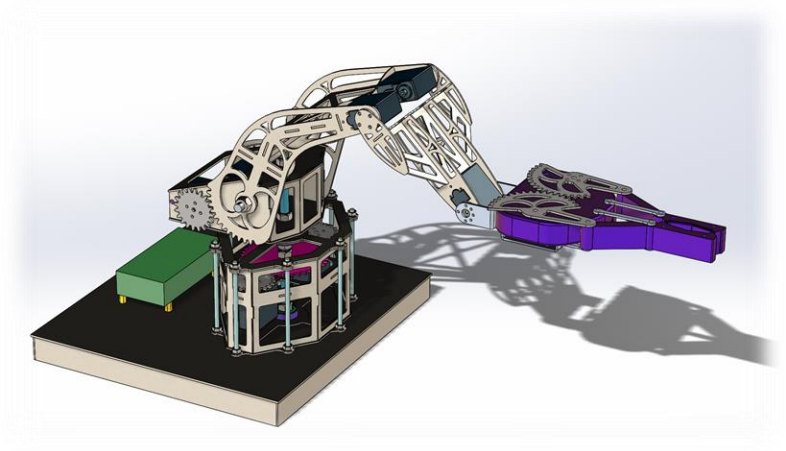
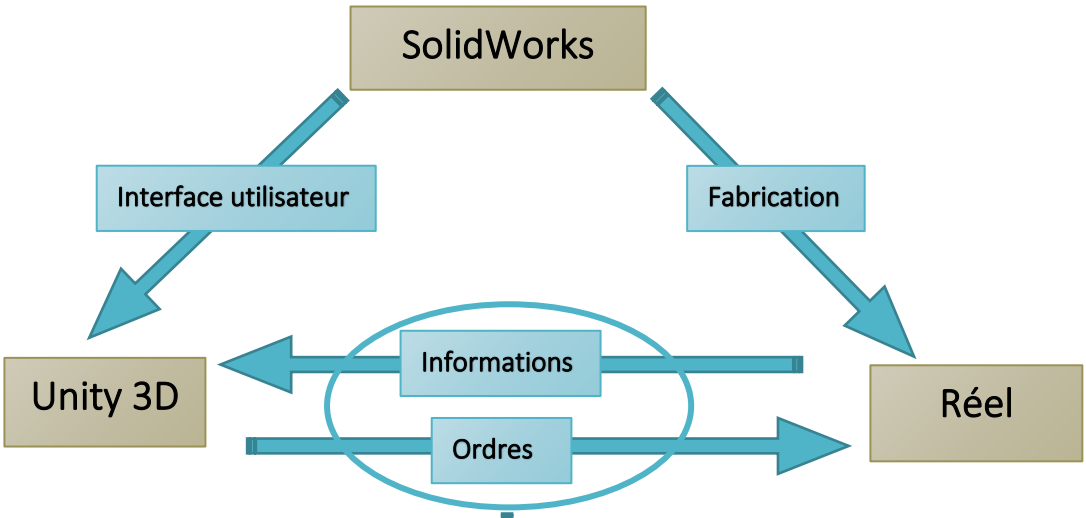
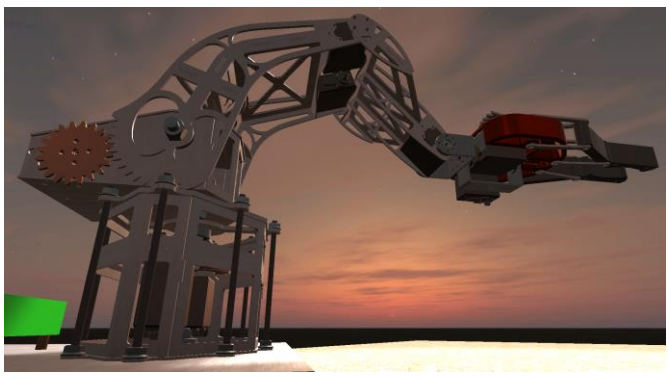
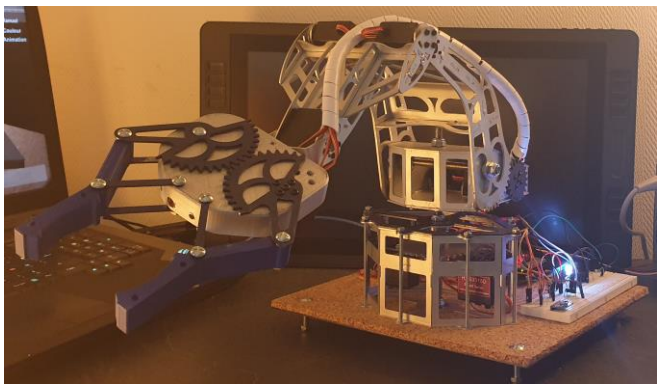
Objectif :

Cette production a pour but de mettre en évidence la continuité entre le virtuel et le réel. Dans nos référentiels, apparait régulièrement la notion d'exploitation du modèle numérique, mais que peut-on mettre derrière cette compétence aujourd'hui ? C'est ce que j'essaierai d'exposer tout au long de cette présentation.









ARDUINO

Table des matières

1- Préambule (Voir video0)

Je m'appelle Pierre Chauvin, je suis enseignant en construction mécanique, j'ai réalisé cette année ce thème du bras robotisé avec mes élèves de chaudronnerie (1ère TCI) et d'électrotechnique (1ère Meelec). Bien évidemment, ce ne sont pas les élèves qui l'ont conçu, mais ils ont participé à presque chacune des étapes suivant leurs compétences et leur spécialité.

Je pense qu'il est de notre rôle, en tant qu'enseignant de construction, d'être initiateur de projets liant différents corps de métier et différentes disciplines. De plus, notre métier évoluant suivant les avancées technologiques, il est de notre devoir d'anticiper les compétences professionnelles futures dont nos élèves auront besoin afin de se démarquer face à un marché de l'emploi de plus en plus encombré.

Les modeleurs volumiques, que maitrisent les concepteurs, fournissent un modèle numérique qui est exploitable pour la fabrication, mais pas que… En effet, il permet également de créer les interfaces utilisateurs des objets connectés qui font partie aujourd'hui de notre quotidien. Sans être moi-même ni électronicien, ni développeur professionnel, j'ai pu très facilement créer cette interface. L'expérience que j'en tire, tant pédagogiquement que personnellement, me donne envie de la partager.

Même si nos référentiels ne parlent pas encore de cette nouvelle dynamique de travail, il est intéressant et même indispensable que nos élèves aient un certain niveau d'acquisition taxonomique de ces nouvelles compétences. Le but n'est bien évidement pas qu'ils maitrisent le processus mais qu'ils aient conscience du lien qui existe aujourd'hui entre le triptyque modèle numérique/ interface utilisateur/ réel.

Les solutions apportées ici sont gratuites, une version payante de l'asset Arduinty est disponible pour une quarantaine d'euros mais n'est pas nécessaire (hors DHT11 et HC SR04). **Les éléments suivants sont nécessaires pour pouvoir commencer** :

- **Solidworks 2018 minimum** (Le modèle numérique du bras robotisé est disponible sur la page Eduscol si vous voulez le réaliser)
- **Unity 2019.1.14** La version est importante et disponible à l'adresse suivante : https://unity3d.com/get-unity/download/archive
- **Asset Arduinity** (Voir vidéo1 : connexion Unity/ Arduino)
- **Kit d'apprentissage Arduino (uno ou mega)** disponible pour une trentaine d'euros

2- Connexion Unity/Arduino (Voir Video1)

Dans cette partie on va voir comment connecter Arduino à Unity grâce à un asset appelé Arduinity. (Voir Vidéo1)

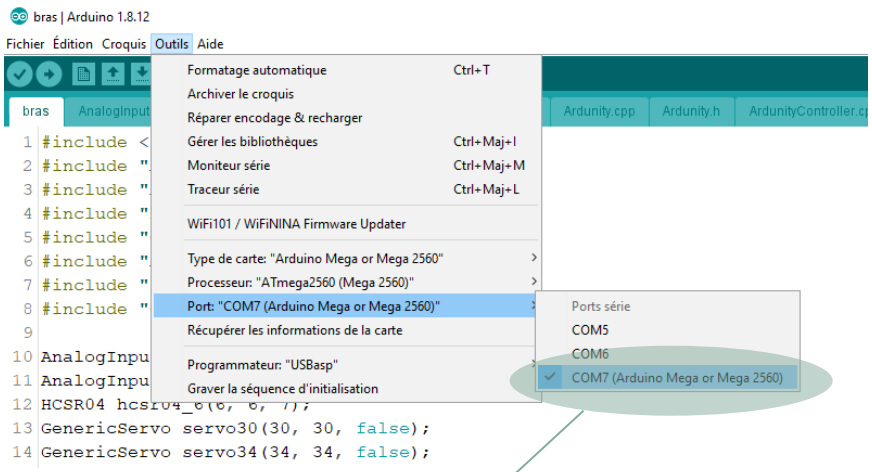
Important :

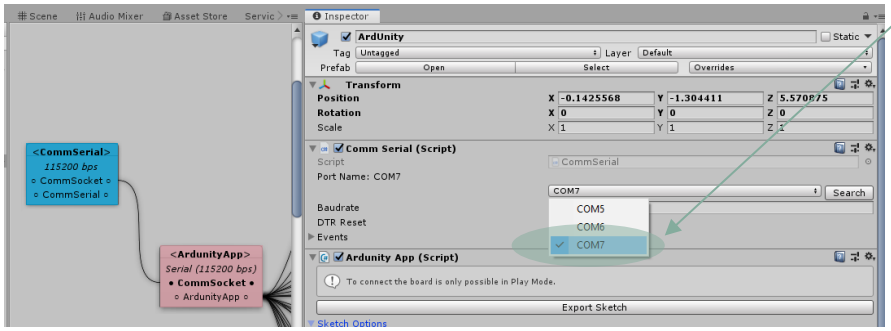
Ardunity utilise un cadriciel, en anglais un « Framework », Un Framework est une collection de code tout fait et prêt à l'usage pour les développeurs, Ardunity utilise une version de Framework Net2.0 qu'Unity est sur le point d'abandonner. Les versions au-delà de la 2019.1.14 n'ont plus de Framework Net2.0. Le développeur de l'asset Ardunity travail sur une mise à jour permettant la compatibilité avec Net4.0 et sera bientôt de nouveau compatible avec les nouvelles versions d'Unity. En attendant il est important de travailler dans la bonne version de Unity.

Lors de cette première phase, on va venir lier Unity à Arduino via son port COM qui peut être différent suivant les configurations. Ce port COM est affecté à votre carte Arduino lors de sa première connexion à votre ordinateur.

Qu'est-ce qu'un port COM :

Le port COM est un port série permettant la communication entre un périphérique et l'ordinateur (une souris, un clavier, ou une carte Arduino par exemple). Il peut être filaire comme dans notre cas (câble USB) mais il peut être également sans fils comme dans le cas de l'utilisation du Bluetooth.



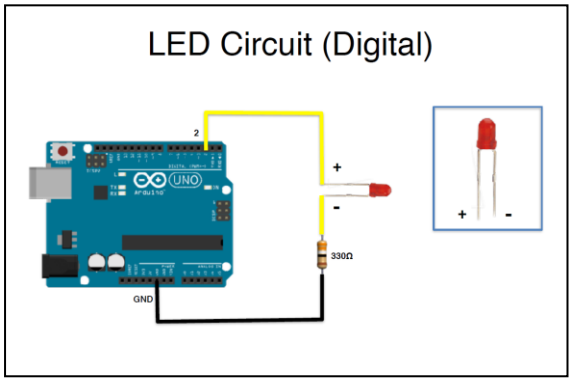


Ce port doit être renseigné dans Unity dans le composant « CommSerial ». Une fois le premier script exporter puis téléverser dans la carte Arduino, la carte est connectée et reconnue par Unity.

3- Les Leds (Voir video2)

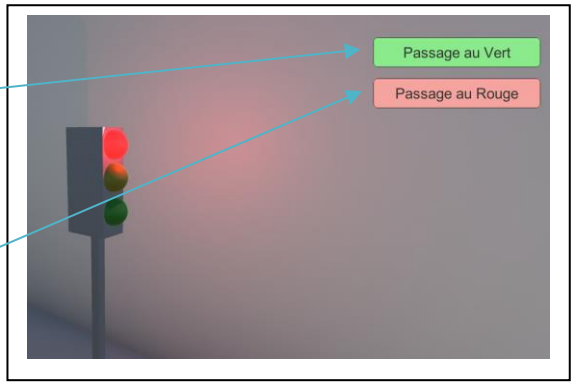
```
using System.Collections;;
using UnityEngine;
using Ardunity;

public class FeuxTricolores : MonoBehaviour
{
    public GameObject rouge, orange, vert;
    public Material rougeMat, orangeMat, vertMat;
    bool rougeValue, orangeValue, vertValue;
    public GameObject pointRouge, pointOrange, pointVert;

    void Start()
    {
        rougeValue = rouge.GetComponent<DigitalOutput>().Value= true;
        orangeValue = orange.GetComponent<DigitalOutput>().Value = false;
        vertValue = vert.GetComponent<DigitalOutput>().Value = false;
    }

    // Update is called once per frame
    void Update()
    {
        rougeValue = rouge.GetComponent<DigitalOutput>().Value;
        orangeValue = orange.GetComponent<DigitalOutput>().Value;
        vertValue = vert.GetComponent<DigitalOutput>().Value;

        if (rougeValue == true)
        {
            rougeMat.EnableKeyword("_EMISSION");
            pointRouge.SetActive(true);
        }
        else
        {
            rougeMat.DisableKeyword("_EMISSION");
            pointRouge.SetActive(false);
        }

        if (orangeValue == true)
        {
            orangeMat.EnableKeyword("_EMISSION");
            pointOrange.SetActive(true);
        }
        else
        {
            orangeMat.DisableKeyword("_EMISSION");
            pointOrange.SetActive(false);
        }

        if (vertValue == true)
        {
            vertMat.EnableKeyword("_EMISSION");
            pointVert.SetActive(true);
        }
        else
        {
            vertMat.DisableKeyword("_EMISSION");
            pointVert.SetActive(false);
        }
    }

    public void PassageAuVert()
    {
        StartCoroutine(PassageV());
    }

    IEnumerator PassageV()
    {
        rougeValue = rouge.GetComponent<DigitalOutput>().Value = false;
        yield return new WaitForSeconds(0.5f);
        orangeValue = orange.GetComponent<DigitalOutput>().Value=true;
        yield return new WaitForSeconds(2);
        orangeValue = orange.GetComponent<DigitalOutput>().Value=false;
        yield return new WaitForSeconds(0.5f);
        vertValue = vert.GetComponent<DigitalOutput>().Value=true;
    }

    public void PassageAuRouge()
    {
        StartCoroutine(PassageR());
    }

    IEnumerator PassageR()
    {
        vertValue = vert.GetComponent<DigitalOutput>().Value = false;
        yield return new WaitForSeconds(0.5f);
        orangeValue = orange.GetComponent<DigitalOutput>().Value = true;
        yield return new WaitForSeconds(2);
        orangeValue = orange.GetComponent<DigitalOutput>().Value = false;
        yield return new WaitForSeconds(0.5f);
        rougeValue = rouge.GetComponent<DigitalOutput>().Value = true;
    }

}
```

Dans ce premier exemple d'exploitation de composant, on va commencer par un exemple simple d'utilisation de Leds.dans un feu tricolore. L'objectif est de créer un petit script permettant l'enchainement de l'allumage des Leds via des temporisations appelées « Couroutine »



L'allumage des Leds est synchronisé avec les émissions de lumière des textures des capsules dans Unity. Chacune des méthodes est affectée aux boutons correspondants.

Interface utilisateur



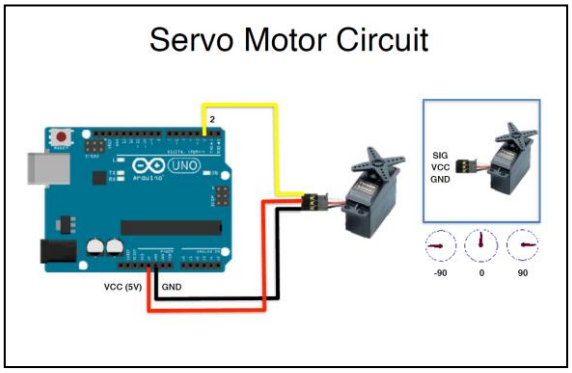
4- Les servomoteurs (voir vidéo3)

Dans cette partie nous verrons comment réaliser le triptyque complet modèle numérique/ interface utilisateur/ réel. On va partir de notre modèle numérique Solidworks puis l'exploiter dans Unity et enfin le contrôler via une carte Arduino. Pour ce faire, vous aurez besoin des éléments suivants :

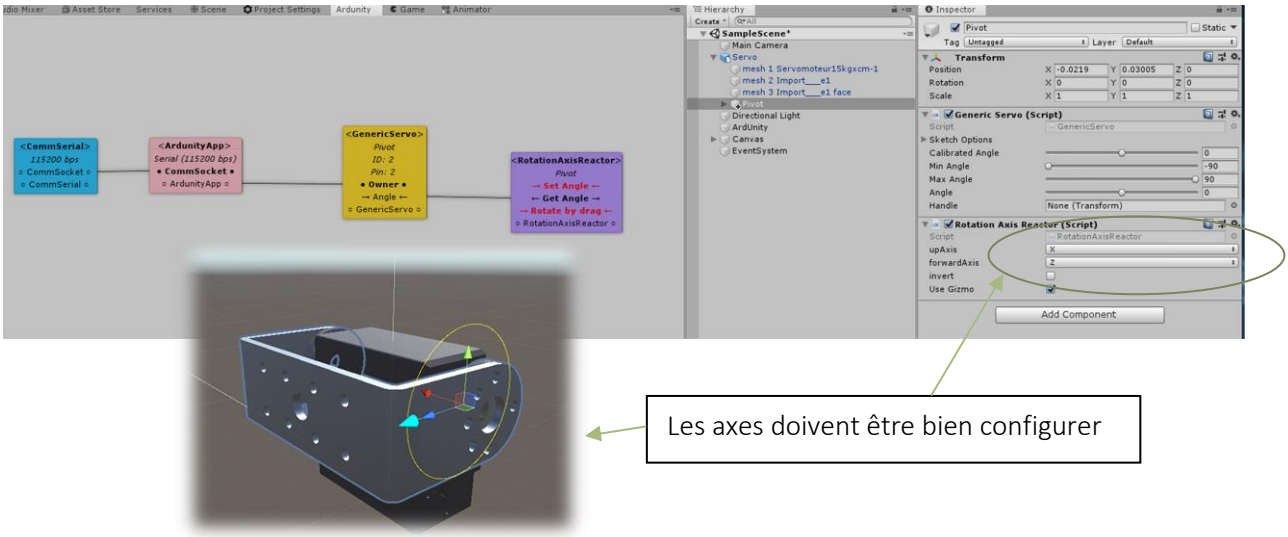
- Macro d'export en «.obj» pour Solidworks :disponible à l'adresse suivante : http://notepad.xavierdetourbet.com/exporter-fichier-obj-depuis-solidworks/

- Le logiciel Blender permettant de convertir les « .obj » en « .fbx » disponible à l'adresse suivante : https://www.blender.org/download/releases/2-83/

Le câble des servomoteurs se fait de la manière suivante :



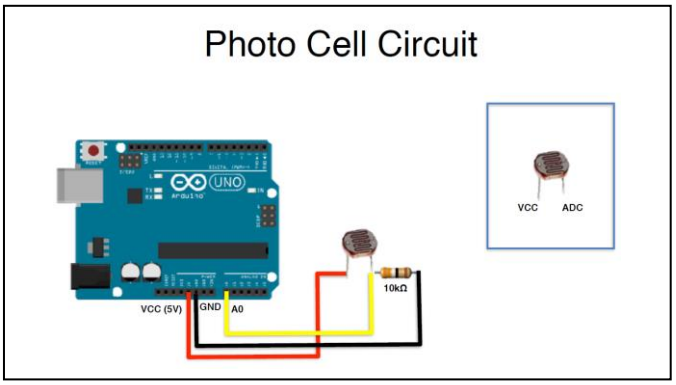
Les blueprints à installer dans la fenêtre Ardunity sont les suivants :



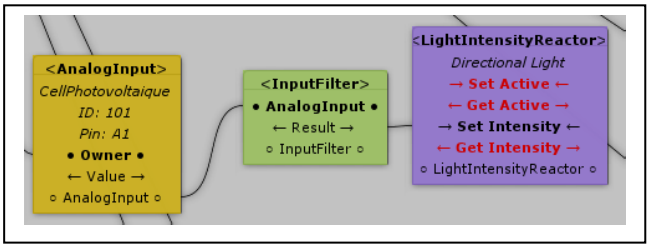
5- La cellule photovoltaïque (Voir vidéo4)

La cellule photovoltaïque permet de détecter une quantité de lumière. Dans notre exemple, la luminosité de la scène dans Unity est directement liée à la luminosité ambiante détectée par la cellule. Dans l'exemple du bras robotisé cela n'a pas trop d'intérêt, je le reconnais aisément. Mais en domotique par exemple, cette information prend tout son sens. En effet plus la luminosité extérieure est importante moins on a besoin de lumière dans la maison. On peut donc aisément lier la quantité de lumière à fournir dans la maison en fonction de la quantité de lumière en extérieure.

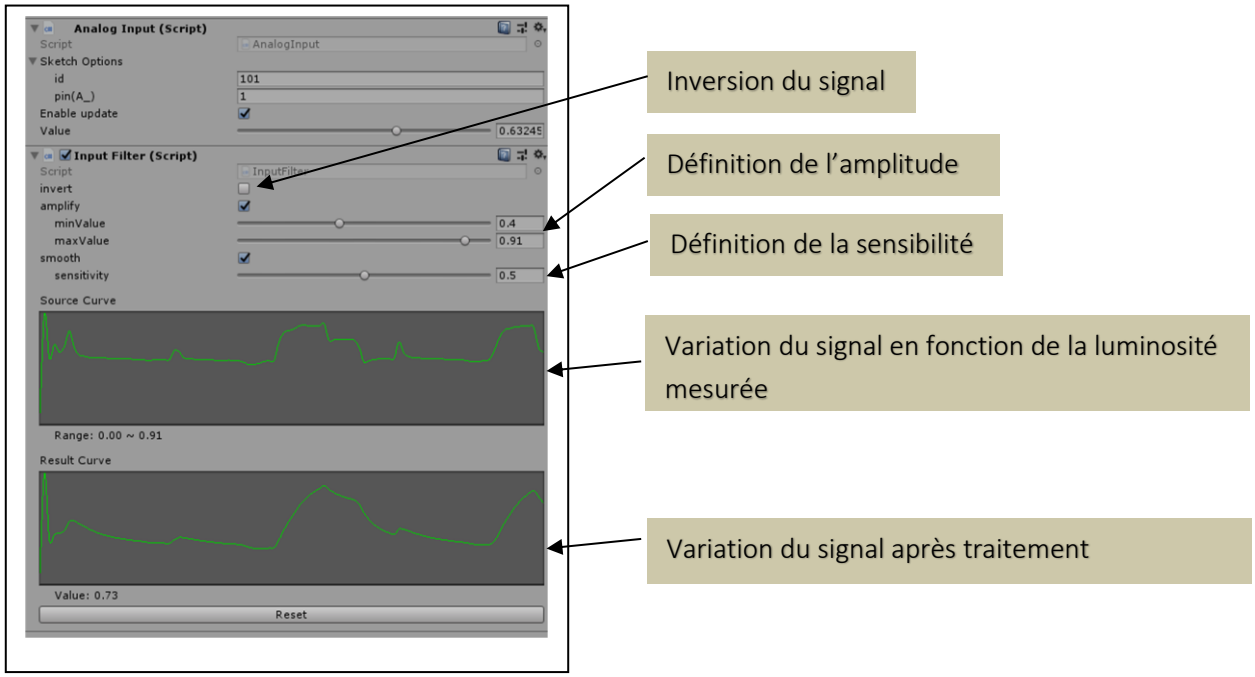
Pour connecter une cellule photovoltaïque procéder de la manière ci-contre.



Les blueprints à utiliser sont les suivants :



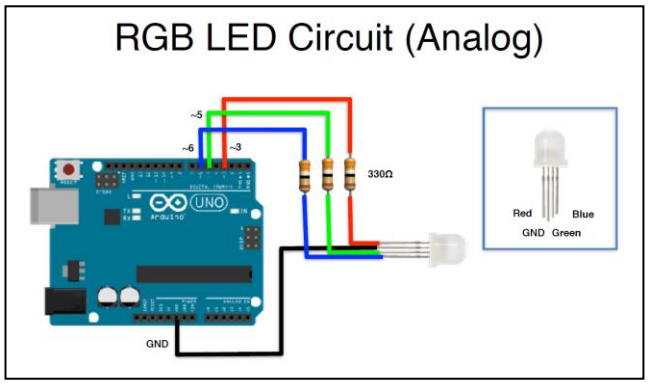
Lors de l'utilisation, on peut voir l'évolution du signal reçu dans le composant « InputFilter » qui permet de maitriser les valeurs mini et maxi ainsi que l'amplification et la sensibilité. C'est également dans ce composant qu'on peut inverser le signal.



6- Les Leds RGB (Voir vidéo5)

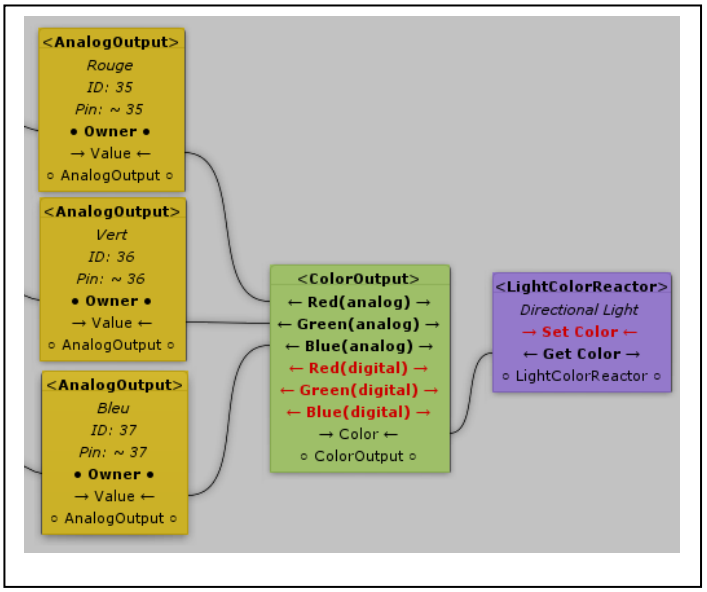
Les Leds RGB sont des leds pouvant prendre n'importe quelle couleur. Dans notre cas, la couleur de l'éclairage de la scène dans Unity est liée à la couleur de la Led RGB. Un asset gratuit (Flexible Color Picker) permet de choisir facilement la couleur via une interface graphique.

Le câblage de la Led RGB est le suivant :



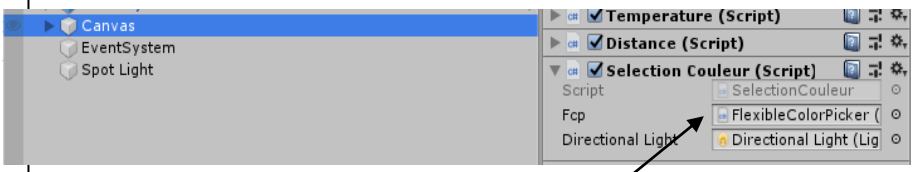
La couleur de la lumière ambiante de la scène dans Unity est appelée « Directional Light » Sa couleur est synchronisée avec la Led RGB via un script disponible ci-dessous.

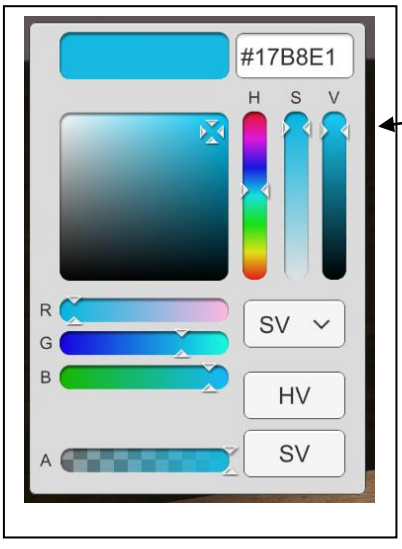
Les blueprints utilisés sont les suivants :




```
1  using UnityEngine;
2
3  public class SelectionCouleur : MonoBehaviour
4  {
5      public FlexibleColorPicker fcp;
6      public Light directionalLight;
7
8
9
10     void Update()
11     {
12         directionalLight.color = fcp.color;
13     }
14 }
```

Ce script est placé sur le « Canvas ». Puis il faut lui renseigner le Fcp (Flexible color picker) ainsi que la « Directional Light ».





Flexible color picker

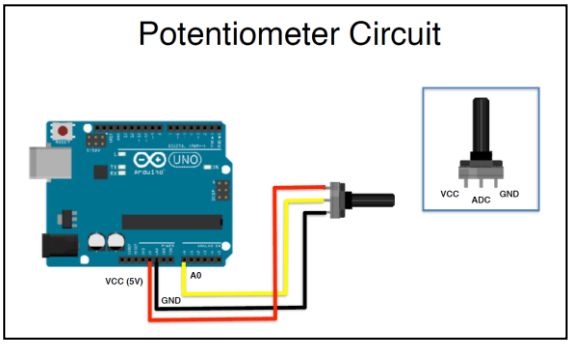
Ce composant permet de sélectionner la couleur en venant cliquer directement sur la couleur voulue. Ici la couleur est gérée par l'utilisateur mais on peut très bien imaginer que la couleur de la Led RGV évolue en fonction de la zone dans laquelle se trouve le bras. Dans une zone de sécurité, la Led s'éclaire en vert et en rouge dans une Zone de danger. L'ensemble peut être détecté facilement par l'intermédiaire de « Box Collider » dont le lien de la documentation est ci-dessous.

https://docs.unity3d.com/ScriptReference/BoxCollider.html

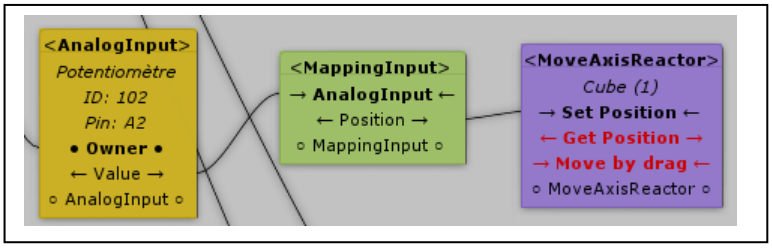
7- Le potentiomètre (Voir vidéo6)

Dans l'exemple suivant, nous allons voir les potentiomètres. Dans un souci d'aborder toutes les fonctions de cet asset, j'ai affecté le potentiomètre à une translation d'un GameObject plutôt qu'à une rotation évoquée lors de la présentation des servomoteurs. Mais j'ai réaffecté le potentiomètre à l'ouverture de la main par la suite, ce qui a plus de sens. L'exemple qui suit approfondit le blueprint « MoveAxisReactor »

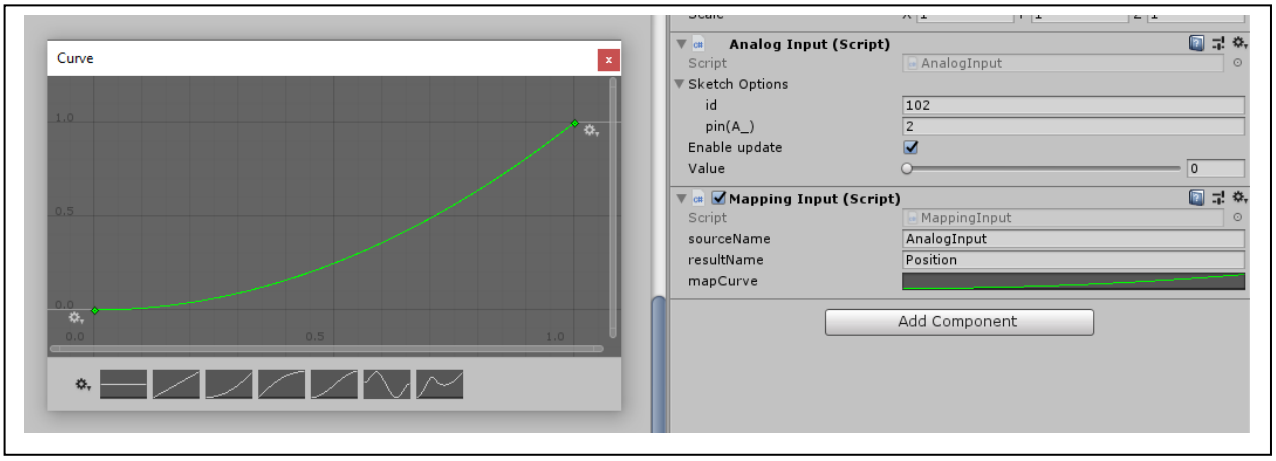
Le câblage du potentiomètre est le suivant :



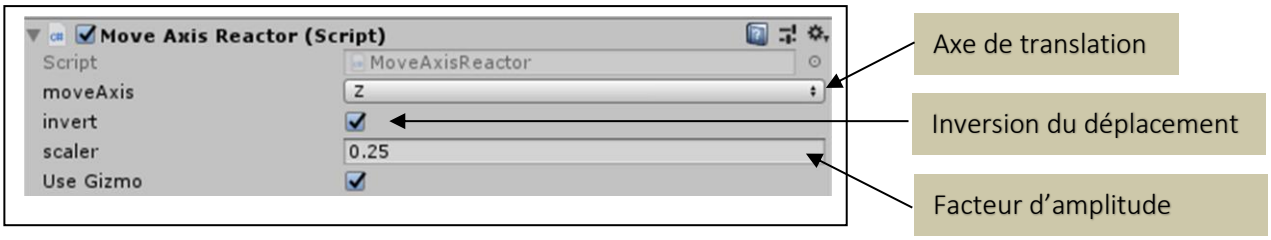
Les blueprints utilisés dans ce cas sont les suivants :



Le blueprint « Mapping Input » permet un traitement du signal de sortie. Il est possible de choisir le type de courbe qui réagira le signal de sortie : Exponentiel, logarithmique, personnalisé, tout est possible dans cette section.

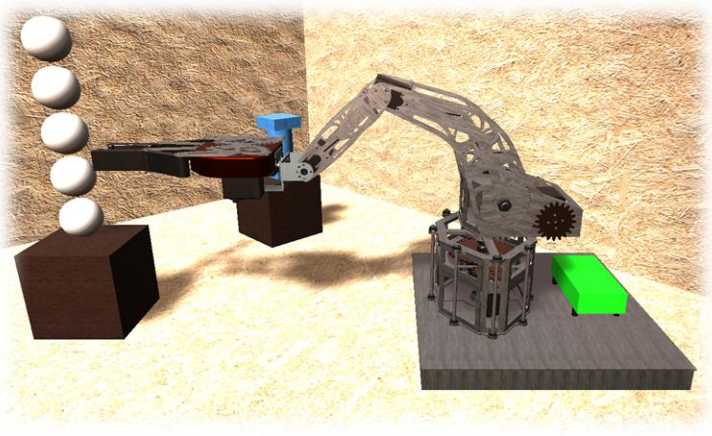


Le blueprint « MoveAxisReactor » permet de modifier la position d'un GameObject sur un axe spécifique, un facteur d'échelle peut être affecté afin de modifier l'amplitude du débattement. Il est également possible d'inverser le déplacement.

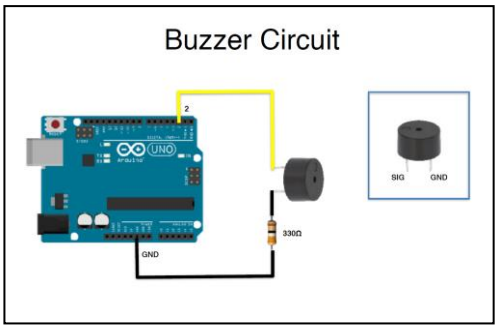


8- Le Buzzer (Voir vidéo7)

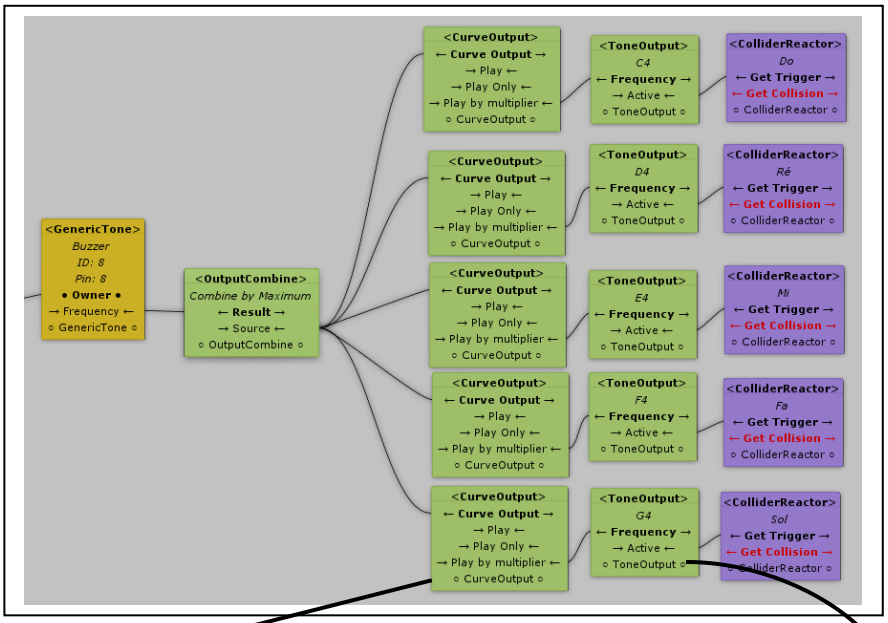
Le buzzer permet d'envoyer un signal sonore spécifique. Il peut être moduler et jouer une note spécifique. Dans l'exemple suivant la collision entre l'extrémité du bras et les sphères est détectée. Chacune des sphères déclenche une note différente. On peut donc imaginer un scénario où le buzzer avertirait de la présence du bras dans une zone spécifique avec une synchronisation de la couleur rouge de la Led RGB.



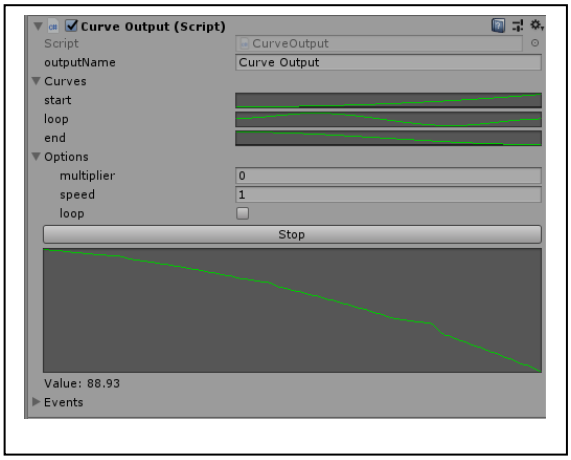
Le câblage du buzzer est le suivant :



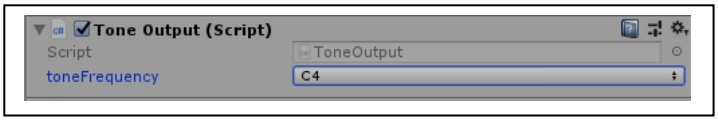
Les blueprints utilisés dans cet exemple sont les suivants :



Les blueprints « ColliderReactor » permette la détection de toute collision. Pour fonctionner, le GameObject concerné doit obligatoirement posséder un composant « Box Collider ».



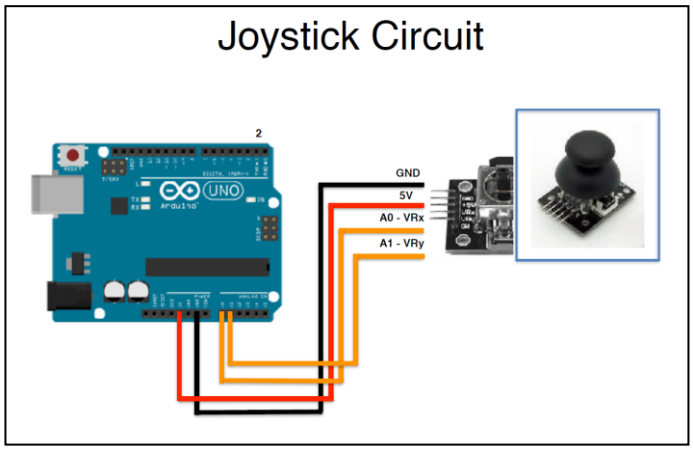
Le blueprint « CurveOutput » permet de moduler le son du buzzer. On peut modifier l'attaque de la note, son vibrato et la manière dont elle file avec les courbes respectivent « start », « loop », « end ».

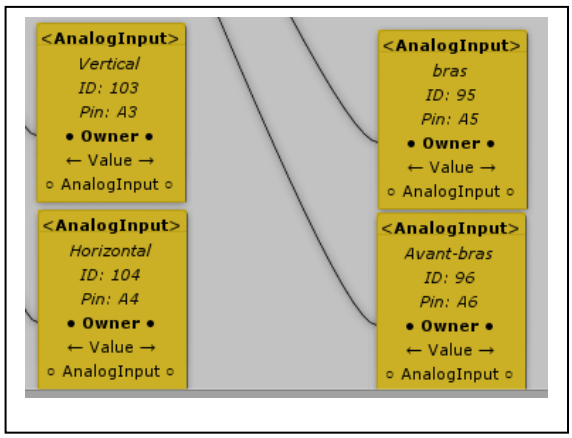


Le blueprint « Tone Output » permet de choisir la note de sortie du buzzer.

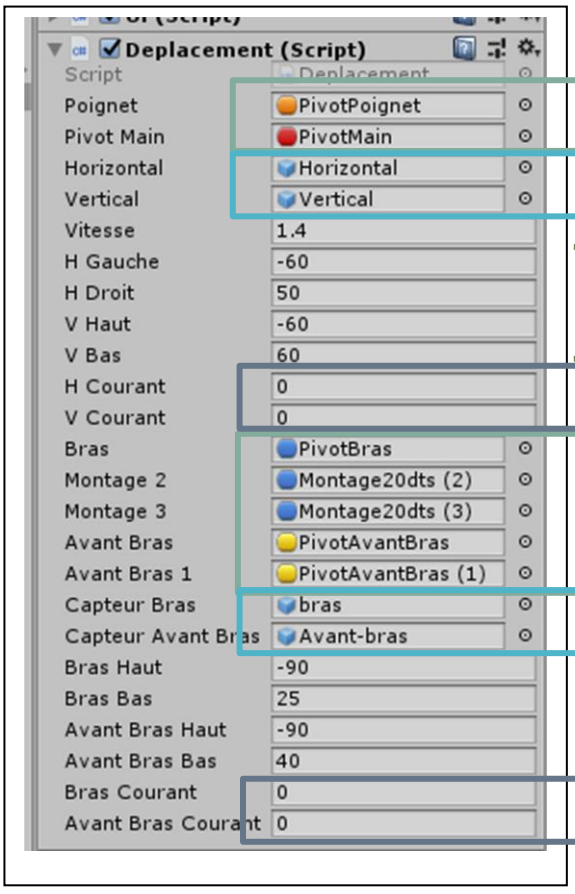
9- Le joystick (Voir vidéo8)

Le joystick n'est ni plus ni moins que 2 potentiomètres croisés. Pour l'élaboration du schéma des blueprints c'est donc le même principe que le potentiomètre (section 7). Dans le cas du bras robotisé, je n'avais que 2 joysticks que j'ai affectés aux rotations de la main ainsi qu'au bras et avant-bras. Le câblage est celui-ci contre.





Le script « Deplacement » gère la rotation des servomoteurs et la position des butées de chaque articulation. Il se base sur 4 valeurs renvoyées par les 2 joysticks.



GameObject possédant les blueprints « GenericServo »

GameObject possédant les blueprints « AnalogInput »

Compléter les valeurs des butées inférieure et supérieure

Valeurs courantes des servomoteurs (information)

GameObject possédant les blueprints « GenericServo »

GameObject possédant les blueprints « AnalogInput »

Compléter les valeurs des butées inférieure et supérieure

Valeurs courantes des servomoteurs (information)

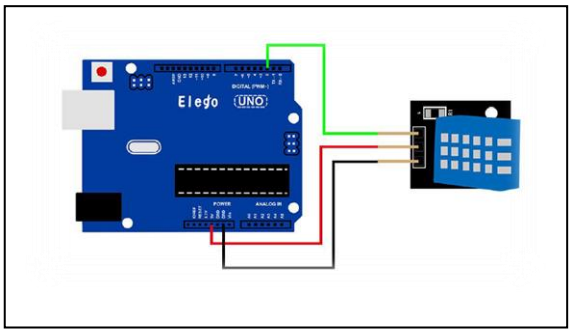
Script de déplacement

```
using UnityEngine;
using Ardunity;

public class Deplacement : MonoBehaviour
{
    public GameObject poignet, pivotMain, horizontal, vertical;

    float hor, ver;
    public float vitesse=1f;
    public float hGauche=-10f, hDroit=10f, vHaut=-10f, vBas=10f, hCourant, vCourant;
    bool peutVert=true;
    bool peutHor=true;

    public GameObject bras,montage2, montage3, avantBras,avantBras1, capteurBras, capteurAvantBras;

    float hor2, ver2;
    public float brasHaut = -10f, brasBas = 10f, avantBrasHaut = 10f, avantBrasBas = -10f, brasCourant, avantBrasCourant;
    bool peutBras = true;
    bool peutAvantBras = true;


    void Start()
    {
        hCourant = pivotMain.GetComponent<GenericServo>().angle;
        vCourant = poignet.GetComponent<GenericServo>().angle;
        hor = ((Mathf.Round((horizontal.GetComponent<AnalogInput>().Value - 0.5f) * 200)) / 100) * vitesse;
        ver = ((Mathf.Round((vertical.GetComponent<AnalogInput>().Value - 0.5f) * 200)) / 100) * vitesse;

        brasCourant = montage3.GetComponent<GenericServo>().angle;
        avantBrasCourant = avantBras.GetComponent<GenericServo>().angle;
        hor2 = ((Mathf.Round((capteurBras.GetComponent<AnalogInput>().Value - 0.5f) * 200)) / 100) * vitesse;
        ver2 = ((Mathf.Round((capteurAvantBras.GetComponent<AnalogInput>().Value - 0.5f) * 200)) / 100) * vitesse;
    }


    void Update()
    {
        hCourant = pivotMain.GetComponent<GenericServo>().angle;
        vCourant = poignet.GetComponent<GenericServo>().angle;
        hor = ((Mathf.Round((horizontal.GetComponent<AnalogInput>().Value - 0.5f) * 200)) / 100)*vitesse;
        ver = ((Mathf.Round((vertical.GetComponent<AnalogInput>().Value - 0.5f) * 200)) / 100)*vitesse;

        brasCourant = montage3.GetComponent<GenericServo>().angle;
        avantBrasCourant = avantBras.GetComponent<GenericServo>().angle;
        hor2 = ((Mathf.Round((capteurBras.GetComponent<AnalogInput>().Value - 0.5f) * 200)) / 100) * vitesse;
        ver2 = ((Mathf.Round((capteurAvantBras.GetComponent<AnalogInput>().Value - 0.5f) * 200)) / 100) * vitesse;

        // Détection des butées horizontales main
        if ((hCourant <= hGauche & hor >= 0) | (hCourant >= hDroit & hor <= 0))
        {
            peutVert = false;
        }
        else
        {
            peutVert = true;
        }
        if (peutVert == true)
        {
            if (hor <= -0.05 | hor >= 0.05)
            {
                pivotMain.transform.Rotate(0, 0, -hor);
            }
            else
            {
                pivotMain.transform.Rotate(0, 0, 0);
            }
        }
```

```
68          // Détection des butées verticales main
69          if ((vCourant <= vHaut & ver <= 0) | (vCourant >= vBas & ver >= 0))
70          {
71              peutHor = false;
72          }
73          else
74          {
75              peutHor = true;
76          }
77          if (peutHor == true)
78          {
79              if (ver <= -0.05 | ver >= 0.05)
80              {
81                  poignet.transform.Rotate(ver, 0, 0);
82              }
83              else
84              {
85                  poignet.transform.Rotate(0, 0, 0);
86              }
87          }
88
89          // Détection des butées bras
90          if ((brasCourant <= brasHaut & hor2 <= 0) | (brasCourant >= brasBas & hor2 >= 0))
91          {
92              peutBras = false;
93          }
94          else
95          {
96              peutBras = true;
97          }
98          if (peutBras == true)
99          {
100             if (hor2 <= -0.05 | hor2 >= 0.05)
101             {
102                 bras.transform.Rotate(hor2/2, 0, 0);
103                 montage2.transform.Rotate(0, 0,  hor2);
104                 montage3.transform.Rotate(0, 0,  -hor2);
105
106             }
107             else
108             {
109                 bras.transform.Rotate(0, 0, 0);
110                 montage2.transform.Rotate(0, 0, 0);
111                 montage3.transform.Rotate(0, 0, 0);
112             }
113         }
114         // Détection des butées avantBras
115         if ((avantBrasCourant <= avantBrasHaut & ver2 >= 0) | (avantBrasCourant >= avantBrasBas & ver2 <= 0))
116         {
117             peutAvantBras = false;
118         }
119         else
120         {
121             peutAvantBras = true;
122         }
123         if (peutAvantBras == true)
124         {
125             if (ver2 <= -0.05 | ver2 >= 0.05)
126             {
127                 avantBras.transform.Rotate(ver2, 0, 0);
128                 avantBras1.transform.Rotate(ver2, 0, 0);
129             }
130             else
131             {
132                 avantBras.transform.Rotate(0, 0, 0);
133                 avantBras1.transform.Rotate(0, 0, 0);
134             }
135         }
136     }
137 }
```

10- La sonde d'humidité et de température DHT11 (Voir vidéo9)

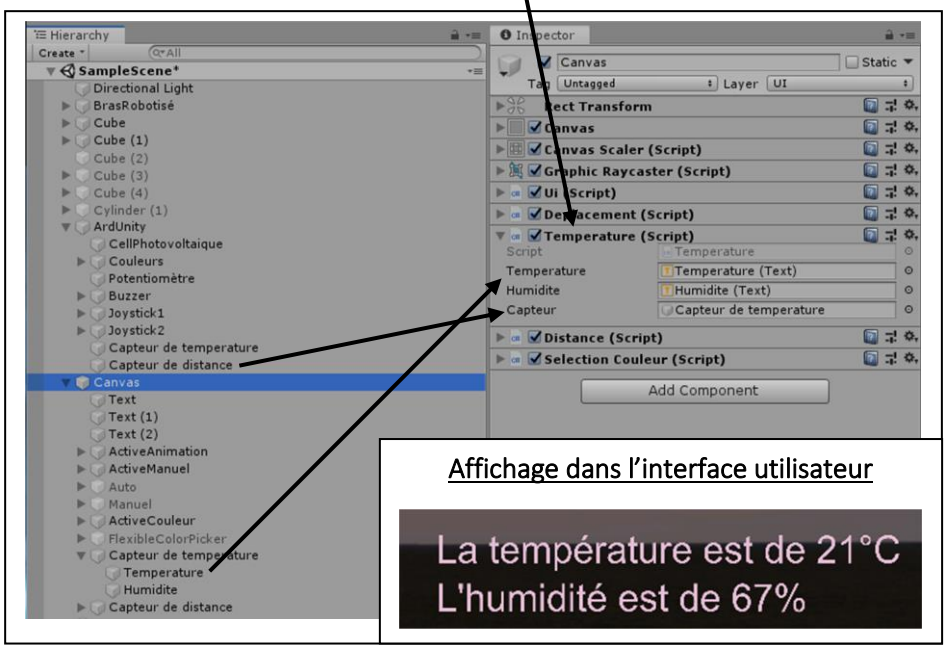
La gestion de la sonde de température et d'humidité DHT11 est disponible sur la version payante de Ardunity (44,67€). Elle se câble comme dans l'exemple ci-contre. Dans notre exemple, on veut afficher les informations de cette sonde dans l'interface utilisateur. Afin de pouvoir exploiter les informations reçues par le blueprint « DHT11 », il faut passer par un script disponible ci-dessous :

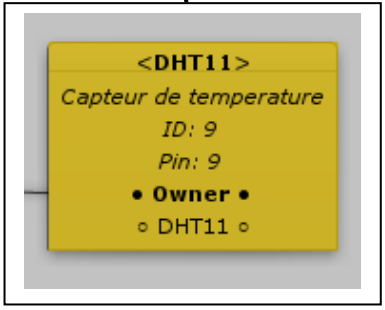


```
public class Temperature : MonoBehaviour
{
    public Text temperature, humidite;
    public GameObject capteur;
    int temp, hum;

    void Start()
    {
        temp = capteur.GetComponent<DHT11>().temperature;
        hum = capteur.GetComponent<DHT11>().humidity;
        temperature.text = "La température est de " + temp + "°C";
        humidite.text = "L'humidité est de " + hum + "%";
    }

    void Update()
    {
        temp = capteur.GetComponent<DHT11>().temperature;
        hum = capteur.GetComponent<DHT11>().humidity;
        temperature.text = "La température est de " + temp + "°C";
        humidite.text = "L'humidité est de " + hum + "%";
    }
}
```

Les éléments entre guillemets sont des chaines de caractères (string) qui s'afficheront telles quelles. C'est pour cela que chaque espace est important.

Le signe **+** permet d'enchainer les différents types de valeurs (String, variable, etc). On dit alors que les trois composantes qui composent le texte « température » ou « humidité » sont concaténées.



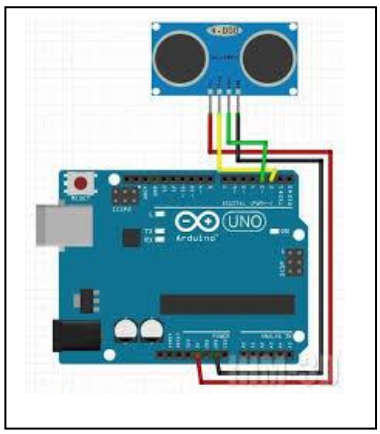


Le blueprint DHT11 n'a qu'une seule entrée digitale car les 2 informations Température et humidité qui proviennent de la sonde sont multiplexées.

Affichage dans l'interface utilisateur

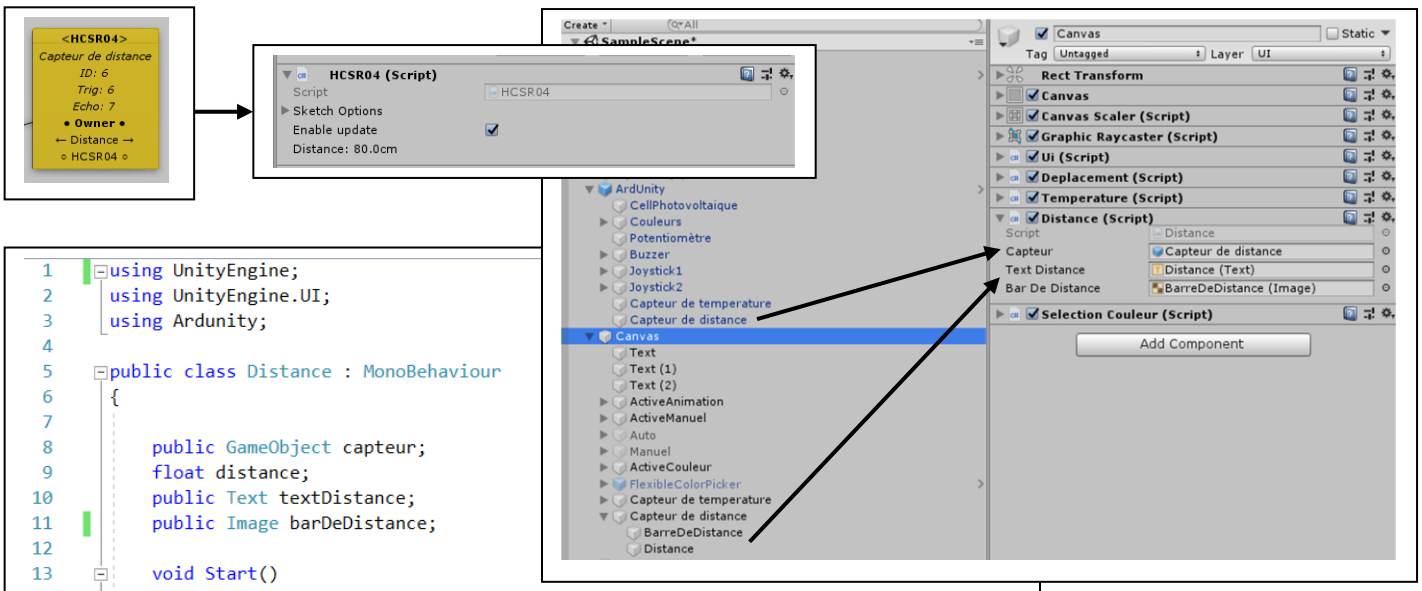
11- Le capteur à ultrason HC-SR04 (Voir vidéo10)

Le blueprint du capteur à ultrason HC-SR04 fait partie de la version payante de l'asset Ardunity. Dans notre exemple le capteur sera fixé sur la main, il détectera la présence d'obstacle ainsi que sa position (80cm maximum). On veut afficher dans l'interface utilisateur une barre de progression allant du blanc au rouge quand l'obstacle se rapproche. Mais on affichera également la valeur numérique de la distance captée en centimètre et au dixième.



Interface utilisateur

Le script ci-dessous permet d'exploiter la valeur récupérer par le blueprint « HCSR04 ». Il permet également de mettre à jour la barre de progression de l'obstacle.




```
1  using UnityEngine;
2  using UnityEngine.UI;
3  using Ardunity;
4
5  public class Distance : MonoBehaviour
6  {
7
8      public GameObject capteur;
9      float distance;
10     public Text textDistance;
11     public Image barDeDistance;
12
13     void Start()
14     {
15         distance = (Mathf.Round((capteur.GetComponent<HCSR04>().distance)*10)/10);
16         textDistance.text = "Obstacle à " + distance + "cm";
17         barDeDistance.fillAmount = 1 - (distance / 80);
18     }
19
20     // Update is called once per frame
21     void Update()
22     {
23         distance = (Mathf.Round((capteur.GetComponent<HCSR04>().distance) * 10) / 10);
24         textDistance.text = "Obstacle à " + distance + "cm";
25         barDeDistance.fillAmount = 1-(distance / 80);
26     }
27 }
```

12- Possibilités pédagogiques, conclusions

Le thème développé ici est un support qui peut servir à de multiples activités autant en atelier sur des classes de SEN ou de Meelec que pour les enseignants de construction. Chacun des points abordés peuvent faire l'objet d'une intégration au système :

- Modification d'un modèle numérique (intégration du capteur HC-SR04 à la main)
- Intégration d'une Led RGB et du buzzer sur la tourelle afin de prévenir de la présence du bras robotisé dans une certaine zone. (Rouge = zone dangereuse, vert = zone sécurisée)
- Intégrer la sonde de température sur les servomoteurs du bras, qui ont tendance à chauffer un peu, avec arrêt automatique en cas de dépassement d'une certaine valeur de température.
- Création d'un pupitre de commande afin d'intégrer les différents joysticks et le potentiomètre pour l'ouverture de la main (Travail avec les enseignants d'art pour le design, l'ergonomie, la signalisation)
- Modification de l'embase pour y accueillir l'alimentation Pc (protection électrique)
- Pilotage de la main en fonction de rotation du bras, avant-bras, poignet. Cette partie plus mathématiques et mécanique peut être l'objet d'une étude cinématique puis d'un calcul vectoriel facilement scriptable. Ce qui pourrait permettre de déplacer la main en fonction du capteur de distance pour automatiser la préhension d'un objet.

En sortant cette fois ci du contexte du bras robotisé, on peut imaginer toute sorte d'application à cette technologie :

- Je compte travailler sur un distributeur de gel hydroalcoolique activable depuis n'importe quel téléphone portable, Androïd pour le moment, qui autorisera un certain nombre de dose par jour afin de limiter le gaspillage. Cette application fonctionnera en Bluetooth avec le blueprint disponible dans la version payante de l'asset Ardunity. La fabrication pourra être fait par les élèves de chaudronnerie, et la partie servomoteur capteur HCSR04, module Bluetooth et Arduino nano fait par les élèves d'électrotechniques
- Une des possibilités de l'asset Ardunity en version payante est de pouvoir exploiter le Wifi. J'aimerai donc créer une maquette d'une maison connectée depuis une interface en réalité virtuelle. L'utilisateur pourrait se retrouver dans une maison virtuelle, réplique de la sienne, et pourrait lui permettre de la piloter mais également d'avoir un retour de son état, résultat des différents capteurs de la maison.

Pour conclure, je pense que ces nouvelles technologies et les moyens relativement simples mis à notre disposition par la communauté méritent d'être explorées et maitrisées. C'est à nous, enseignants, de les démocratiser afin que nos élèves puissent inventer leurs métiers de demain, que nous ne connaissons pas encore, mais qui passeront obligatoirement par ces outils. Une croissance basée sur le carbone est limitée, une croissance basée sur la connaissance et l'information est illimitée.