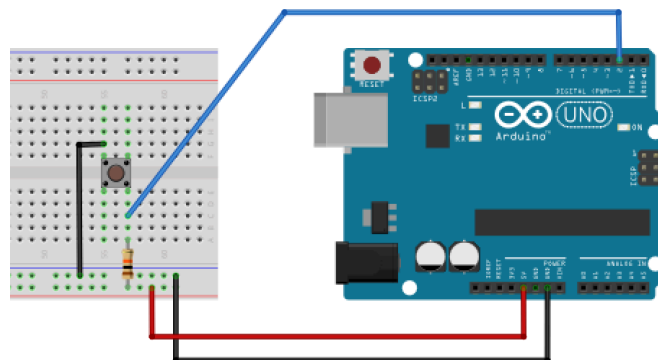
- **Rappel : Lorsque vous lancez la communication série entre Godot et Arduino, veillez à ce que le moniteur série sur Arduino ne soit pas ouvert.**
- **Rappel : Le port de la carte dans Godot doit correspondre au port choisi dans Arduino :**



- **Rappel : Ajouter un délai d'une dizaine de millisecondes entre chaque envoi de la chaîne de caractère vers Godot sinon Godot ne pourra pas suivre le rythme (étonnant non ? Ajouter du délai pour avoir moins de délai.. Si vous êtes curieux de savoir ce qui pose problème, demandez-nous !).**
- Essayez d'isoler vos petits tests avant de les intégrer dans le plus gros code afin d'éviter d'être perdu lors du debugging.
- Si vous voulez tester un code que vous avez mis dans la partie "setup()" du code arduino plusieurs fois d'affilé sans avoir besoin de brancher et rebrancher la carte ou de téléverser à nouveau, il vous suffit d'appuyer sur le bouton Reset sur la carte Arduino.



- Lorsque vous branchez un capteur ou tout autre composant à la carte, il est de bon usage de le faire quand la carte est hors tension (débrancher la carte) afin d'éviter tout faux contact entre les câbles et de cramer la carte ou les capteurs.
- Pour la lecture du bouton il y a plusieurs méthodes. Voir ces méthodes [ici](#). Vous allez avoir besoin d'un algorithme anti-rebond, et peut-être d'un algorithme pour transformer le 1 lorsque le bouton est appuyé et 0 lorsqu'il est relâché en un switch (un appui = valeur à 0 et un autre appui = valeur à 1).
- Vous allez potentiellement utiliser beaucoup de capteurs, l'Arduino aura une broche de signal disponible pour chacun d'entre eux, en revanche, celle-ci ne dispose que de 3 broches GND et une 5V. Il est de bonne usage d'utiliser une planche de prototypage pour démultiplier le nombre de connexion possible avec une broche spécifique (fil rouge pour le 5V et fil noir pour le GND) :

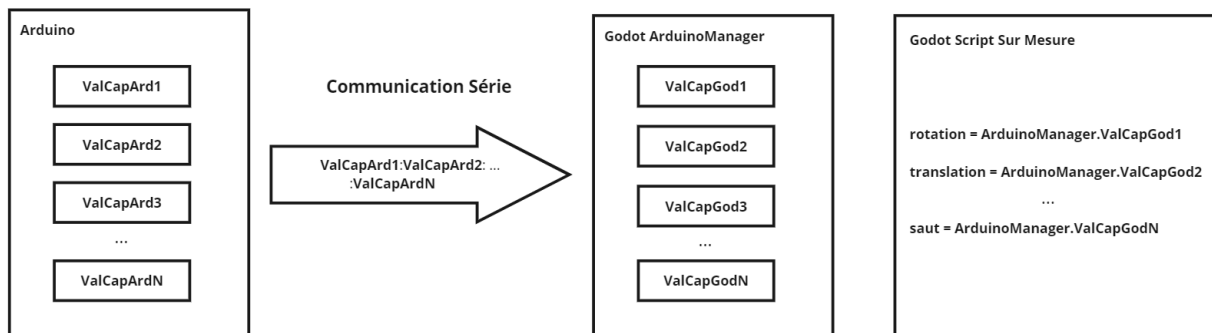


- N'hésitez pas à tester des codes simples en posant des question à ChatGPT également.

Communication Arduino à Godot (et vis-versa)

Les codes mis à disposition montrent comment l'arduino peut envoyer les données lues provenant des capteurs à Godot.

Le schéma simplifié est le suivant :



- Vous devez définir les variables des capteurs que vous voulez utiliser (tous les capteurs ne sont pas initialisés et lu de la même manière).
- Vous ajouter ces valeurs à la chaîne de caractère envoyée à Godot en prenant bien en compte la position des valeurs dans la chaîne.
- Dans Godot, vous extrayez ces valeurs en récupérant chacune d'elle à la position où vous l'avez mise dans le tableau qui sépare la chaîne de caractère.
- Dans les scripts des composants, vous faites appel à ces valeurs pour modifier des paramètres tels que la position, la rotation, la vitesse, etc.. (soyez originaux et créatifs).

Tutoriel : Mise en place de l'environnement pour la communication série

Ce tutoriel explique les étapes nécessaires pour configurer correctement .NET, installer les dépendances requises et résoudre les erreurs courantes liées à System.IO.Ports.

1. Vérification de l'environnement de développement

Avant de commencer, il est essentiel de s'assurer que l'environnement de développement est correctement configuré.

1.1 Vérification de la version de .NET

Ouvrir un terminal (PowerShell ou CMD) et exécuter la commande suivante :

```
dotnet --info
```

Le projet doit être compatible avec .NET 8. Si la version affichée est inférieure à .NET 8, il est recommandé de mettre à jour en téléchargeant la dernière version depuis le site officiel :

<https://dotnet.microsoft.com/en-us/download/dotnet/8.0>.

2. Installation et configuration de System.IO.Ports

System.IO.Ports est le package nécessaire pour gérer la communication série en C#. Il doit être installé et configuré correctement.

2.1 Vérification des packages installés

Dans le dossier du projet Godot, exécuter la commande suivante pour lister les dépendances :

```
dotnet list package
```

Si System.IO.Ports n'apparaît pas dans la liste, il doit être installé.

2.2 Installation de System.IO.Ports

Dans un terminal, exécuter :



```
dotnet add package System.IO.Ports --version 8.0.0
```

```
dotnet restore
```

Enfin, recompiler le projet :

```
dotnet build
```

3. Mise à jour du projet vers .NET 8 (si nécessaire)

Si le projet est configuré pour .NET 6.0 et que l'installation de System.IO.Ports pose problème, il est conseillé de le mettre à jour vers .NET 8.

3.1 Modification du fichier **.csproj**

Ouvrir le fichier .csproj du projet et modifier la ligne :

```
<TargetFramework>net6.0</TargetFramework>
```

La remplacer par :

```
<TargetFramework>net8.0</TargetFramework>
```

3.2 Régénération des fichiers de compilation

Exécuter les commandes suivantes :

```
dotnet clean
```

```
dotnet build
```

Cette mise à jour permet d'utiliser la dernière version de System.IO.Ports et d'améliorer la compatibilité avec Godot 4.4.