Institut Universitaire de Technologie,
Aix-Marseille Université

ANNEXES
RAPPORT DE STAGE de fin de deuxième année
Bachelor Universitaire de Technologie
Spécialité Réseaux et Télécommunications
parcours cybersécurité

Développement logiciel embarqué et ressources pédagogiques pour la carte STeaMi

Mattéo Canada

Laboratoire d'Aix-Périmentation et de Bidouille

Responsable entreprise : Sébastien Nedjar

Responsable académique : Arnaud Février

2026

Annexe 1: Présentation du LAB et de la carte STeaMi

Cette annexe complète les sections 2.2 à 2.6.5 du rapport.

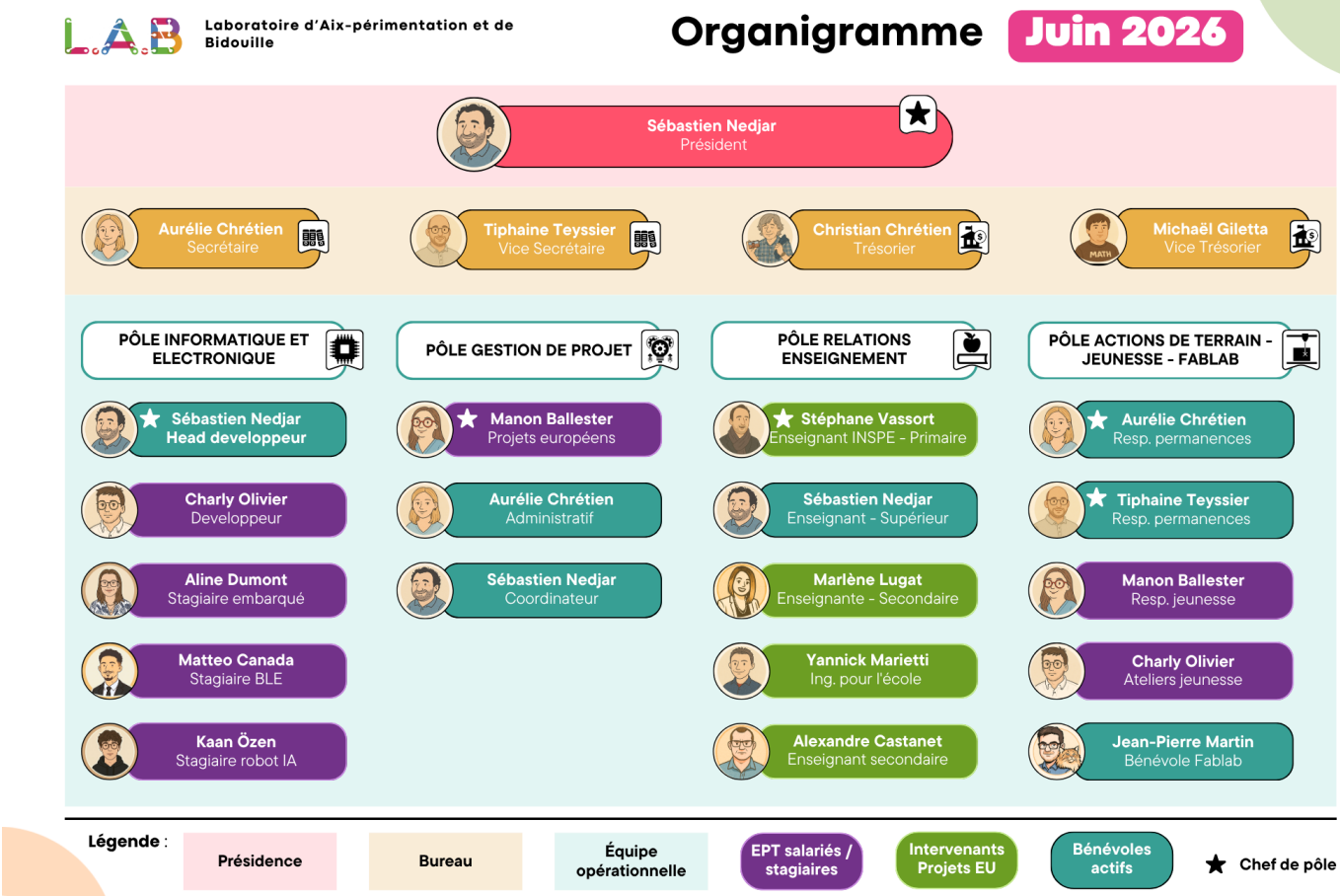


Figure A1.3: Organigramme du L.A.B

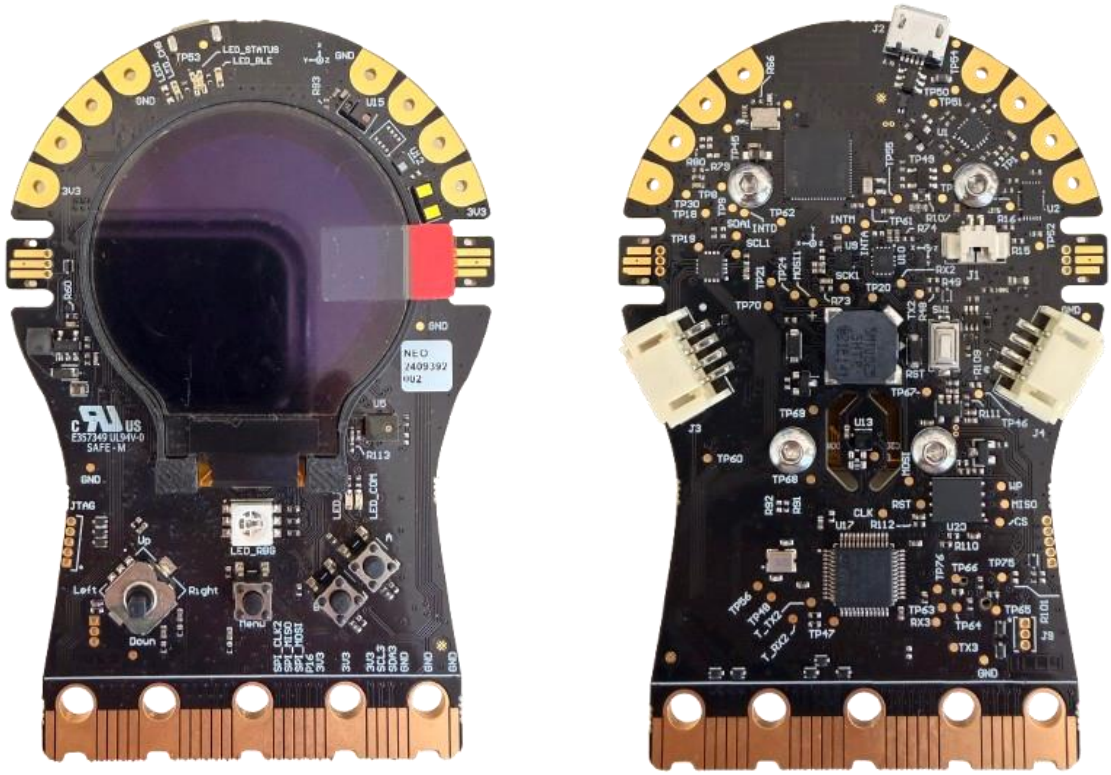


Figure A1.1: STeaMi vu de face et de dos.

CATÉGORIE / SOUS-SYSTÈME	DESCRIPTION & SPÉCIFICATIONS TECHNIQUES	ICÔNE / VISUEL
MICROCONTRÔLEUR (MCU)	**STMicroelectronics STM32WB55RG** Dual-Core (ARM Cortex-M4 @64MHz + Cortex-M0+). Ultra-basse consommation. BLE intégré.	
CONNECTIVITÉ	**Bluetooth Low Energy (BLE 5.2)** Idéal pour l'IoT et expériences en réseau.	
INTERFACE UTILISATEUR (I/O)	**Écran OLED Circulaire 128x128 pixels.** **7 Boutons Poussoirs Tactiles** (D-Pad 4-dirs, 2 Action, 1 Menu). 1 LED RGB. 1 Buzzer.	
CAPTEURS (7 Intégrés)	**ToF Distance** (VL53L1X, 4m). **IMU 6-DoF** (ISM330DLCTR). **Boussole** (LIS2MDL). **Temp/Humidité** (WSEN-HIDS). **Pression** (WSEN-PADS). **Lumière/Gestes** (APDS-9960). **Son** (Microphone).	
STOCKAGE & FLASH	64 Mbit (8 Mo) Winbond Flash memory via liaison série/USB. Stockage de scripts, données et ressources.	
ALIMENTATION	Batterie rechargeable intégrée. Autonomie de terrain (~1 journée). Charge via connecteur USB-MicroB.	
EXTENSIONS	Connecteur Edge compatible micro:bit (robots, shields). Ports rapides Jacdac & Qwiic / STEMMA QT.	
ÉCOSYSTÈME LOGICIEL	Blocs (MakeCode & Vittascience, Collège). MicroPython (Lycée). Arduino C/C++ (Supérieur).	

Figure A1.2: Tableau des spécifications techniques.

Capteur	Modèle	Mesures
Accéléromètre/Gyroscope	ISM330DLC	6 axes + température
Magnétomètre	LIS2MDL	3 axes magnétiques
Distance	VL53L1X	Time-of-Flight, 0-4 m
Pression	WSEN-PADS	26-126 kPa, 24 bits
Humidité/Température	WSEN-HIDS	RH + °C
Lumière/Geste	APDS-9960	ALS, couleur, proximité, geste
Microphone	IMP34DT05	PDM MEMS omnidirectionnel

Figure A1.3: Tableau des spécifications des capteurs internes.

Caractéristique	Détail
Modèle	Winstar WEO128128BWPP3N00001
Contrôleur	SSD1327
Technologie	OLED (blanc)
Résolution	128 x 128 pixels
Interface	SPI1 (interne)
Connecteur	FFC/FPC 24 positions, pas 0.50 mm

Figure A1.4: Tableau des spécifications techniques de l'écran OLED.

Caractéristique	Détail
Modèle	STM32WB55RG
Cœur	ARM Cortex-M4 @ 64 MHz
Flash	1 Mo
RAM	256 Ko SRAM
Radio	BLE 5.2, OpenThread, ZigBee
Antenne	Interne + filtre MLPF-WB55-01E3 (passe-bas 2.45 GHz)
Horloge	Quartz 32 MHz + Quartz 32 KHz (RTC)
Boîtier	VFQFPN68

Figure A1.5: Tableau des spécifications technique de l'architecture de la STeaMi.

Caractéristique	Détail
Stack	Bluetooth 5.2 BLE
Bande	2.4 GHz ISM
Protocoles	BLE, OpenThread, ZigBee
Antenne	Intégrée sur PCB
Filtre RF	MLPF-WB55-01E3

Figure A1.6: Tableau des spécifications techniques de la stack BLE.

Annexe 2: Installation de l'environnement de développement sur Debian

Cette annexe complète la section 3.1.1 du rapport en détaillant les commandes utilisées pour configurer l'environnement de développement de la carte STeaMi sur une machine Debian.

1. Installation de Python et des outils de gestion d'environnement.

```
sudo apt update
sudo apt install python3 python3-pip python3-venv
```

2. Création et activation de l'environnement virtuel.

```
python3 –m venv steami-env
source steami-env/bin/activate
```

L'environnement virtuel isole les dépendances du projet et garantit une configuration reproductible.

3. Installation de mpremote.

```
pip install mpremote
```

Mpremote permet de communiquer avec la carte via USB et d'accéder au REPL.

4. Installation de Thonny.

```
sudo apt install thonny
```

Thonny est utilisé pour le téléversement, l'exécution et le débogage du code MicroPython, avec affichage des sorties en temps réel.

Annexe 3 : Système de localisation par trilatération BLE

Cette annexe complète la section 3.2.3 du rapport.

```python
# ANNEXE – Extraits essentiels du système de localisation par trilatération BLE

# ## VERSION MICROPYTHON ######################################################
# --- Calibration du modèle de propagation -----------------------------------
# RSSI_ref : puissance mesurée à 1 m de chaque balise
# n        : exposant de propagation (path loss)
RSSI_REF = {"Beacon_M1": -75, "Beacon_M2": -75, "Beacon_M3": -80}
PATH_LOSS_N = {"Beacon_M1": 2.5, "Beacon_M2": 2.5, "Beacon_M3": 2.5}
# Coordonnées physiques des balises (cm), origine sur M1
BEACON_POS = {
    "Beacon_M1": (0, 0),
    "Beacon_M2": (420, 0),
    "Beacon_M3": (330, 278),
}

def rssi_to_distance(rssi, beacon_name):
    """Convertit un RSSI en distance (cm) via le modèle log-distance.
    d(m) = 10 ^ ((RSSI_ref - RSSI) / (10 * n))
    """
    ref = RSSI_REF.get(beacon_name, -75)
    n = PATH_LOSS_N.get(beacon_name, 2.5)
    d_m = 10 ** ((ref - rssi) / (10 * n))
    return min(d_m * 100, MAX_DIST_CM)  # m -> cm, puis plafonnement

def trilaterate(distances):
    """Estime une position 2D à partir des distances aux 3 balises.
    Le système des trois équations de cercles est linéarisé par
    soustraction (élimination des termes en x² et y²), puis résolu par
    la règle de Cramer. Renvoie (x, y) en cm, ou None si invalide.
    """
    if len(distances) < 3:
        return None
    names = list(distances.keys())
    x1, y1 = BEACON_POS[names[0]]
    x2, y2 = BEACON_POS[names[1]]
    x3, y3 = BEACON_POS[names[2]]
    r1, r2, r3 = distances[names[0]], distances[names[1]], distances[names[2]]
    A = 2 * (x2 - x1)
    B = 2 * (y2 - y1)
    C = r1**2 - r2**2 - x1**2 + x2**2 - y1**2 + y2**2
    D = 2 * (x3 - x1)
    E = 2 * (y3 - y1)
    F = r1**2 - r3**2 - x1**2 + x3**2 - y1**2 + y3**2
    denom = A * E - B * D
    if abs(denom) < 1e-6:        # balises alignées : pas de solution unique
        return None
    x = (C * E - F * B) / denom
    y = (A * F - D * C) / denom
    return (x, y)

def apply_filter(new_pos):
    """Lissage exponentiel avec seuil minimal de déplacement (anti-gigue)."""
    global filtered_pos
    if filtered_pos is None:
        filtered_pos = new_pos
        return filtered_pos
    fx = ALPHA * new_pos[0] + (1 - ALPHA) * filtered_pos[0]
    fy = ALPHA * new_pos[1] + (1 - ALPHA) * filtered_pos[1]
    if math.sqrt((fx - filtered_pos[0])**2 + (fy - filtered_pos[1])**2) < MIN_MOVE_CM:
        return filtered_pos     # déplacement négligeable : on ne bouge pas
    filtered_pos = (fx, fy)
    return filtered_pos
```

Figure A3.1: Extrait du code MicroPython de trilateration.

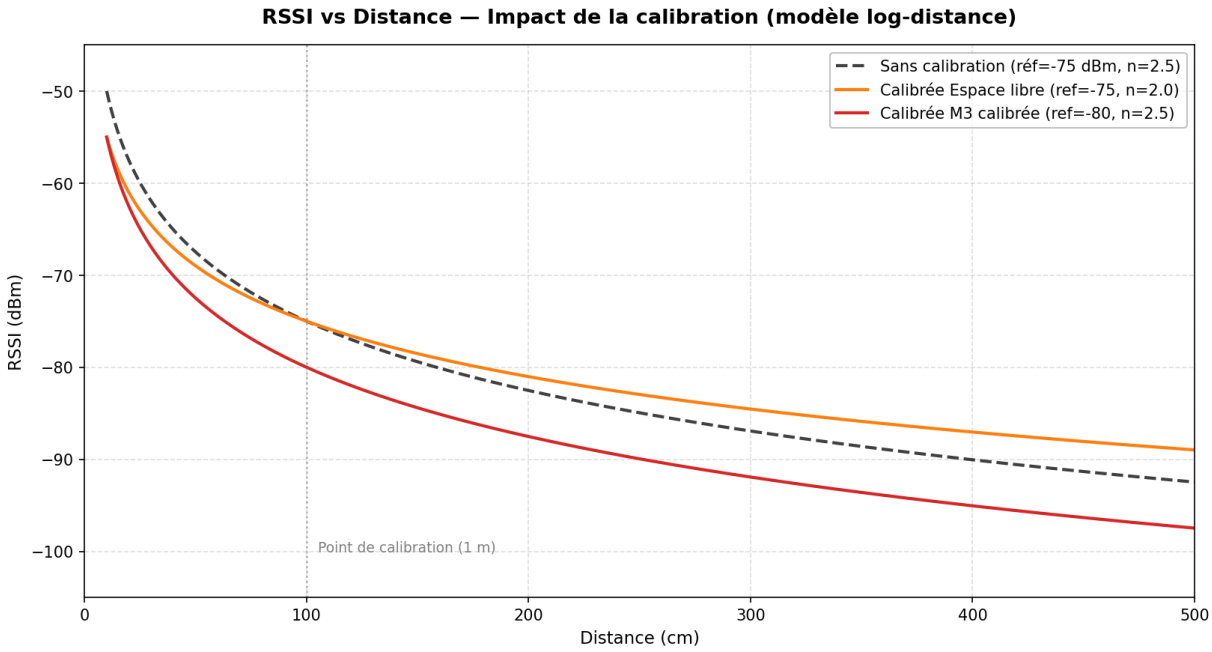


Figure A3.2: Courbe représentative du Path Loss Model selon les milieux

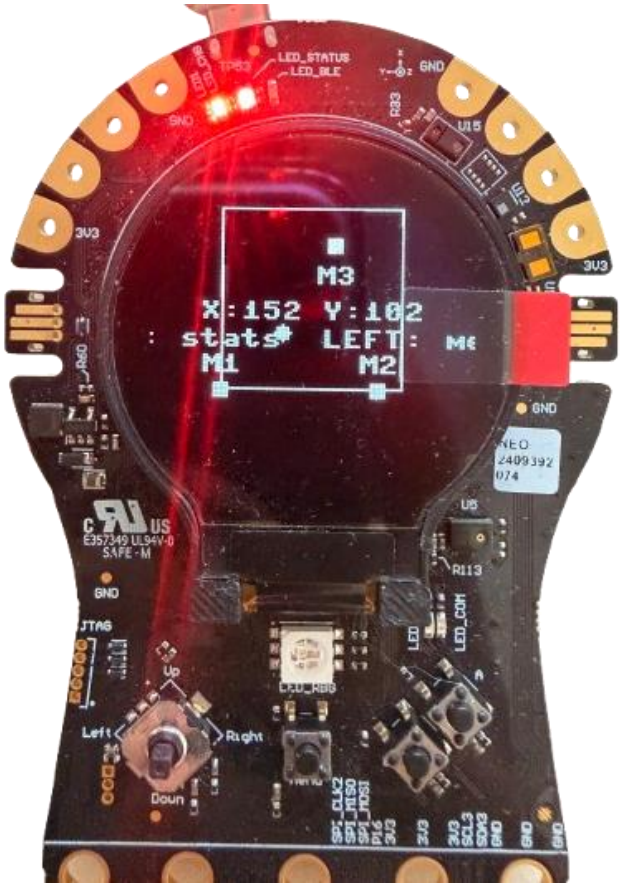


Figure A3.3 : Affichage de la position map.

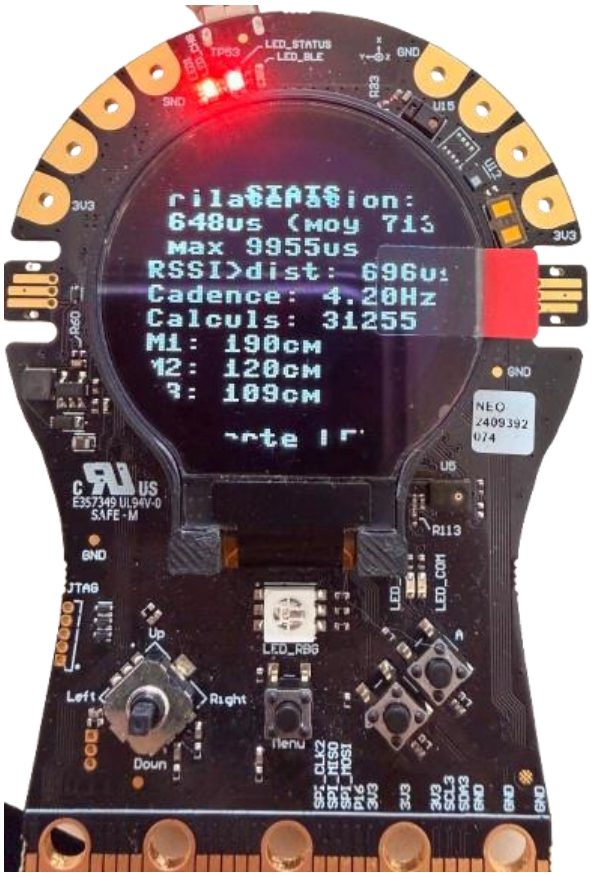


Figure A3.4 : Affichage des performances du script.

Annexe 4: Système de localisation en C++

Cette annexe complète la section 3.3.3 du rapport.

```cpp
// ============================================================================
// ANNEXE – Version C++ / STM32duinoBLE (extraits essentiels)
// ============================================================================
//
// Mêmes principes algorithmiques que la version MicroPython, avec deux
// différences notables liées au langage et à la cible matérielle :
//   - calcul en simple précision (powf, sqrtf) pour exploiter la FPU
//     du Cortex-M4F plutôt que la double précision logicielle ;
//   - validation supplémentaire : une distance jugée non fiable
//     (signal trop faible) invalide l'ensemble du calcul.

// --- Calibration : exposants n affinés expérimentalement par balise ---------
static const float RSSI_REF[BEACON_COUNT]   = {-75.0f, -75.0f, -80.0f};
static const float PATH_LOSS_N[BEACON_COUNT] = {3.99f, 4.32f, 2.66f};
static const float BEACON_X[BEACON_COUNT] = {0.0f, 420.0f, 330.0f};
static const float BEACON_Y[BEACON_COUNT] = {0.0f, 0.0f, 278.0f};

// Convertit un RSSI en distance (cm). Renvoie une valeur négative si le
// signal est trop faible : la mesure est alors rejetée.
// powf() (et non pow()) maintient le calcul en simple précision, ce qui
// correspond à l'unité flottante (FPU) du Cortex-M4F.
float rssiToDistance(int rssi, int idx) {
    float d = powf(10.0f, (RSSI_REF[idx] - rssi) / (10.0f * PATH_LOSS_N[idx]));
    d *= 100.0f;                // mètres -> centimètres
    if (d > MAX_DIST_CM)
        return -1.0f;           // distance non fiable
    return d;
}
// Trilatération : résolution algébrique d'un système linéaire obtenu par
// différence des équations des cercles centrés sur chaque balise.
// Renvoie false si le système est non résoluble ou le résultat aberrant.
bool trilaterate(float r0, float r1, float r2, float& outX, float& outY) {
    float x1 = BEACON_X[0], y1 = BEACON_Y[0];
    float x2 = BEACON_X[1], y2 = BEACON_Y[1];
    float x3 = BEACON_X[2], y3 = BEACON_Y[2];
    float A = 2.0f * (x2 - x1);
    float B = 2.0f * (y2 - y1);
    float C = r0*r0 - r1*r1 - x1*x1 + x2*x2 - y1*y1 + y2*y2;
    float D = 2.0f * (x3 - x1);
    float E = 2.0f * (y3 - y1);
    float F = r0*r0 - r2*r2 - x1*x1 + x3*x3 - y1*y1 + y3*y3;
    float denom = A * E - B * D;
    if (fabsf(denom) < 1e-6f)    // balises alignées : pas de solution unique
        return false;
    outX = (C * E - F * B) / denom;   // règle de Cramer
    outY = (A * F - D * C) / denom;
    // Rejet si la position calculée sort de la zone plausible
    float dx = outX - CENTROID_X;
    float dy = outY - CENTROID_Y;
    if (sqrtf(dx * dx + dy * dy) > MAX_VALID_CM)
        return false;
    return true;
}
// Filtre exponentiel sur la position (ALPHA = 0.15 : lissage fort) avec
// seuil minimal de déplacement pour supprimer le tremblement résiduel.
void applyFilter(float newX, float newY) {
    if (filteredX < 0.0f) {      // première position : initialisation directe
        filteredX = newX;
        filteredY = newY;
        return;
    }
    float fx = ALPHA * newX + (1.0f - ALPHA) * filteredX;
    float fy = ALPHA * newY + (1.0f - ALPHA) * filteredY;
    float dx = fx - filteredX;
    float dy = fy - filteredY;
    if (sqrtf(dx * dx + dy * dy) < MIN_MOVE_CM)
        return;                  // déplacement négligeable : ignoré
    filteredX = fx;
    filteredY = fy;
}
```

Figure A4.1: Extrait du code C++ du code de trilateration.

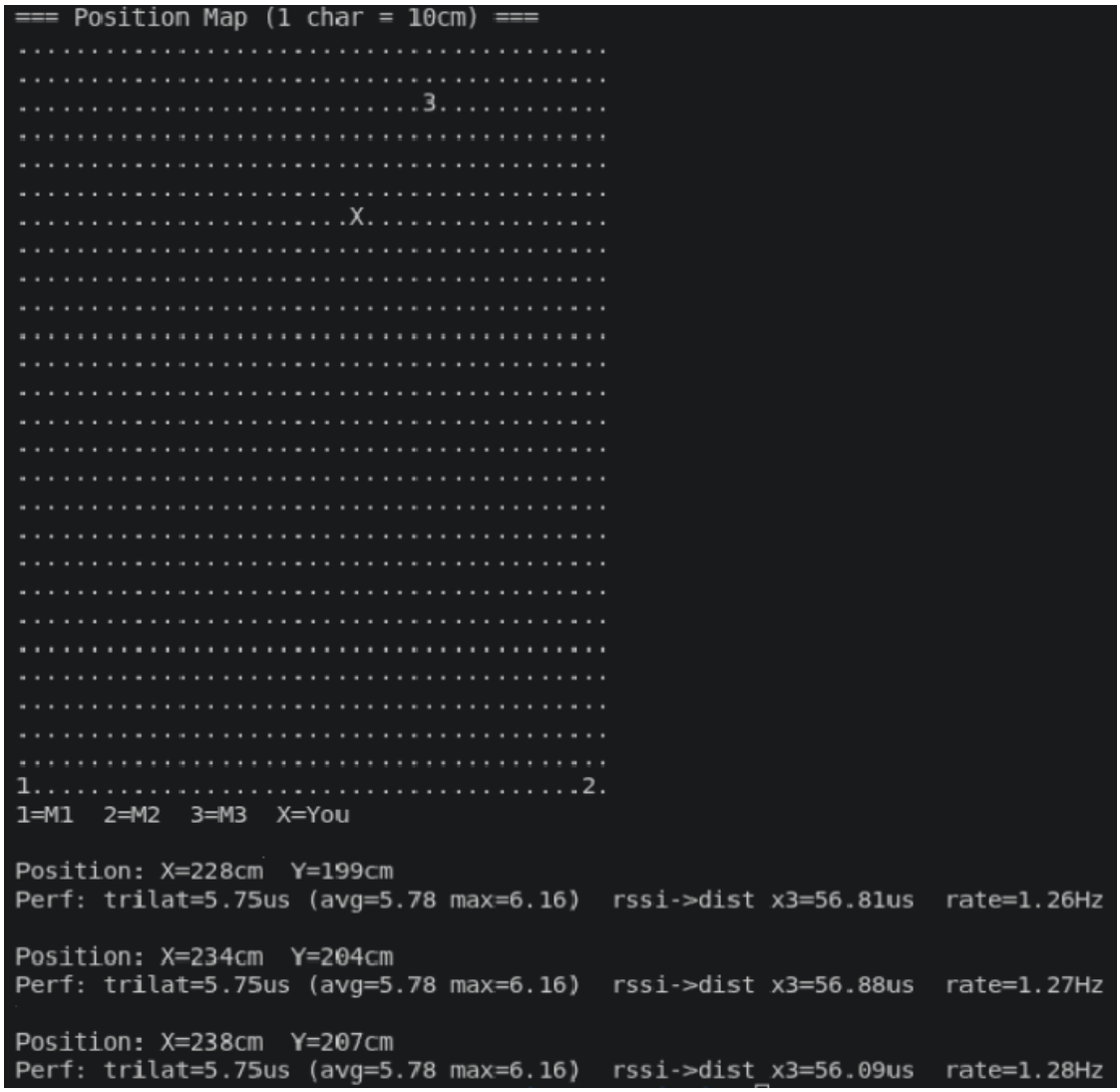


Figure A4.2: Position Map avec performances du programme C++.

CRITÈRE DE PERFORMANCE TECHNIQUE	MICROPYTHON	C++ (ARDUINO IDE)	
EXÉCUTION DU CODE	**Interprété** (Traduction en direct par un interpréteur embarqué; surcharge pour les instructions brutes).	**Nativement Compilé** (Le code machine s'exécute directement; vitesse d'exécution maximale).	VS
VITESSE DE TRAITEMENT Raw	**Modérée** (Surcharge de l'interpréteur; vérification des types de données, etc.; adapté au journal de données données de capteurs).	**Excellente** (Pas d'interpréteur au runtime; code bas niveau et efficace; réponse en temps réel).	CPU
TEMPS D'INITIALISATION	**Plus Rapide** (Le flashage du script prend quelques secondes; cycles de prototypage agiles).	**Plus Lent** (Nécessite une compilation complète et un téléversement à chaque fois; processus itératif plus lent).	VS
CONSOMMATION MÉMOIRE	**Significative** (L'interpréteur utilise une RAM fixe; moins d'espace pour les grands tableaux de données).	**Optimisée** (Utilisation minimale de la RAM; allocation manuelle précise de la mémoire pour le code utilisateur).	RAM VS
EMPREINTE FLASH	**Grande** (Le firmware inclut l'interpréteur complet; limite la complexité du script sur une mémoire flash limitée).	**Petite** (Le code utilisateur est compacté; laisse plus de place pour des algorithmes complexes).	FLASH
ACCÈS REGISTRES BAS NIVEAU	**Limité** (Accès via des bibliothèques Python prédéfinies; l'accès direct aux registres est plus difficile).	**Complet** (Permet une configuration précise des registres du STM32WB55 via I2C/SPI directs).	
OPTIMISATION DE LA CONSOMMATION	**Bonne** (Gestion implicite de l'alimentation; contrôlée et simplifiée via les couches logicielles).	**Critique** (Permet un contrôle granulaire sur les modes de veille de chaque cœur (M4 vs. M0+); économie d'énergie complexe).	

Source des données techniques : Analyses de performance STM32WB55 et tests sur la STeaMi (steami.cc)

Figure A4.3: Tableau démontrant les différences de performances entre MicroPython et C++

Annexe 5 : Ressources pédagogiques et événements

Cette annexe complète les sections 4.2, 4.3 et 4.4 du rapport.

Exemples de fiches pédagogiques crées :

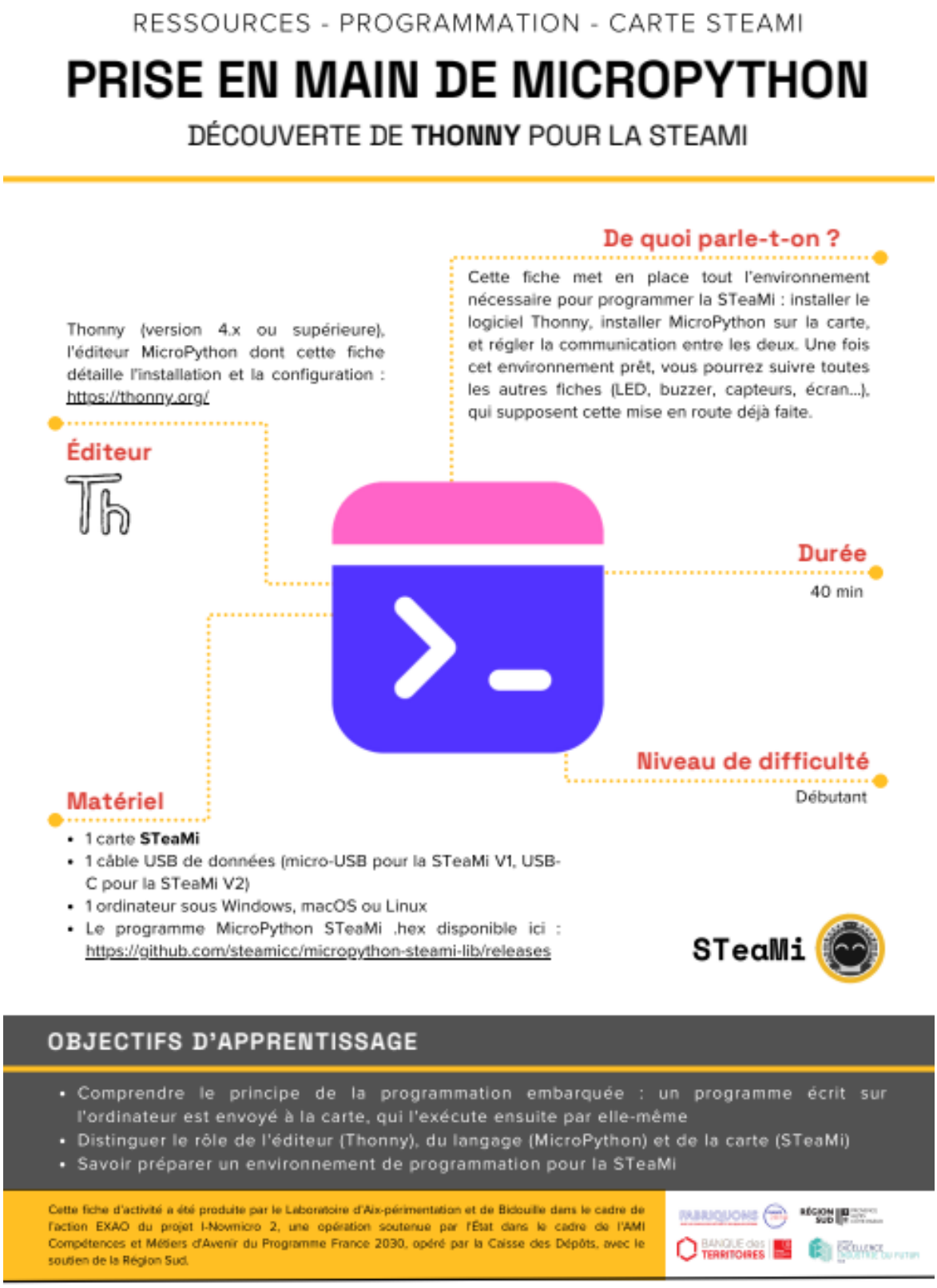


Figure A5.1: Fiche introduction à l'IDE Thonny.

Lien vers la fiche complète : https://wiki.labaixbidouille.com/ressources/inovmicro-exao/t03-decouverte-thonny

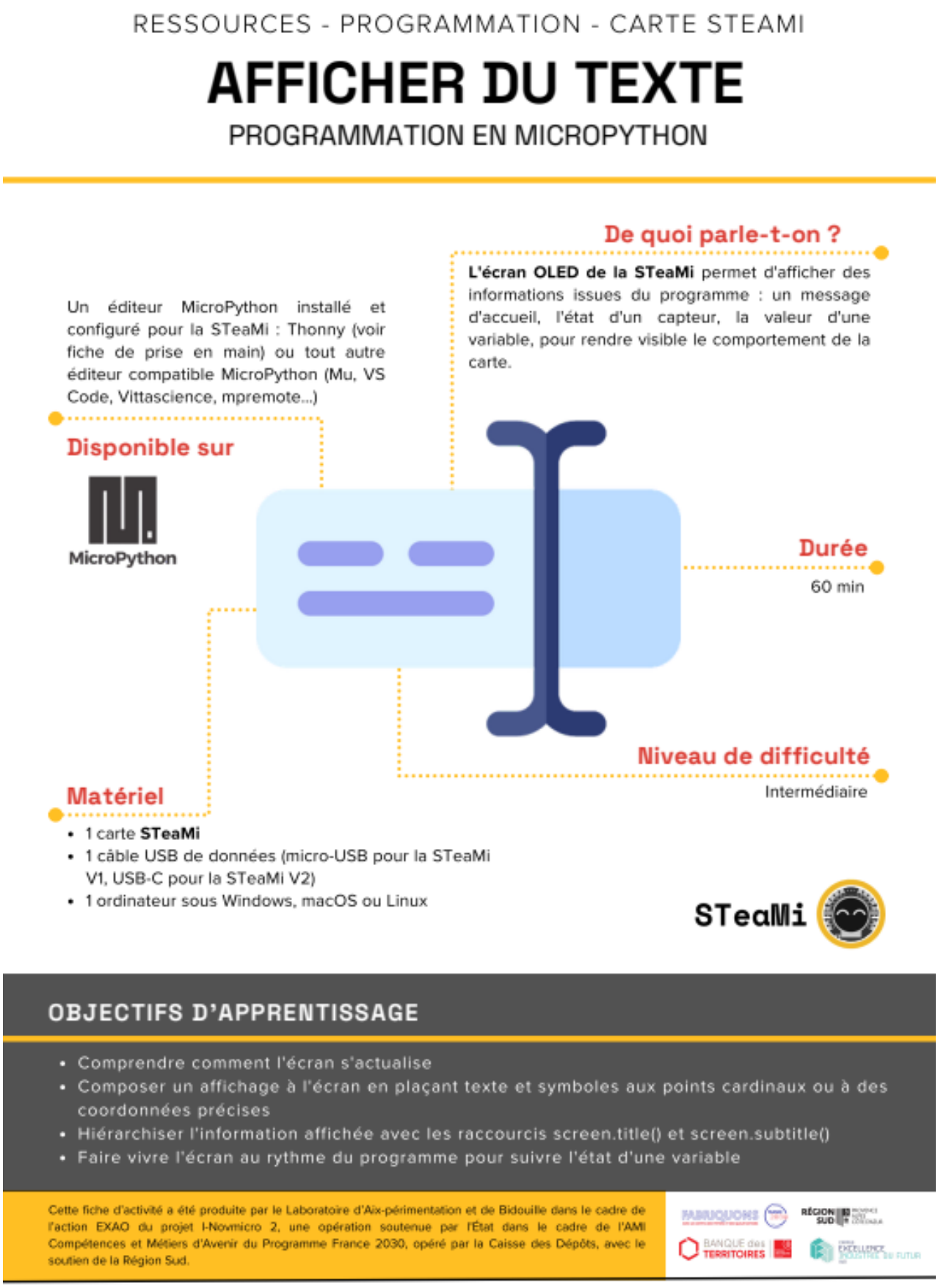


Figure A5.2: Fiche pédagogique prise en main de l'écran SSD1327.

Lien vers la fiche complète : https://wiki.labaixbidouille.com/ressources/inovmicro-exao/i10-texte-oled
Cette fiche, de niveau intermédiaire, accompagne les enseignants dans la prise en main de l'écran OLED SSD1327 de la STeaMi en MicroPython. Elle explique comment actualiser l'affichage, positionner du texte et des symboles à des coordonnées précises ou aux points cardinaux, et hiérarchiser l'information à l'aide des raccourcis screen.title() et screen.subtitle(). L'objectif est de rendre visible le comportement du programme en vérifiant l'état d'un capteur, la valeur d'une variable, le message d'accueil directement sur la carte, sans connexion permanente à un ordinateur. Comme l'ensemble des fiches produites, elle suit le gabarit éditorial du wiki (durée, matériel, objectifs d'apprentissage, licence CC BY-SA 4.0) afin d'être directement réutilisable en classe.

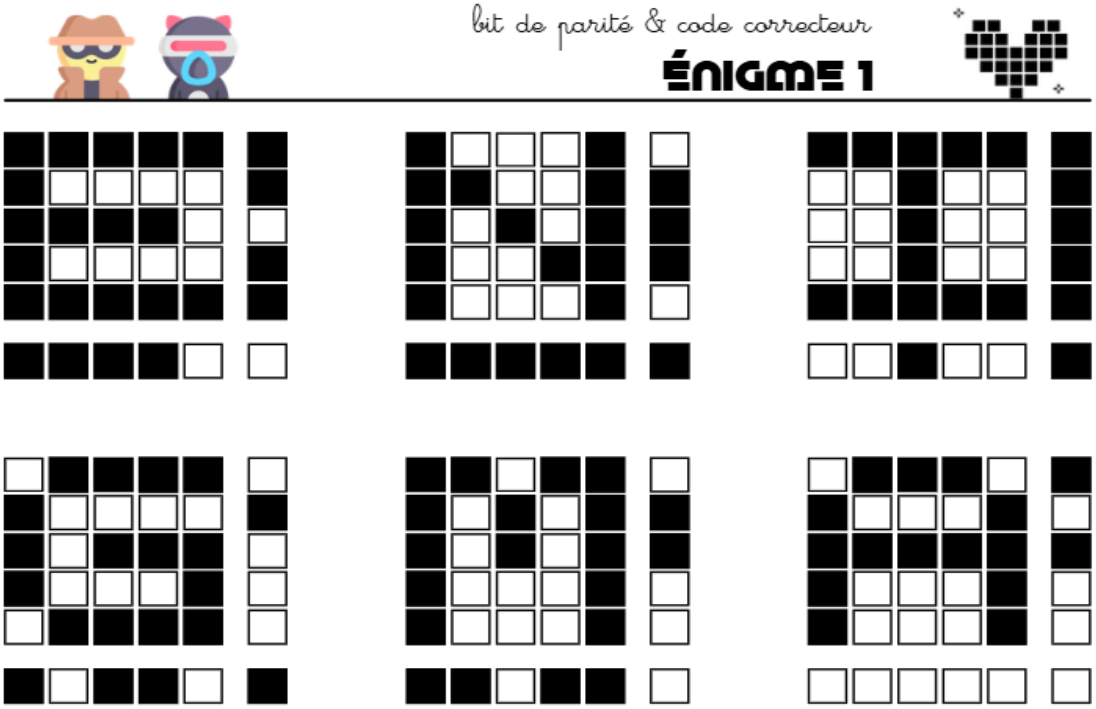


Figure A5.3 : Exercice Bit de parité proposé aux CE1/CE2.



Figure A5.4 : Matériel utilisé lors de l'exercice Dr Hanoi.

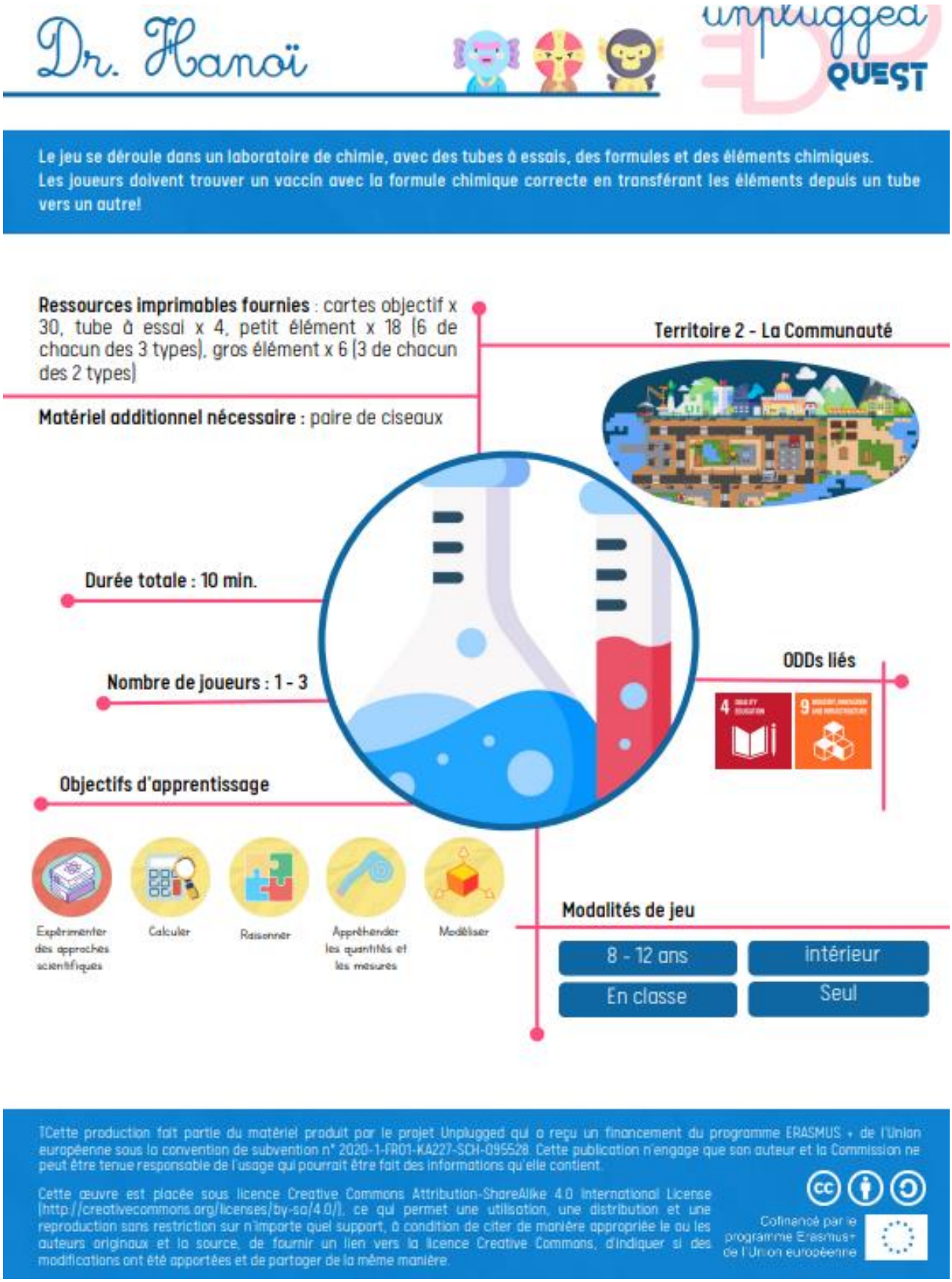


Figure A5.5 : Page de garde la fiche Dr Hanoi.

Ci-dessous le lien pour consulter l'exercice complet :

https://labaixbidouille.wixsite.com/unplugged-metaverse/fr/hanoi

Stand Journée ITER Robots.



Figure A5.6 : Photo du stand en pleine journée.

Les élèves présents étaient très heureux de pouvoir tester les STeaMi via des jeux comme le Maze Game, Tétris (version plus complexe) et même Snake.