

BTS CIEL
Épreuve E6.2 - Session 2026
Dossier de projet

Projet : **ESTOM**
Espace de **S**ensibilisation au
Tri des **O**rdures **M**énagères

“Course du Tri”



Étudiants :

- Mohamed CHAIDA-BOURDOT (IR 1)
- Soheybe ABDAT (IR 2)
- Fatma HACENE (IR 3)
- Thomas BLANC (ER 1)
- Kevin PIRES SALVADOR (ER 2)

Sommaire

Partie Commune.....	5
I - Expression du besoin.....	6
I.1 - Contexte.....	6
II.2 - Problématique.....	7
II - Répartition des tâches.....	7
III - Acteurs et Utilisations Générales.....	9
IV - Compte rendu des réunions.....	11
IV.1 - Réunion du 06/01/2026.....	11
IV.2 - Réunion du 03/02/2026.....	13
IV.3 - Réunion du 01/04/2026.....	14
IV.3 - Réunion du 30/05/2026.....	18
V - Fiches d'audit de sécurité et recettes.....	21
V.1 - Fiche d'audit de sécurité.....	21
V.2 - Fiches de recette.....	22
Partie Personnelle IR1: BOM/Bus.....	32
I - Introduction personnelle.....	33
II - Objectifs et Cahier des Charges.....	33
III - Mon Rôle et Mon Environnement.....	34
IV - Mes Réalisations Techniques.....	34
V - Résolution de Problèmes.....	35
VI - Travail d'Équipe et Intégration.....	36
VII - Les Prochaines Étapes.....	36
VIII - Bilan de la Phase Précédente.....	37
IX. Évolution et Sécurisation de la Communication.....	38
IX.1 - Version 1.0: Le prototype initial.....	38
IX.2 - Version 1.6: La standardisation et la fiabilisation par le texte.....	38
IX.3 - Version 2.0: L'aboutissement et la distinction des rôles.....	39
X - Refonte de l'Architecture et Nouvelles Fonctionnalités.....	39
X.1 - La Modularité par la Programmation Orientée Objet (POO).....	40
X.2 - Stabilisation de l'affichage (Migration vers NeoPixel).....	41
X.3 - Mécanique de jeu: Le système "Anti-Spam" (Cooldown).....	41
X.4 - Surveillance matérielle: La jauge de batterie intelligente.....	42
XI - Tests Grandeur Nature & Débogage.....	43
XI.1 - Le bug du "Blackout" visuel (Problème de jauge).....	43
XI.2 - Le piège matériel de l'ESP32-S3 (Conflit de port série USB CDC).....	44
Partie Personnelle IR2: PAV/Abribus.....	46

I - Introduction personnelle.....	47
II - Objectifs et Exigences.....	48
II.1 - Objectifs fonctionnels.....	48
II.2 - Exigences techniques.....	49
III - Planification.....	50
II.1 - Outils logiciels.....	50
II.2 - Tâches planifiées.....	53
II.3 - Horaires de travail.....	53
IV - Présentation de ma partie.....	54
II.1 - Généralités.....	54
IV.1 - Architecture technique.....	56
IV.3 - Sécurité et fiabilité.....	57
V - Déroulement du travail.....	59
V.0 - Partie physique: l'infrarouge.....	59
V.1 - Le travail réalisé.....	62
VI - Conclusion.....	65
Partie Personnelle IR3: Supervision.....	66
I - Introduction personnelle.....	67
II - Objectifs et Exigences.....	67
II.1 - Objectifs fonctionnels.....	67
III - Planification.....	70
III.1 - Outils et méthodes.....	70
III.2 - Tâches planifiées.....	71
IV - Présentation de ma partie.....	72
IV.1 - Généralités.....	72
IV.2 - Architecture technique.....	72
IV.3 - Sécurité et fiabilité.....	72
V - Déroulement du travail.....	73
V.0 - Partie physique: le Wi-Fi.....	73
V.1 - Partie 1: Mise en place.....	74
V.2 - Partie 2: Le serveur TCP et le programme Qt/C++.....	78
V.3 - Partie 3: Le serveur Web.....	85
V.4 - Quelques changements.....	88
VI - Conclusion.....	90
Partie Personnelle ER1: BOM/Bus.....	91
I - Introduction personnelle.....	92
II - Organisation du projet – Diagramme de Gantt prévisionnel et réel.....	93
III - Présentation du Travail.....	94
III.2 - Mise en place de l'environnement de travail.....	97
III.3 - Réception et analyse des signaux infrarouges.....	101
III.4 - Pilotage de la matrice de LED.....	105

III.5 - Test Anti-choc et Rebond.....	107
III.6 - Brasage et câblage de la nouvelle carte.....	109
III.7 Instructions de Câblage et d'Assemblage des Composants.....	113
III.8 Mise en Œuvre et Configuration de la Carte.....	116
III.9 - Le choix de la batterie et les problèmes rencontrés.....	118
III.10 Guide de Dépannage et Diagnostic (Troubleshooting).....	119
III.11 - Fiche de recette.....	121
Partie Personnelle ER2: PAV/Abribus.....	122
I - Présentation générale.....	123
I.1 - Objectif technique.....	123
II - Organisation du travail.....	124
II.1 - Diagramme de Gantt prévisionnel.....	124
II.2 - Observation et compréhension de la carte.....	124
II.3 - Partie teste des LED avec Arduino IDE.....	125
II.4 - Code pour le DIP SWITCH.....	126
II.5 - Présentation de l'équipement.....	128
Annexe.....	130
ANNEXE IR1.....	131
Annexe 1: Code estomBOM (estomBOM.ino):.....	131
Annexe 2: Photo du montage sur le test des DIP SWITCH.....	133
Annexe 3: Cablage sur breadboard.....	134
ANNEXE IR2.....	135
ANNEXE IR3.....	144
Annexe 1: Serveur TCP - CDatabase.cpp.....	144
Annexe 2: Serveur TCP - CConnectChecker.cpp.....	149
Annexe 3: Serveur TCP - CTcpSender.cpp.....	152
Annexe 4: Serveur TCP - CTcpServer.cpp.....	153
Annexe 5: Serveur TCP - CApp.cpp.....	156
Annexe 6: Serveur Web - index.php.....	159
Annexe 7: Serveur Web - gestion-de-la-partie.php.....	160
Annexe 7: Serveur Web - parametre-du-jeu.php.....	168

Partie Commune

I - Expression du besoin

I.1 - Contexte

Luberon Monts de Vaucluse agglomération (LMV) a commandé un projet auprès des étudiants en BTS CIEL l'année dernière (2025), qui consiste à sensibiliser les gens sur l'importance de trier et de la récolte des déchets à l'aide d'un jeu ludique utilisant des Bennes à Ordures Ménagères (BOM) et des Points d'Apport Volontaires (PAV), simulant la récolte des déchets.

Il y a 4 BOMs et 12 PAV, 1 bus et 3 abribus par couleur. Les couleurs sont BLEU pour papier, VERT pour le verre, JAUNE pour le tri et GRIS FONCÉ pour les ordures ménagères mais aussi la présence d'un C'Mon Bus et 3 abribus, donc 5 joueurs au total avec 5 véhicules et 15 PAV/Abribus.

Pour cette édition 2026, notre mission est de finaliser le prototype afin de le rendre pleinement opérationnel. L'enjeu n'est plus seulement de faire fonctionner le matériel, mais de l'optimiser via des batteries de divers tests tant en électronique qu'en informatique.

II.2 - Problématique

Comment concevoir et programmer un jeu interactif pédagogique, capable de simuler une collecte de déchets par le biais de véhicules robotisés (BOM et Bus) interagissant avec des points d'apport volontaire (PAV) et un système de supervision centralisé ?

II - Répartition des tâches

IR1: CHAIDA-BOURDOT Mohamed (BOM/BUS)

- Capteurs infrarouge
- Capteurs de choc
- Signalement au Superviseur de la présence d'un PAV/ABRIBUS
- Changement de couleur et type grâce au DIP Switch

IR2: ABDAT Soheybe (PAV/ABRIBUS)

- Système infrarouge
- Complicité avec le BOM
- Signal au Superviseur le niveau de batterie restant
- Changement de couleur et type grace au DIP Switch

IR3: HACENE Fatma (SUPERVISION)

- Réception et traitement des données des PAV/ABRIBUS et des BOM/BUS
- Envoie des ordres au PAV/ABRIBUS et BOM/BUS
- Vue en temps réel sur la partie et sur l'état de chaque PAV/ABRIBUS et BOM/BUS
- Affichage des scores à la fin de la partie
- Gère la base de données

ER1: BLANC Thomas (BOM/BUS)

- Configuration de l'environnement de développement (ATtiny85 et ESP32-S3).
- Décodage et traitement des trames infrarouges (Protocole NEC).
- Conception et programmation des animations visuelles sur la matrice de LED (Interface Homme-Machine).
- Gestion de la sécurité matérielle (Détection des chocs et algorithme d'anti-rebond).

ER2: PIRES SALVADOR Kevin (PAV/ABRIBUS)

- Programmation de la matrice de LED 4x4 sur Arduino
- Analyse Routage carte et schéma de câblage
- Test de couleur pour les LED
- Configuration de L'ESP32-S3

III - Acteurs et Utilisations Générales

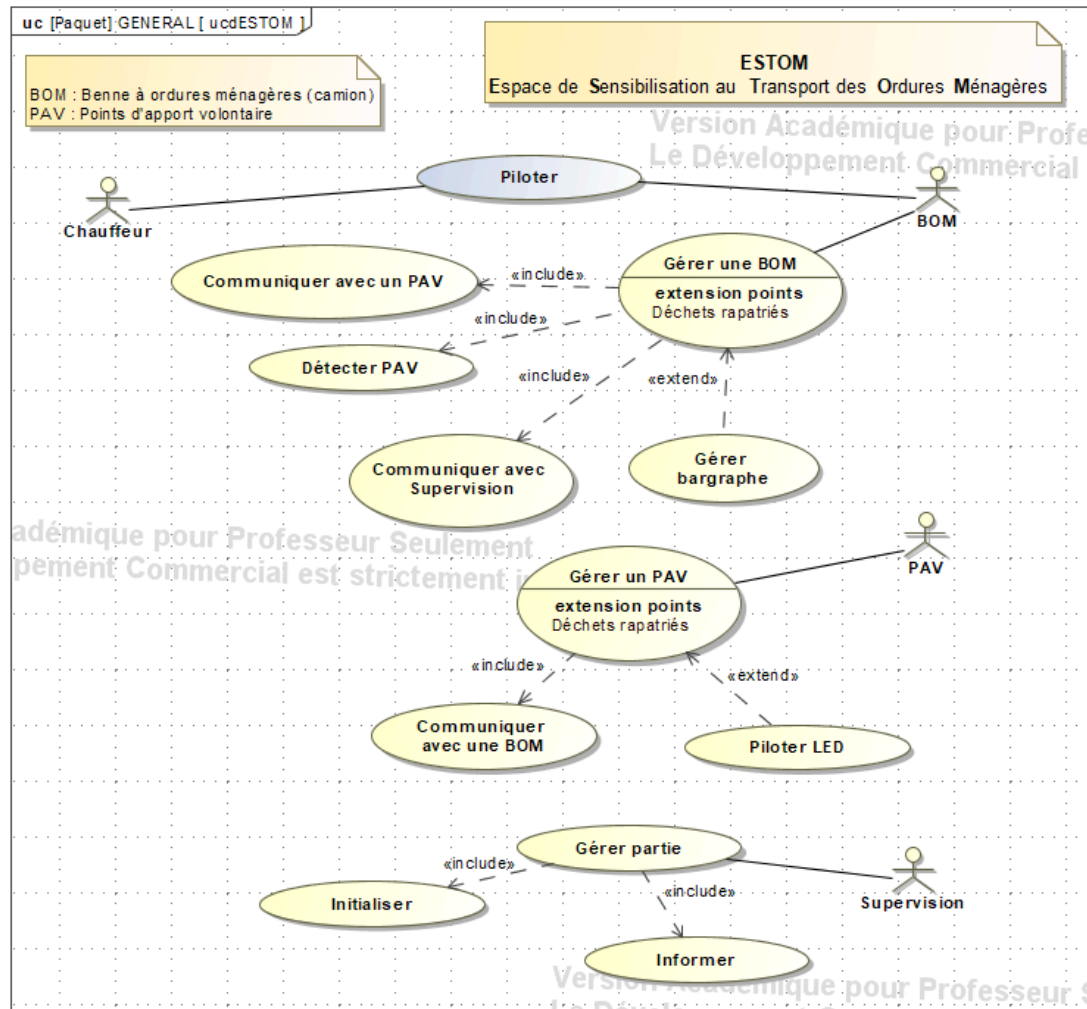


Diagramme des cas d'utilisation ucdESTOM

Piloter un BOM (ou bus) :

- La BOM se pilote via un smartphone.
- Le châssis 4x4 assure une bonne stabilité.
- Le joueur reçoit des indications grâce aux LED et aux retours du système.
- Le pilotage reste actif pendant toute la partie.

Gérer un BOM (ou bus) :

- La BOM détecte les PAV grâce à l'infrarouge.
- Elle identifie le type du PAV et vérifie la compatibilité du déchet.
- Si le type correspond, elle effectue le transfert.
- Elle met à jour son niveau de remplissage.

- Elle envoie ses informations à la SuperVision.
- Elle signale immédiatement toute collision.
- Une barre lumineuse indique la progression du joueur.

Gérer un PAV (ou abribus) :

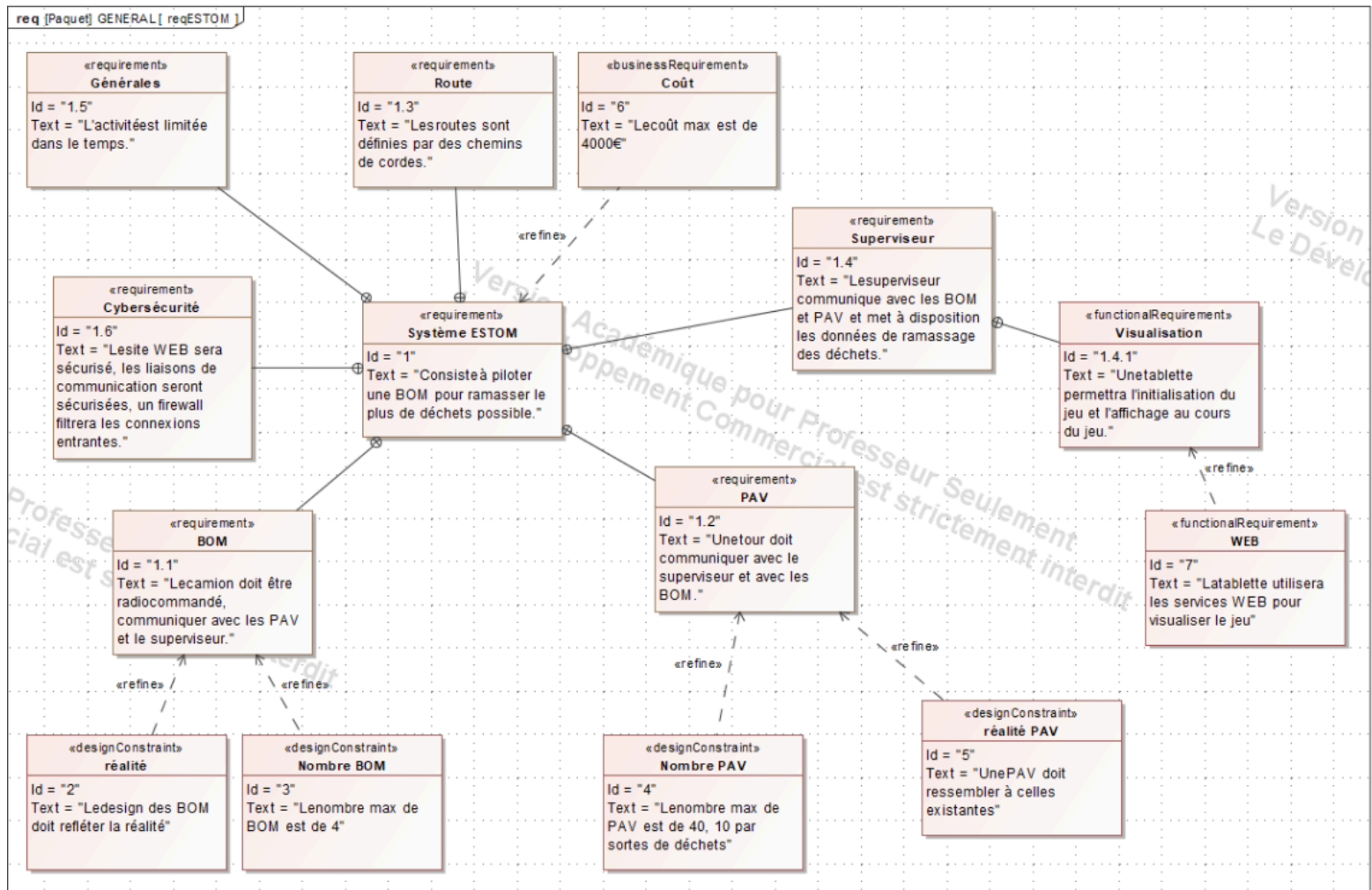
- Le PAV simule un vidage progressif.
- Lorsqu'une BOM approche, il transmet son type et son niveau.
- Il attend une commande de la BOM pour valider un transfert.
- Si le transfert est accepté, il met à jour son état.
- Il respecte un temps d'attente avant d'être vide.
- Si le BOM bouge trop tôt, le cycle recommence.

Gérer la Supervision :

- La Supervision initialise toutes les BOM/bus et tous les PAV/abribus.
- Elle suit en temps réel leurs actions et leurs niveaux.
- Elle reçoit les trames envoyées par les modules.
- Elle gère les règles du jeu et les scores.
- Elle affiche l'état du terrain sur une interface web.
- À la fin, elle envoie un ordre d'arrêt à tous les modules.
- Les résultats sont disponibles immédiatement.

Rôle du Joueur :

- Le joueur pilote la BOM.
- Il doit se positionner correctement devant les PAV.
- Il doit respecter les temps d'attente.
- Son objectif est de trier correctement et d'éviter les collisions



IV - Compte rendu des réunions

IV.1 - Réunion du 06/01/2026

Participants : HACENE Fatma, CHAIDA-BOURDOT Mohamed, ABDAT Soheybe

Absents : Aucun

Ordre du jour

Objectif : Faire en sorte que le PAV envoie des données au BOM

1. Code Arduino pour le BOM et le PAV

Discussions : Faire en sorte que le microcontrôleur du BOM et du PAV puisse se connecter en Wi-Fi et en TCP.

Conclusion et actions :

- Installation du logiciel Arduino
- Installation des librairies *WiFiEsp* et *IRMP*
- Installation du *Board Manager* Arduino AVR Boards
-

2. Type de communication

Comme nous devons poursuivre le travail des 2^e année de l'année dernière, c'est donc par Wi-Fi que nous communiquerons nos données entre le PAV, le BOM et bien sûr le superviseur.

Conclusion et actions :

- Nous avons pu constater que la connexion Wi-Fi fonctionne correctement
- Les codes Arduino sont réalisés en langage C

3. Type de capteur

Nous utilisons un capteur infrarouge dans ce projet. Grâce à ce capteur, qui détecte l'emplacement de la BOM, les données pourront circuler.

Conclusion et actions :

- Le code est terminé
- Il faut que les ER puissent nous fournir un résultat pour permettre l'affichage des couleurs

4. Configuration et paramétrage

(Point à demander à Monsieur Antoine : les PAV s'allument-ils tous ou seulement certains selon le nombre de joueurs ?)

Conclusions et actions :

- Les parties durent 3 minutes
- Le PAV et le BOM envoient un signal à la supervision pour indiquer qu'ils sont prêts
- Paramétrage du BOM et du PAV via le site internet dans les paramètres

Conclusions et actions

- Les parties durent **3 minutes**.
- Le PAV et le BOM envoient un signal à la supervision pour indiquer qu'ils sont **prêts**.
- Le paramétrage du BOM et du PAV se fait via le **site internet**, dans les paramètres.

IV.2 - Réunion du 03/02/2026

Participants : HACENE Fatma, CHAIDA-BOURDOT Mohamed, ABDAT Soheybe, PIRES SALVADOR Kevin & BLANC Thomas

Absents : Aucun

Ordre du jour

Objectif : Se présenter et discuter avec la cliente sur le projet

Nous avons eu une réunion avec Mme MOY, la cliente du projet ESTOM. Elle est chargée de Prévention Déchet chez LMV. On a discuté à propos de quelques attentes mais aussi de quelques questions.

Elle nous a confirmé une date de démonstration : **le 21 mars à Cavaillon, à 10h, à la MJC** mais la date a été reportée au **1er avril, au siège de LMV Agglomération, à Cavaillon.**

IV.3 - Réunion du 01/04/2026

Participants : HACENE Fatma, CHAIDA-BOURDOT Mohamed & ABDAT Soheybe

Absents : PIRES SALVADOR Kevin & BLANC Thomas

Ordre du jour

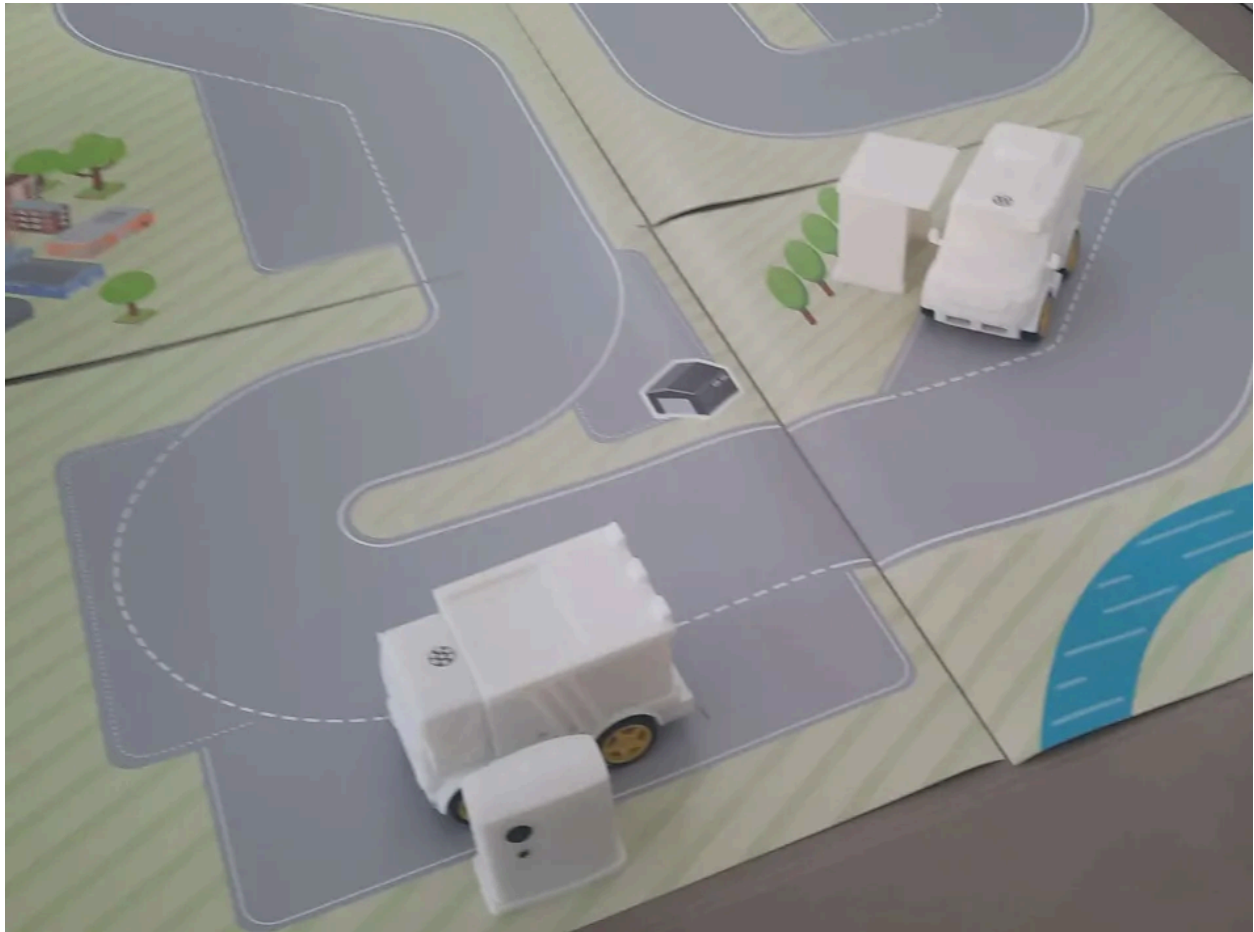
Objectif : Démonstration du prototype du jeu en intérieur

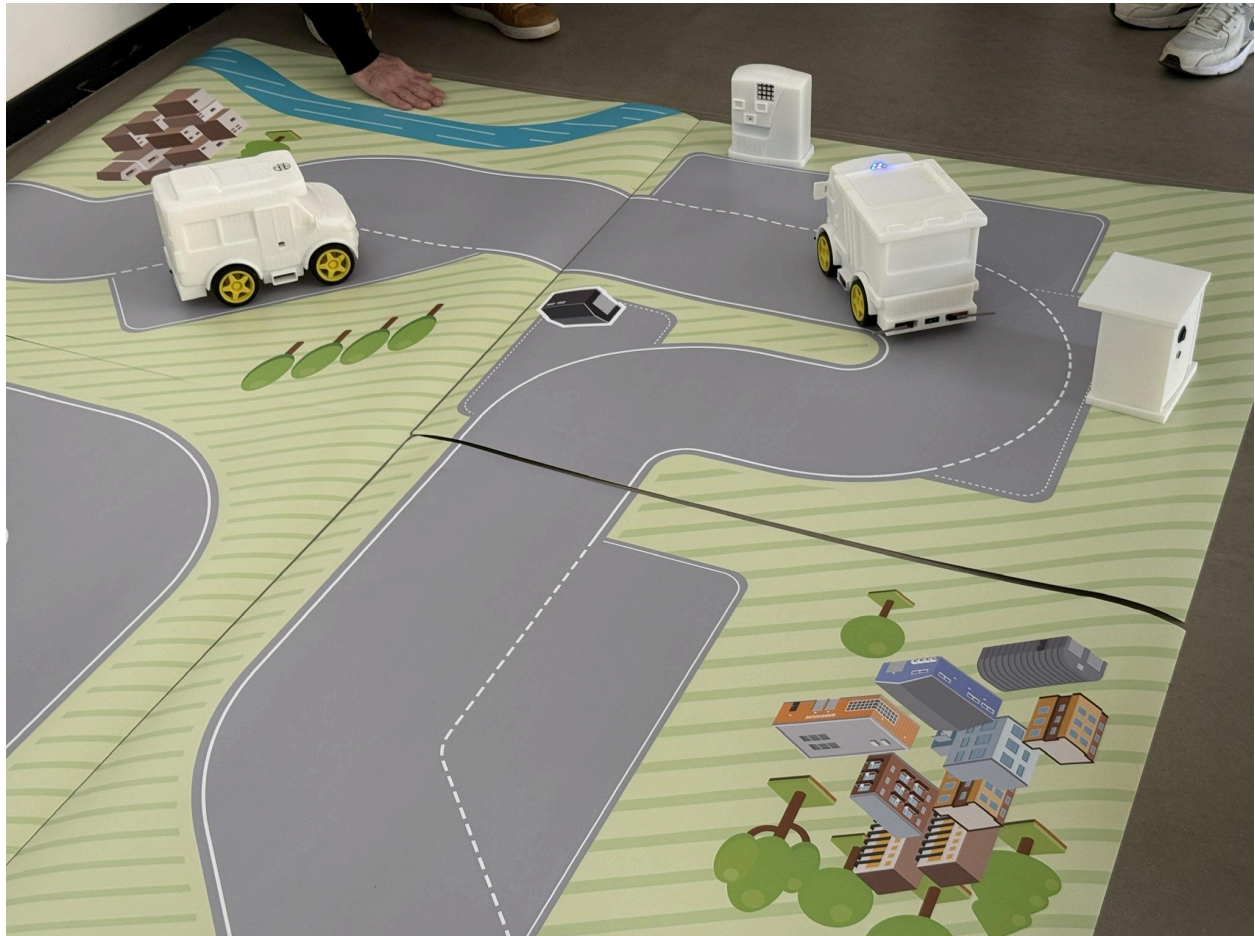
Cette réunion a eu lieu directement sur le site de LMV, à Cavaillon, afin de tester le PAV dans un environnement réel, avec le vrai BOM et les conditions extérieures (lumière du soleil, distance, obstacles, etc.).

L'objectif était de vérifier le comportement du système en dehors de la salle de TP et de confirmer que l'infrarouge, la communication réseau et les différents états du PAV fonctionnaient correctement sur le terrain.

Sur place, nous avons pu observer comment le BOM réagissait en passant devant le PAV. Les tests ont montré que la trame infrarouge était bien reçue, même si parfois il fallait repasser une seconde fois ou se rapprocher un peu plus, ce qui est normal avec l'IR en extérieur. Ces essais ont permis d'identifier quelques points à améliorer, notamment au niveau de l'animation LED et de la stabilité de l'émission IR.

La réunion s'est conclue avec une liste de petites corrections à apporter pour rendre le système plus clair et plus fiable, ainsi qu'un retour global positif sur le fonctionnement général du prototype.





IV.3 - Réunion du 30/05/2026

Participants : HACENE Fatma & ABDAT Soheybe

Absents : PIRES SALVADOR Kevin, BLANC Thomas & CHAIDA-BOURDOT Mohamed

Ordre du jour

Objectif : Démonstration du prototype du jeu en extérieur

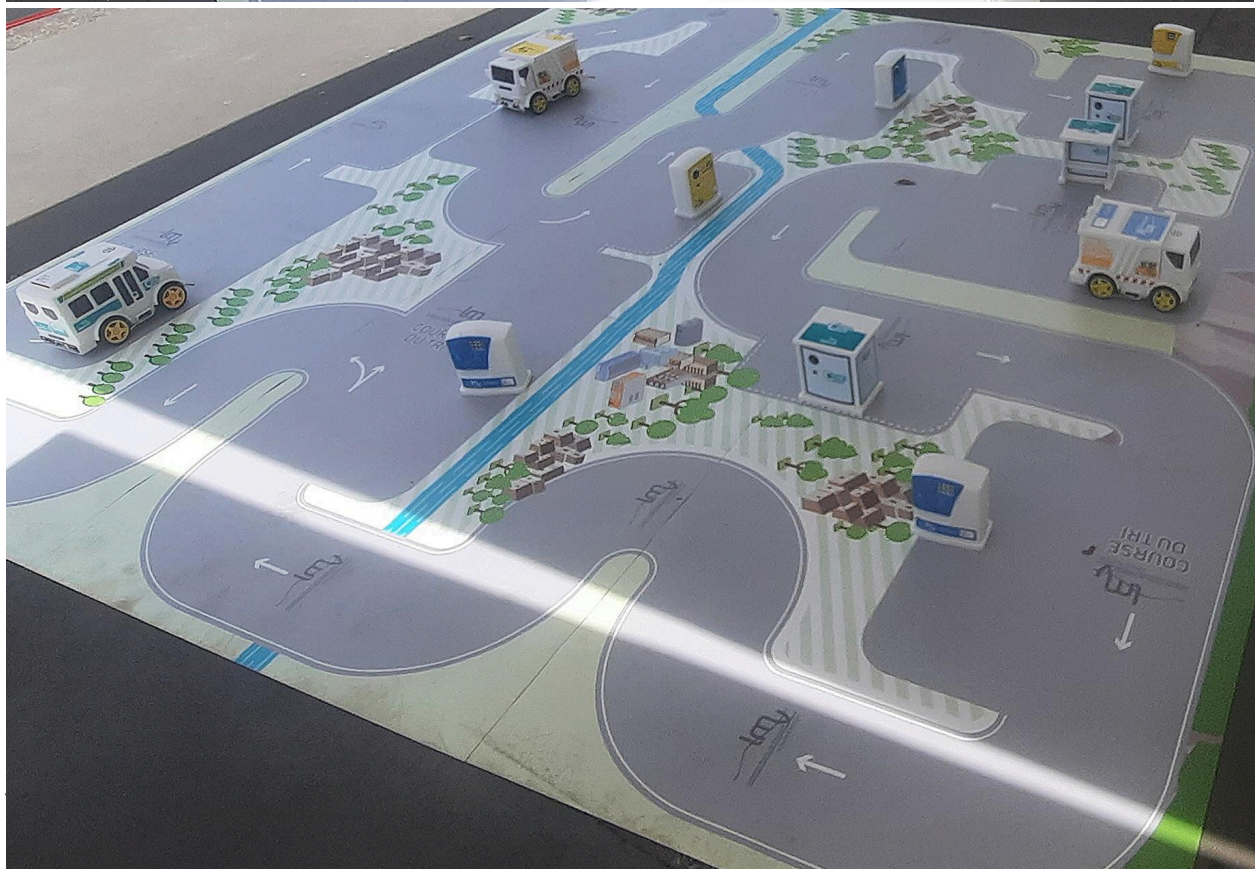
On se retrouve avec l'événement LMVélo qui est géré par LMV Agglomération. Il se passe devant la gare de Cavaillon. Fatma et Soheybe se sont occupés d'animer le stand du projet ESTOM avec des personnes de LMV. Ceci était une démonstration dans une situation réelle.

Munis d'une tablette et des téléphones fournis par LMV, ainsi que le tapis de la route, Fatma et Soheybe ont fait découvrir le projet aux gens qui s'y sont intéressés. Des enfants, ainsi que des animateurs et le personnel de LMV ont joué au jeu.

C'est aussi aujourd'hui que LMV nous a donné le nom du jeu, **Course du Tri**, qui est écrit sur le circuit. Les retours des clients et des personnes étaient positifs mais il reste quand même beaucoup de corrections:

- Le faisceau de l'infrarouge est trop petit donc il fallait se coller au PAV pour pouvoir récupérer les déchets.
- La connexion de la Raspberry Pi, que se soit la Pi 3 qu'on travaillait avec ou une nouvelle Pi 5, est limitée à seulement 7 connexions, donc il fallait limiter le nombre de BOMs et de PAVs qui se connecte.
- Parfois, les PAVs se déconnectent mais en redémarrant la supervision, ils retrouvent la connexion et on peut donc continuer.
- Les PAVs et abribus n'ont pas de signalétique pour montrer l'emplacement de l'infrarouge. Ceci va être géré par LMV.

Malgré les défauts, les enfants qui ont participé se sont amusés et les clients sont satisfaits de l'avancement du projet.





V - Fiches d'audit de sécurité et recettes

V.1 - Fiche d'audit de sécurité

Fiche d'audit de sécurité.		Réf. FASxxxxxx
Projet	Auteur	Date
Projet ESTOM	Fatma HACENE	30/05/2026
Partie de projet		
Superviseur		
Désignation	Risque	Solution
Accès WEB	Intrusion Injection de requête SQL Lecture de trame	Passer le site en HTTPS Sécuriser les formulaires Sécuriser les droits des fichiers sur le serveur
SGBD	Attaque pour trouver le mot de passe admin ou autre Piratage des données	Ex : Mot de passe complexe du SGBD. Chiffrement complexe du mot de passe.
WIFI	Accès non autorisé au réseau Interception de trafic Man-In-The-Middle Contournement du réseau local	Utilisation de WPA2 avec chiffrement AES (CCMP) Utiliser un mot de passe unique et fort Désactiver le Wi-Fi Protected Setup (évite les vulnérabilités brute-force)
Synthèse		
Le serveur Web est en HTTPS et les formulaires sont sécurisés. La base de données est configurée avec un mot de passe complexe et sécurisé, dont l'utilisateur du système a des droits restreint. Le point d'accès Wi-Fi est bien configuré avec un mot de passe complexe et avec la méthode WPA2-PSK et le chiffrement AES. Le WPS est bien désactivé.		

V.2 - Fiches de recette

FICHE RECETTE, QUALIFICATION DU SYSTÈME

Nom du système :	ESTOM	Code de la campagne de test :	
Technicien 1 :	Fatma HACENE	Date :	30/05/2026
Technicien 2 :	Soheybe ABDAT		
Technicien 3 :	Mohamed CHAIDA-BOURDOT		
Technicien 4 :	Thomas BLANC		

IDENTIFICATION DU SCÉNARIO

Identification du scénario de recette	
Titre :	Initialisation du système
Objectif du scénario :	Observer si tout le système s'initialise correctement et entièrement.

CONDITIONS INITIALES NÉCESSAIRES POUR EFFECTUER LA RECETTE

La supervision doit être alimentée, les BOM, PAV, bus et abribus sont chargés.

EXÉCUTION DU TEST

Description courte :	Allumer tout le matériel.
Résultats attendus :	Le matériel peut se connecter au Wi-Fi, puis au serveur. Les LEDs des BOM, PAV, bus et abribus doivent arrêter de clignoter après un certain temps.

BILAN

Quelques matériaux peuvent se connecter sur le Wi-Fi et donc aussi le serveur.
D'autres ne peuvent pas s'y connecter et donc le système entier n'est pas initialisé, seulement une partie.

REMARQUES

La Raspberry Pi a une limite de 7 connexions. Il faudrait peut-être utiliser une clé Wi-Fi.

CONCLUSION

VALIDE		NON VALIDE	X
--------	--	------------	---

(Mettre une croix dans la case correspondante)

FICHE RECETTE, QUALIFICATION DU SYSTÈME

Nom du système :	ESTOM	Code de la campagne de test :	
Technicien 1 :	Fatma HACENE	Date :	30/05/2026
Technicien 2 :	Soheybe ABDAT		
Technicien 3 :	Mohamed CHAIDA-BOURDOT		
Technicien 4 :	Thomas BLANC		

IDENTIFICATION DU SCÉNARIO

Identification du scénario de recette	
Titre :	Initialisation de la supervision
Objectif du scénario :	Observer si la supervision s'initialise correctement.

CONDITIONS INITIALES NÉCESSAIRES POUR EFFECTUER LA RECETTE

La supervision doit être alimentée.

EXÉCUTION DU TEST

Description courte :	Brancher la supervision.
Résultats attendus :	La Raspberry Pi s'initialise correctement, le Wi-Fi est joignable et le serveur est en route. Le site Web devient accessible.

BILAN

Le point d'accès Wi-Fi est opérationnel, le serveur tourne dès le démarrage et le site Web est facilement joignable dès l'alimentation de la Raspberry Pi.

REMARQUES

--

CONCLUSION

VALIDE	X	NON VALIDE	
--------	----------	------------	--

(Mettre une croix dans la case correspondante)

FICHE RECETTE, QUALIFICATION DU SYSTÈME

Nom du système :	ESTOM	Code de la campagne de test :	
Technicien 1 :	Fatma HACENE	Date :	30/05/2026
Technicien 2 :	Soheybe ABDAT		
Technicien 3 :	Mohamed CHAIDA-BOURDOT		
Technicien 4 :	Thomas BLANCH		

IDENTIFICATION DU SCÉNARIO

Identification du scénario de recette	
Titre :	Départ de la partie
Objectif du scénario :	Observer si les BOM, PAV, bus et abribus reconnaissent et se mettent à l'état de départ de partie.

CONDITIONS INITIALES NÉCESSAIRES POUR EFFECTUER LA RECETTE

Les matériaux doivent être alimentés, connectés à la supervision et initialisés.

EXÉCUTION DU TEST

Description courte :	Initialisation de la partie par le site Web en cliquant sur le bouton "Lancer la partie".
Résultats attendus :	Les LEDs des PAV et abribus sont allumés (l'état plein) et que le BOM et l'abribus peuvent détecter l'infrarouge.

BILAN

La partie démarre et on observe bien les LEDs des PAV qui s'allument.

REMARQUES

Il se trouve qu'il y a un petit délai entre le départ de la partie et la réaction des PAV à l'ordre.

CONCLUSION

VALIDE	X	NON VALIDE	
--------	---	------------	--

(Mettre une croix dans la case correspondante)

FICHE RECETTE, QUALIFICATION DU SYSTÈME

Nom du système :	ESTOM	Code de la campagne de test :	
Technicien 1 :	Fatma HACENE	Date :	30/05/2026
Technicien 2 :	Soheybe ABDAT		
Technicien 3 :	Mohamed CHAIDA-BOURDOT		
Technicien 4 :	Thomas BLANC		

IDENTIFICATION DU SCÉNARIO

Identification du scénario de recette	
Titre :	Départ transfert
Objectif du scénario :	Observation de la détection du PAV par le BOM et initialisation de l'état départ transfert.

CONDITIONS INITIALES NÉCESSAIRES POUR EFFECTUER LA RECETTE

Les matériaux doivent être alimentés, connectés à la supervision et en cours de partie.
Un BOM doit être devant un PAV de la même couleur.

EXÉCUTION DU TEST

Description courte :	Placer le BOM devant l'infrarouge du PAV.
Résultats attendus :	Le BOM doit reconnaître le PAV puis envoyer le status départ transfert au superviseur. Les LEDs du BOM clignotent alors que ceux du PAV s'éteignent de haut en bas.

BILAN

Le BOM reconnaît bien le PAV de même couleur et donc initialise le transfert.

REMARQUES

Le faisceau de l'infrarouge est très petit, il faut se rapprocher vraiment du PAV mais faire attention à ne pas avoir de choc.

CONCLUSION

VALIDE	X	NON VALIDE	
--------	---	------------	--

(Mettre une croix dans la case correspondante)

FICHE RECETTE, QUALIFICATION DU SYSTÈME

Nom du système :	ESTOM	Code de la campagne de test :	
Technicien 1 :	Fatma HACENE	Date :	30/05/2026
Technicien 2 :	Soheybe ABDAT		
Technicien 3 :	Mohamed CHAIDA-BOURDOT		
Technicien 4 :	Thomas BLANC		

IDENTIFICATION DU SCÉNARIO

Identification du scénario de recette	
Titre :	Fin transfert
Objectif du scénario :	Observation de la fin de transfert entre le BOM et le PAV.

CONDITIONS INITIALES NÉCESSAIRES POUR EFFECTUER LA RECETTE

Les matériaux doivent être alimentés, connectés à la supervision et en cours de partie.
 Un BOM doit être devant un PAV de la même couleur.
 Le départ transfert doit être déjà initié.

EXÉCUTION DU TEST

Description courte :	Le BOM ne doit pas bouger pendant 5 secondes.
Résultats attendus :	Le BOM doit envoyer le status fin transfert au superviseur. 2 LEDs du BOM s'allument et le PAV passe en état vide avec les LEDs et l'infrarouge éteints.

BILAN

Le BOM finit bien le transfert et le PAV s'éteint.

REMARQUES

--

CONCLUSION

VALIDE	X	NON VALIDE	
--------	----------	------------	--

(Mettre une croix dans la case correspondante)

FICHE RECETTE, QUALIFICATION DU SYSTÈME

Nom du système :	ESTOM	Code de la campagne de test :	
Technicien 1 :	Fatma HACENE	Date :	30/05/2026
Technicien 2 :	Soheybe ABDAT		
Technicien 3 :	Mohamed CHAIDA-BOURDOT		
Technicien 4 :	Thomas BLANC		

IDENTIFICATION DU SCÉNARIO

Identification du scénario de recette	
Titre :	Annuler transfert
Objectif du scénario :	Observation de l'arrêt de détection de l'infrarouge par le BOM et initialisation de l'état annulation transfert.

CONDITIONS INITIALES NÉCESSAIRES POUR EFFECTUER LA RECETTE

Les matériaux doivent être alimentés, connectés à la supervision et en cours de partie.
 Un BOM doit être devant un PAV de la même couleur.
 Le départ transfert doit être déjà initié.

EXÉCUTION DU TEST

Description courte :	Faire bouger le BOM avant que les 5 secondes aboutissent.
Résultats attendus :	Le BOM doit reconnaître l'absence de l'infrarouge et envoie le status d'annulation de transfert au superviseur. Les LEDs du BOM s'éteignent alors que celles du PAV se rallument.

BILAN

Lorsque le BOM est bougé, il se remet au remplissage précédent et le PAV revient à l'état plein.

REMARQUES

--

CONCLUSION

VALIDE	X	NON VALIDE	
--------	---	------------	--

(Mettre une croix dans la case correspondante)

FICHE RECETTE, QUALIFICATION DU SYSTÈME

Nom du système :	ESTOM	Code de la campagne de test :	
Technicien 1 :	Fatma HACENE	Date :	30/05/2026
Technicien 2 :	Soheybe ADBAT		
Technicien 3 :	Mohamed CHAIDA-BOURDOT		
Technicien 4 :	Thomas BLANC		

IDENTIFICATION DU SCÉNARIO

Identification du scénario de recette	
Titre :	Test couleurs différente PAV et BOM
Objectif du scénario :	Confirmer que le BOM ne réagit pas lorsqu'il se place devant un PAV de couleur différente.

CONDITIONS INITIALES NÉCESSAIRES POUR EFFECTUER LA RECETTE

Le système doit être initialisé et la partie doit être en cours. Le PAV et BOM doivent être de couleurs différentes.

EXÉCUTION DU TEST

Description courte :	Le DIP switch du PAV et du BOM sont réglés sur différents bits qui leur donnent une couleur différente. On place le BOM devant le PAV.
Résultats attendus :	Le message de départ transfert ne doit pas être envoyé par le BOM. Le PAV reste en état plein.

BILAN

Le BOM de couleur jaune ne récupère pas les déchets du PAV de couleur bleu.

REMARQUES

CONCLUSION

VALIDE	X	NON VALIDE	
--------	---	------------	--

(Mettre une croix dans la case correspondante)

FICHE RECETTE, QUALIFICATION DU SYSTÈME

Nom du système :	ESTOM	Code de la campagne de test :	
Technicien 1 :	Fatma HACENE	Date :	30/05/2026
Technicien 2 :	Soheybe ABDAT		
Technicien 3 :	Mohamed CHAIDA-BOURDOT		
Technicien 4 :	Thomas BLANC		

IDENTIFICATION DU SCÉNARIO

Identification du scénario de recette	
Titre :	Collision
Objectif du scénario :	Détection de collision par le BOM.

CONDITIONS INITIALES NÉCESSAIRES POUR EFFECTUER LA RECETTE

Les matériaux doivent être alimentés, connectés à la supervision et en cours de partie.
Il faut 2 BOMs assez proches.

EXÉCUTION DU TEST

Description courte :	Les BOMs doivent créer une collision.
Résultats attendus :	Le BOM doit envoyer un status de choc et le site Web doit afficher le choc.

BILAN

Les capteurs de chocs fonctionnent, on voit bien des chocs sur le site Web.

REMARQUES

Le robot dans les BOM peut parfois être un parasite pour le capteur.

CONCLUSION

VALIDE	X	NON VALIDE	
--------	---	------------	--

(Mettre une croix dans la case correspondante)

FICHE RECETTE, QUALIFICATION DU SYSTÈME

Nom du système :	ESTOM	Code de la campagne de test :	
Technicien 1 :	Fatma HACENE	Date :	30/05/2026
Technicien 2 :	Soheybe ABDAT		
Technicien 3 :	Mohamed CHAIDA-BOURDOT		
Technicien 4 :	Thomas BLANC		

IDENTIFICATION DU SCÉNARIO

Identification du scénario de recette	
Titre :	Fin de Partie, temps imparti
Objectif du scénario :	Observation de la réinitialisation des matériaux dès la fin de la partie

CONDITIONS INITIALES NÉCESSAIRES POUR EFFECTUER LA RECETTE

Les matériaux doivent être alimentés, connectés à la supervision et en cours de partie.
La partie doit arriver à son terme dans le temps imparti.

EXÉCUTION DU TEST

Description courte :	Affiche le résultat de la partie et remet tout à l'état initial
Résultats attendus :	Les matériaux se remettent à leur état initial, que les LEDs sont éteintes et que le site Web affiche les résultats de la partie.

BILAN

Dès que le temps s'écoule, la partie est terminée et le podium est affiché.

REMARQUES

CONCLUSION

VALIDE	X	NON VALIDE	
--------	---	------------	--

(Mettre une croix dans la case correspondante)

FICHE RECETTE, QUALIFICATION DU SYSTÈME

Nom du système :	ESTOM	Code de la campagne de test :	
Technicien 1 :	Fatma HACENE	Date :	30/05/2026
Technicien 2 :	Soheybe ABDAT		
Technicien 3 :	Mohamed CHAIDA-BOURDOT		
Technicien 4 :	Thomas BLANC		

IDENTIFICATION DU SCÉNARIO

Identification du scénario de recette	
Titre :	Fin de Partie, site Web
Objectif du scénario :	Observation de la réinitialisation des matériaux dès la fin de la partie.

CONDITIONS INITIALES NÉCESSAIRES POUR EFFECTUER LA RECETTE

Les matériaux doivent être alimentés, connectés à la supervision et en cours de partie.
La partie doit être arrêtée par le site Web.

EXÉCUTION DU TEST

Description courte :	On clique sur le bouton "Réinitialiser la partie".
Résultats attendus :	Les matériaux se remettent à leur état initial, que les LEDs sont éteintes.

BILAN

En cliquant sur le bouton, la partie s'achève immédiatement.

REMARQUES

CONCLUSION

VALIDE	X	NON VALIDE	
--------	---	------------	--

(Mettre une croix dans la case correspondante)

Partie Personnelle IR1: BOM/Bus

CHAIDA-BOURDOT Mohamed

I - Introduction personnelle

Dans le cadre de notre projet de développement débuté en janvier, j'ai été affecté à la partie logicielle et matérielle embarquée, principalement axée sur la communication et la détection d'état de la BOM (Benne à Ordures Ménagère) et du PAV (Point d'Apport Volontaire).

L'objectif de cette première phase était de m'approprier l'environnement de développement. Ce travail a nécessité une forte collaboration, non seulement avec mes professeurs (Mr. Antoine, Mr. Hortolland), mais aussi avec les étudiants en électronique (ER), afin de lier la partie développement et la partie physique.

II - Objectifs et Cahier des Charges

- **Objectif global du projet :** Rendre la BOM et le PAV intelligents et communicants pour la supervision de la partie.
- **Mes objectifs techniques spécifiques :**
 - Assurer l'identification fiable de l'engin (couleur / type).
 - Détecter les collisions en temps réel sans fausser les scores.
 - Maintenir une communication constante et bidirectionnelle avec le serveur central (envoi de l'état, réception d'ordres).

DIAGRAMME DE GANTT	20		▲ Activité de Mohamed	155 h	Lun 05/01/26	Mar 16/06/26	
	21		Etude du programme BOM	20 h	Lun 12/01/26	Mar 27/01/26	4
	22		Réalisation du cahier de recette	10 h	Mar 27/01/26	Mar 03/02/26	21
	23		Prise en main de l'Arduino IDE	8 h	Mar 03/02/26	Mar 10/02/26	22
	24		Mise en oeuvre de la solution 2025	40 h	Mar 10/02/26	Lun 30/03/26	23
	25		Partie Physique	4 h	Lun 01/06/26	Mar 02/06/26	29
	26		TMO du bandeau LED	15 h	Lun 30/03/26	Mar 28/04/26	24
	27		Intégration bandeau LED dans le code du	4 h	Mar 28/04/26	Lun 04/05/26	26
	28		TMO des capteurs chocs	25 h	Lun 04/05/26	Mar 26/05/26	27
	29		Intégration des capteurs de chocs dans le code du	4 h	Mar 26/05/26	Lun 01/06/26	28
	30		Rédaction du dossier de projet	20 h	Lun 01/06/26	Mar 16/06/26	29

III - Mon Rôle et Mon Environnement

- **Ma mission** : Développement embarqué (C++) et validation matérielle des capteurs avant leur intégration par les ER.
- **Mon environnement de travail** :
 - *Matériel* : Microcontrôleur XIAO ESP32S3, Breadboards, Capteurs de choc (poussoirs), DIP Switchs, LED IR, Oscilloscope.
 - *Logiciel* : Arduino IDE.

IV - Mes Réalisations Techniques

- **A. Identification du système (Les DIP Switchs)** :
 - *Action* : Câblage et programmation de l'identification de la benne.
 - *Technique* : Configuration matérielle en **INPUT_PULLUP** (pour éviter les états flottants) et opérations binaires (Bitwise) pour combiner 3 interrupteurs en une seule valeur d'identification.
 - *Visuels* :

BOM/PAV	DP SW 1	DP SW 2	DP SW 3
GRIS	ON	OFF	OFF
VERT	ON	ON	OFF
BLEU	ON	OFF	ON
JAUNE	ON	ON	ON
BUS/ABRI BUS	OFF	X	X

000= 0 ; 001 =1 ; 010 =2 ; 011= 3 ; BUS/ABRI BUS= DP SW 1 = OFF

MONTAGE : VOIR ANNEXE 2

- **B. Détection d'événements immédiate (Le Capteur de Choc)** :
 - *Action* : Création d'un système de détection instantanée des collisions.
 - *Technique* : Mise en place d'**interruptions matérielles (FALLING)** pour garantir qu'aucun choc ne soit raté, même si le processeur est occupé à faire autre chose.

- *Visuels :*

```

21 // --- FONCTION D'INTERRUPTION ---
22 void IRAM_ATTR gestionChoc() {
23     // 1. On coupe immédiatement l'écoute !
24     // Le processeur ne peut plus être interrompu par les rebonds ou le relâchement
25     detachInterrupt(digitalPinToInterrupt(PIN_CHOC));
26
27     // 2. On lève le drapeau
28     chocDetecte = true;
29 }

```

- **C. Communication et Synchronisation Serveur :**
 - *Action :* Structuration des trames de données réseau.
 - *Technique :* Utilisation de la bibliothèque ArduinoJson pour formater les envois (Type, Couleur, Progression, Collisions) et traitement de la logique des ordres reçus (ex: reprise de progression après une coupure).
 -

V - Résolution de Problèmes

- **A. Le blocage de l'environnement de développement :**
 - *Problème :* Impossibilité de téléverser le code sur l'ESP32 au début du projet.
 - *Solution :* Recherche et modification directe du fichier de configuration `arduino-cli.yaml` (ajout de `connection_timeout: 300s`).
- **B. Le défi de l'Anti-Rebond (Debouncing) du capteur de choc :**
 - *Problème :* Les vibrations mécaniques du bouton généraient de multiples faux contacts (rebonds), incrémentant le compteur de manière erronée.
 - *Solution :* Développement d'un filtre logiciel utilisant `millis()` et la logique `detachInterrupt` pour créer une "fenêtre sourde" (ex: 250ms) ignorant les rebonds physiques.
 - *Visuels :*

VI - Travail d'Équipe et Intégration

- **Collaboration ER (Étude & Réalisation) :** Tests conjoints avec Thomas Blanc sur le capteur de choc physique. Validation de mes codes sur leurs montages matériels.
- **Soutien technique :** Collaboration avec Soheybe sur le débogage de l'adressage des broches.

VII - Les Prochaines Étapes

```

test-CHOC.ino
1 /*
*** CHOC DÉTECTÉ *** -> Total : 222
--- TEST CAPTEUR CHOC DEMARRÉ ---
Compteur actuel : 0
*** CHOC DÉTECTÉ *** -> Total : 1
*** CHOC DÉTECTÉ *** -> Total : 2
*** CHOC DÉTECTÉ *** -> Total : 3
*** CHOC DÉTECTÉ *** -> Total : 4
*** CHOC DÉTECTÉ *** -> Total : 5
*** CHOC DÉTECTÉ *** -> Total : 6
*** CHOC DÉTECTÉ *** -> Total : 7
*** CHOC DÉTECTÉ *** -> Total : 8
*** CHOC DÉTECTÉ *** -> Total : 9
*** CHOC DÉTECTÉ *** -> Total : 10
*** CHOC DÉTECTÉ *** -> Total : 11
*** CHOC DÉTECTÉ *** -> Total : 12
*** CHOC DÉTECTÉ *** -> Total : 13
*** CHOC DÉTECTÉ *** -> Total : 14
*** CHOC DÉTECTÉ *** -> Total : 15
*** CHOC DÉTECTÉ *** -> Total : 16
*** CHOC DÉTECTÉ *** -> Total : 17
*** CHOC DÉTECTÉ *** -> Total : 18
*** CHOC DÉTECTÉ *** -> Total : 19
*** CHOC DÉTECTÉ *** -> Total : 20
*** CHOC DÉTECTÉ *** -> Total : 21
*** CHOC DÉTECTÉ *** -> Total : 22
*** CHOC DÉTECTÉ *** -> Total : 23
*** CHOC DÉTECTÉ *** -> Total : 24
*** CHOC DÉTECTÉ *** -> Total : 25
*** CHOC DÉTECTÉ *** -> Total : 26
*** CHOC DÉTECTÉ *** -> Total : 27
*** CHOC DÉTECTÉ *** -> Total : 28
*** CHOC DÉTECTÉ *** -> Total : 29
*** CHOC DÉTECTÉ *** -> Total : 30
*** CHOC DÉTECTÉ *** -> Total : 31
*** CHOC DÉTECTÉ *** -> Total : 32
*** CHOC DÉTECTÉ *** -> Total : 33
*** CHOC DÉTECTÉ *** -> Total : 34
*** CHOC DÉTECTÉ *** -> Total : 35
*** CHOC DÉTECTÉ *** -> Total : 36
*** CHOC DÉTECTÉ *** -> Total : 37

```

- **Architecture Logicielle** : Restructuration complète du code en Programmation Orientée Objet (création des classes .cpp et .h) pour le rendre plus professionnel, lisible et maintenable.
- **Transposition** : Adapter le programme validé du PAV pour la version finale de la BOM, et finaliser le traitement des trames Infrarouges (RX_ESP32).
- **Test grandeur nature** : Tester l'ensemble des fonctionnalités sur le terrain avec les BOM et PAV ainsi que la raspberry en réel.

VIII - Bilan de la Phase Précédente

1. Bilan de la Revue 1 : Ce qui fonctionnait déjà (La Théorie) Pendant, ma première revue de projet, mon travail s'était concentré sur la validation individuelle des composants sur table (breadboard), j'avais validé trois fonctionnalités :

- L'identification matérielle : La lecture fiable des *DIP Switchs* pour définir si le véhicule est un Bus ou une BOM et de quelle couleur il est (Jaune, Vert, Bleu, Gris).
- La détection des chocs : La mise en place du capteur de collision avec un système d'interruption au front descendant pour ne rater aucun obstacle.

- La communication de base : La capacité de l'ESP32 à se connecter au Wi-Fi et à envoyer une première version de données au format texte (JSON) vers la supervision.

À la fin de la Revue 1, j'avais un système où chaque élément fonctionnait très bien de son côté, de manière isolée.

2. L'enjeu de la Revue 2 : L'objectif de cette deuxième phase de projet n'était plus de créer des morceaux de code, mais de faire l'intégration totale. Il fallait assembler toutes ces fonctionnalités pour qu'elles fonctionnent en même temps.

Cette phase a consisté à :

1. S'adapter aux évolutions du réseau : Le protocole de communication de la supervision a changé (passage à la version 1.6), il a fallu mettre à jour le "langage" du camion.
2. Rendre le camion robuste : Gérer les imprévus d'une vraie partie de jeu (perte de réseau, spam des capteurs infrarouges, batterie faible).
3. Le débogage sur le terrain : Passer des tests sur un bureau aux tests "grandeur nature" avec les vrais camions imprimés, ce qui a révélé des comportements inattendus qu'il a fallu analyser et réparer en direct.

IX. Évolution et Sécurisation de la Communication

Évolution du protocole de communication JSON (De la v1.0 à la v2.0)

Au cours de la phase d'intégration du projet, le protocole de communication entre la Benne à Ordures Ménagères (BOM) et le Superviseur a subi plusieurs révisions majeures. L'objectif de ces évolutions était d'améliorer la fiabilité des échanges sur le réseau Wi-Fi, de standardiser le nommage des variables, et d'intégrer de nouvelles mécaniques de jeu.

Voici les trois grandes étapes de cette évolution :

IX.1 - Version 1.0: Le prototype initial

La première version du protocole posait les bases de la communication. Les données étaient envoyées sous forme de valeurs numériques brutes (entiers) et le nommage des clés utilisait des majuscules. Les variables étaient parfois spécifiques au matériel (comme la notion de "Progression" pour les LEDs).

- Exemple - Trame d'initialisation (Superviseur vers BOM) en v1.0 :
- JSON

```
{
  "Ordre": 0,
  "Etatv": 1,
  "NbChocs": 2,
  "Progression": 35
}
```

IX.2 - Version 1.6: La standardisation et la fiabilisation par le texte

Lors des premiers essais réseau, nous avons constaté que l'interprétation de nombres bruts pouvait provoquer des instabilités. La version 1.6 a introduit un changement architectural majeur : toutes les valeurs sont désormais transmises sous forme de chaînes de caractères (Strings). De plus, la nomenclature a été lissée (passage en minuscules/camelCase) : la clé **"action"** est devenue **"status"**, **"Ordre"** est devenu **"ordre"**, et **"Progression"** est devenu **"leds"**. Enfin, cette version a ajouté la gestion dynamique de la **"luminosite"** pour l'affichage NeoPixel de la BOM.

- Exemple - Trame d'initialisation (Superviseur vers BOM) en v1.6 :
- JSON

```
{
  "ordre": "0",
  "etat": "1",
  "nbChocs": "2",
  "leds": "4"
}
```

IX.3 - Version 2.0: L'aboutissement et la distinction des rôles

La version finale (v2.0) du 29 avril 2026 vient clarifier définitivement le rôle de chaque équipement sur le plateau. Le vocabulaire a été affiné pour éviter toute confusion lors de la programmation de la machine à états : **"etat"** devient **"etatJeu"** et **"nbChocs"** devient **"collisions"**. Surtout, cette version fait la distinction stricte entre les acteurs mobiles (BOM / BUS) et les infrastructures fixes (PAV / Abri Bus, noté "ABB"). Le PAV a désormais ses propres

états de jeu spécifiques (Vide "20", Plein "21", Vidage "22") qu'il peut communiquer au superviseur.

- Exemple - Trame d'initialisation (Superviseur vers BOM) en v2.0 finale :
- JSON

```
{
  "ordre": "0",
  "etatJeu": "1",
  "collisions": "2",
  "leds": "4"
}
```

Bilan : Ces mises à jour successives m'ont contraint à adapter régulièrement le code C++ de l'ESP32. La transition vers le format "String" de la v1.6 m'a particulièrement poussé à repenser la manière dont le microcontrôleur lit et convertit (**.toInt()**) les trames réseau, garantissant aujourd'hui une parfaite stabilité lors des tests grandeur nature.

X - Refonte de l'Architecture et Nouvelles Fonctionnalités

Une fois la communication réseau stabilisée, il a fallu adapter le "cerveau" du camion. Le code qui fonctionnait très bien sur un bureau s'est révélé trop fragile pour une vraie partie de jeu. Nous avons donc procédé à une refonte de l'architecture logicielle pour la rendre "à l'épreuve des joueurs".

X.1 - La Modularité par la Programmation Orientée Objet (POO)

Au fur et à mesure que nous ajoutons des fonctionnalités au camion (Wi-Fi, JSON, Infrarouge, Capteurs), notre fichier principal risquait de devenir un "code spaghetti" : un programme gigantesque, illisible, où la moindre modification risquait de créer des bugs inattendus.

Pour sécuriser notre développement, nous avons fait le choix de basculer vers une architecture en Programmation Orientée Objet (C++). Le principe a été de découper notre système en plusieurs "boîtes noires" indépendantes, appelées des Classes.

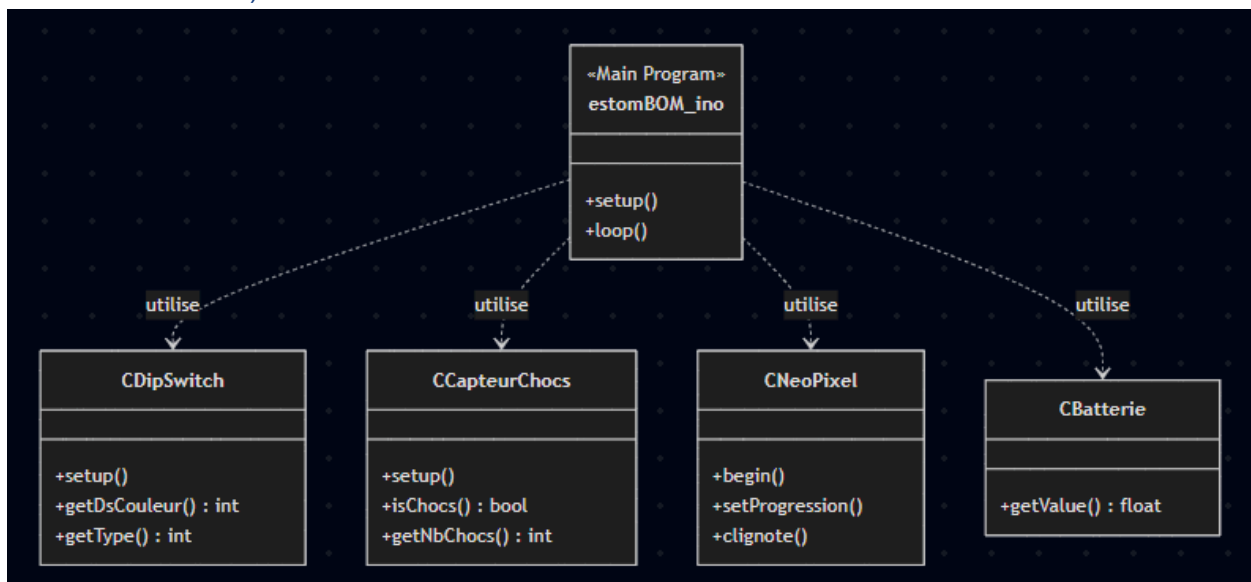
Concrètement, j'ai réparti les fonctionnalités matérielles dans des fichiers séparés (**.cpp** et **.h**):

- **CDipSwitch** : Gère uniquement la lecture des interrupteurs pour déduire la couleur et le type du camion.

- **CCapteurChocs** : S'occupe exclusivement de la détection physique des collisions grâce aux interruptions matérielles.
- **CNeoPixel** : Gère toute la logique lumineuse de l'anneau LED (couleurs, clignotement).
- **CBatterie** : Une classe ajoutée récemment, dont le seul but est de surveiller la tension d'alimentation.

Les bénéfices de cette approche pour notre projet :

1. **La lisibilité** : Notre fichier principal (**estomBOM.ino**) ne gère plus la complexité matérielle. Il se contente de donner des ordres simples, comme **afficheur.clignote()** ou **cc.getNbChocs()**. Il se concentre uniquement sur la logique du jeu et le réseau.
 2. **Le débogage facilité** : Si les LEDs réagissent mal, je sais que je dois chercher l'erreur uniquement dans les fichiers **cneopixel.cpp**, sans risquer de casser le capteur de chocs par accident.
 3. **L'évolutivité** : Cette structure nous permet de créer le code du "Bus" ou d'un "Nouveau Camion" très rapidement, en réutilisant simplement nos "briques" (les classes) déjà testées et validées.
- Diagramme de classe simplifié (montrant l'interaction entre **estomBOM.ino** et les fichiers **.h**).



X.2 - Stabilisation de l’affichage (Migration vers NeoPixel)

À l'origine, nous utilisons la bibliothèque **FastLED** pour piloter l'anneau lumineux du camion. Cependant, lors de l'intégration avec le Wi-Fi, nous avons remarqué des instabilités : les interruptions réseau entraient parfois en conflit avec le rafraîchissement des LEDs.

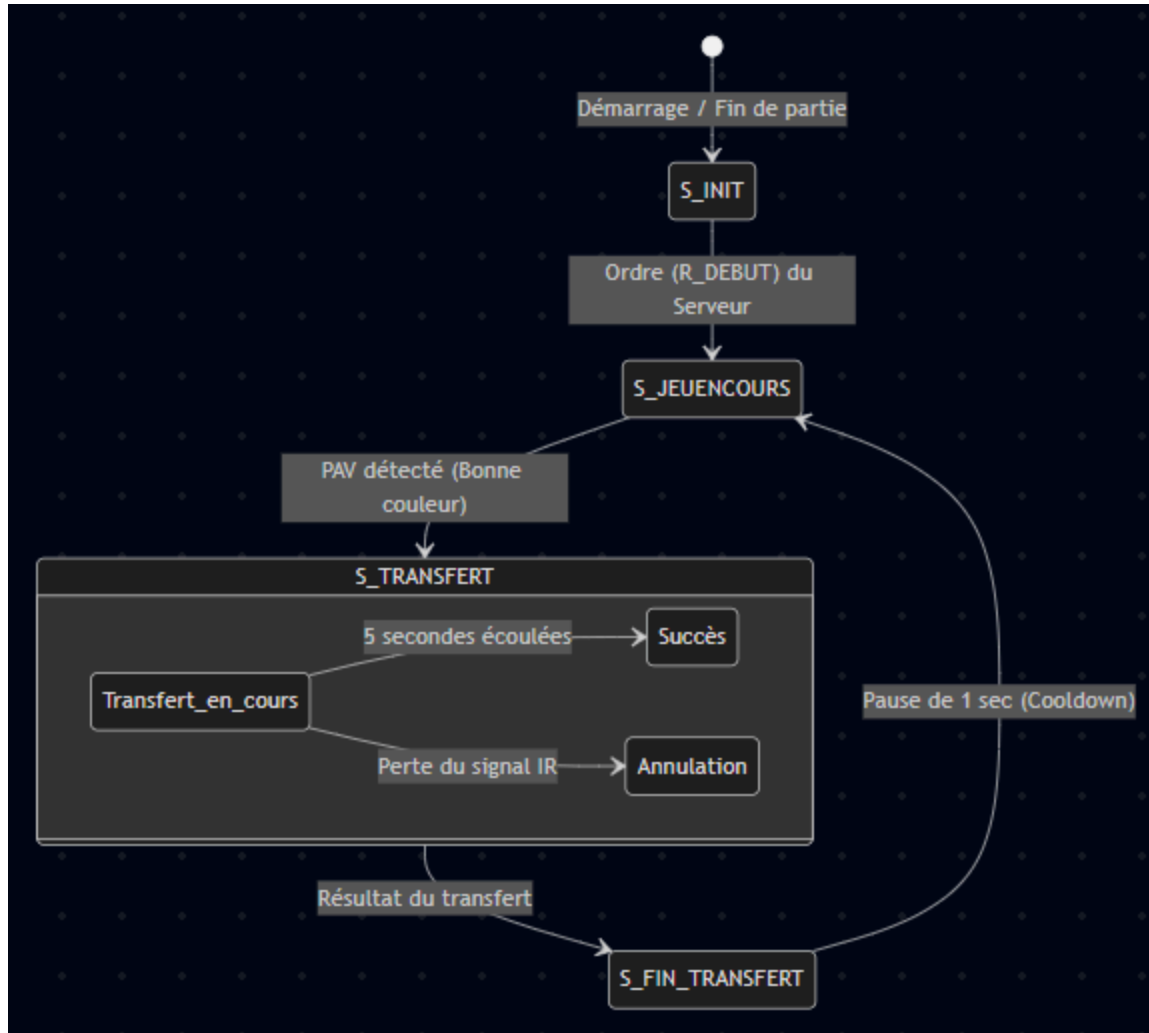
Pour garantir un affichage fluide et sans crash, nous avons migré toute la gestion lumineuse vers la bibliothèque Adafruit NeoPixel. J'ai intégré cette technologie à travers une nouvelle classe (**CNeoPixel**) qui gère les 7 LEDs de l'anneau de manière autonome. Désormais, la LED centrale (n°0) sert de "témoin de fonctionnement", tandis que les LEDs extérieures (1 à 6) affichent la jauge de progression des déchets.

X.3 - Mécanique de jeu: Le système “Anti-Spam” (Cooldown)

Lors des premiers tests logiques, un défaut de jouabilité est apparu : si un joueur réussissait un transfert de 5 secondes mais laissait son camion garé devant le Point d'Apport Volontaire (PAV), le capteur infrarouge relançait immédiatement un nouveau transfert en boucle.

La solution logicielle : Nous avons modifié la machine à états du programme en ajoutant un état transitoire appelé **S_FIN_TRANSFERT**. Désormais, qu'un transfert réussisse ou soit annulé, le camion entre dans cet état et déclenche un chronomètre interne d'une seconde. Pendant ce laps de temps, le camion s'endort volontairement et ignore les signaux infrarouges. Ce "cooldown" (temps de recharge) laisse le temps au joueur d'éloigner son camion sans déclencher d'action parasite.

Diagramme d'état (Machine à états) simplifié montrant la boucle **S_JEUENCOURS** -> **S_TRANSFERT** -> **S_FIN_TRANSFERT**.



X.4 - Surveillance matérielle: La jauge de batterie intelligente

Sur un plateau de jeu, une panne de batterie soudaine d'un véhicule ruinerait l'expérience de la supervision et des joueurs. Nous avons donc ajouté une fonctionnalité de diagnostic matériel embarqué.

Fonctionnement : Nous avons créé une classe **CBatterie** qui lit la tension analogique sur la broche **GPIO_ANA** (convertie ensuite en Volts réels par le code).

- Si la tension est bonne, la LED centrale de l'anneau NeoPixel affiche la couleur de l'équipe (Jaune, Vert, Bleu ou Gris).
- Si la tension chute en dessous de 6.8V (seuil critique défini dans nos constantes), le système bascule le paramètre **g_batterie_faible** sur **true**. La LED centrale du camion devient instantanément Orange pour alerter le joueur, sans pour autant interrompre la partie en cours.
- Extrait du code de la classe **CBatterie** :

```

if (bf==true) // LED du milieu indice 0

    _pixels.setPixelColor(0, _pixels.Color(orange[0], orange[1], orange[2]));

else

    _pixels.setPixelColor(0, _pixels.Color((couleurs[coul][0]*lum)%256,
(couleurs[coul][1]*lum)%256, (couleurs[coul][2]*lum)%256));

```

XI - Tests Grandeur Nature & Débogage

Jusqu'ici, nos codes fonctionnaient sur maquette de bureau. L'étape cruciale a été l'intégration dans les vrais camions imprimés en 3D et la confrontation aux conditions réelles d'une partie (lumière ambiante, déplacements, réseau). Cette phase a mis en évidence des bugs logiques que j'ai dû diagnostiquer et corriger en direct.

XI.1 - Le bug du “Blackout” visuel (Problème de jauge)

Le symptôme : Lors de nos premières simulations réelles, un comportement très étrange est apparu. Quand le camion effectuait un transfert de déchets, l'anneau clignotait correctement. Mais dès que le transfert réussissait, toutes les LEDs du camion s'éteignaient d'un coup, y compris le voyant central de fonctionnement. Le camion semblait "mort". Pourtant, en l'éteignant et en le rallumant, il affichait par magie la bonne jauge de progression.

Mon diagnostic : J'ai analysé les trames envoyées par le serveur au début de la partie. La supervision avait envoyé le paramètre de jeu "**luminosite**": "0" (pour demander un éclairage faible). En lisant le fichier gérant les LEDs, j'ai compris que la couleur était calculée mathématiquement en se multipliant par cette jauge de luminosité (ex: **Couleur * lum**). Or, multiplier n'importe quelle couleur par 0 donne du Noir absolu ! La jauge s'affichait bien, mais en noir.

La correction : J'ai modifié la fonction **setProgression()** dans la classe **CNeoPixel** pour y ajouter une "sécurité matérielle" empêchant la valeur de tomber à zéro : **if (lum < 1) lum = 1;**. Dès lors, l'éclairage restait visible à sa puissance minimale sans éteindre le camion.

```

void CNeoPixel::setProgression(int coul, int lum, int nb, bool bf) {

    int i;

    if (nb > _numLeds) nb = _numLeds;

```

```

__nbAllumed = nb;

if (lum < 1) lum = 1; // la ligne de code, Bug blackout led début de partie

if (lum > 3) lum = 3;

```

XI.2 - Le piège matériel de l'ESP32-S3 (Conflit de port série USB CDC)

Le symptôme : Lors de l'intégration du capteur infrarouge géré par l'ATtiny85, nous avons utilisé la commande `Serial.read()` pour que la BOM écoute les données de l'ATtiny. Le code était syntaxiquement parfait, le câblage était bon, mais la BOM restait complètement "sourde" et ne recevait aucune donnée.

Mon diagnostic : Le problème ne venait ni de l'électronique ni du code, mais de la configuration de l'environnement de développement (Arduino IDE) pour notre carte spécifique (le XIAO ESP32-S3). Cette puce possède une particularité : elle gère nativement l'USB (USB CDC). Dans le menu "Outils" de l'IDE, un paramètre mal réglé forçait la commande `Serial` à écouter uniquement le port USB (le câble vers l'ordinateur) au lieu d'écouter les broches physiques (UART) reliées à notre ATtiny. En clair, le camion écoutait le mauvais canal !

La correction : J'ai dû reconfigurer correctement les paramètres de compilation dans le menu Outils de l'Arduino IDE (notamment l'option "*USB CDC On Boot*" ou le routage matériel) pour forcer le microcontrôleur à utiliser la vraie liaison série matérielle vers l'ATtiny. Dès lors, la communication s'est mise à fonctionner parfaitement.



Partie Personnelle IR2: PAV/Abribus

ABDAT Soheybe

I - Introduction personnelle

Dans ce projet de deuxième année de BTS CIEL, j'ai été affecté au développement d'une partie du système ESTOM, qui signifie *Espace de Sensibilisation au Tri d'Ordures Ménagères*. Ce projet est demandé par la communauté de communes Luberon Mont de Vaucluse (LMV). L'objectif général est de créer un ensemble d'équipements capables d'aider les habitants à mieux comprendre le tri des déchets, tout en rendant ce tri plus interactif et plus visible.

Dans mon cas, on m'a confié la responsabilité du **Point d'Apport Volontaire (PAV)** ainsi que des **abribus**. Le PAV est un module électronique installé à proximité des conteneurs de tri. Il possède un système de couleur permettant d'identifier le type de déchet associé : Gris, Jaune, Bleu ou Vert. Ces couleurs sont définies grâce à des DIP Switch, ce qui permet de configurer facilement chaque PAV selon son emplacement.

Le rôle du PAV ne se limite pas à afficher une couleur. Il doit aussi communiquer avec le **BOM**, c'est-à-dire le camion poubelle, grâce à un système d'infrarouge. Le BOM doit être capable de lire les données envoyées par le PAV lorsqu'il passe devant. Pour cela, j'ai utilisé la programmation Arduino avec une carte ESP32, ainsi que plusieurs bibliothèques comme **WiFiEsp**, **IRMP** et les outils fournis pour l'ESP32. Ces bibliothèques permettent de gérer la communication réseau, l'envoi de trames infrarouges et l'ensemble des fonctionnalités nécessaires au bon fonctionnement du système.

Ce projet n'a pas commencé de zéro. Il s'agit de la suite directe du travail réalisé par les anciens étudiants de deuxième année. Dans mon cas, j'ai repris la base de code écrite par **Guilhem Bussi**, un étudiant de l'année dernière. Mon travail consistait à comprendre ce qu'il avait mis en place, à corriger certains points qui ne fonctionnaient pas comme prévu, et à apporter des améliorations pour rendre le PAV plus fiable et plus simple à utiliser. Cela m'a demandé du temps, car il fallait analyser le code existant, comprendre sa logique, puis modifier ou compléter certaines parties pour obtenir un résultat cohérent.

Ce projet m'a permis de travailler sur un système embarqué complet, mêlant électronique, communication réseau, gestion d'états, infrarouge et programmation orientée objet. C'est une expérience qui m'a appris à reprendre un travail déjà en cours de développement, et à l'améliorer et à le rendre fonctionnel, ce qui est une compétence importante dans le domaine de l'informatique et de l'électronique..

II - Objectifs et Exigences

II.1 - Objectifs fonctionnels

Gestion de l'infrarouge

Le PAV doit être capable d'envoyer un signal infrarouge au BOM (le bus de collecte). Ce signal contient plusieurs informations importantes, comme la couleur du PAV, son type, et une partie de son adresse IP. Le BOM doit pouvoir lire cette trame lorsqu'il passe devant le PAV.

L'infrarouge est donc un élément essentiel du projet, car il permet la communication entre les équipements sur le terrain.

Le PAV doit se connecter automatiquement au réseau WiFi fourni par LMV. Cette connexion permet au PAV de communiquer avec la supervision, d'envoyer son état en temps réel et de recevoir des ordres. La connexion doit être stable, et en cas de perte de réseau, le PAV doit être capable de se reconnecter automatiquement.

Le programme Arduino doit gérer plusieurs éléments :

- **Lecture des DIP Switch** pour déterminer la couleur et le type du PAV.
- **Gestion de l'infrarouge**, c'est-à-dire la création et l'envoi des trames IR.
- **Communication TCP et WiFi**, pour envoyer des données à la supervision.
- **Gestion des états du PAV** (VIDE, PLEIN, VIDAGE).
- **Affichage LED**, pour indiquer visuellement l'état du PAV.

L'ensemble doit fonctionner de manière fluide, sans blocage, et être capable de tourner en continu.



II.2 - Exigences techniques

Les exigences techniques définissent les contraintes matérielles et logicielles que le PAV doit respecter pour fonctionner correctement. Elles garantissent que le système est compatible avec le reste du projet ESTOM et qu'il peut être utilisé en conditions réelles.

Exigences matérielles

Microcontrôleur ESP32

Le système doit être basé sur un ESP32. Ce choix est justifié par plusieurs raisons :

- il possède un **WiFi intégré**, indispensable pour communiquer avec la supervision
- il offre une **puissance de calcul suffisante** pour gérer l'infrarouge, les LED et la communication réseau ;
- il est compatible avec les bibliothèques utilisées dans le projet, notamment **IRMP** et **ArduinoJson**.

DIP Switch 3 bits

Le dispositif doit intégrer un ensemble de trois interrupteurs permettant :

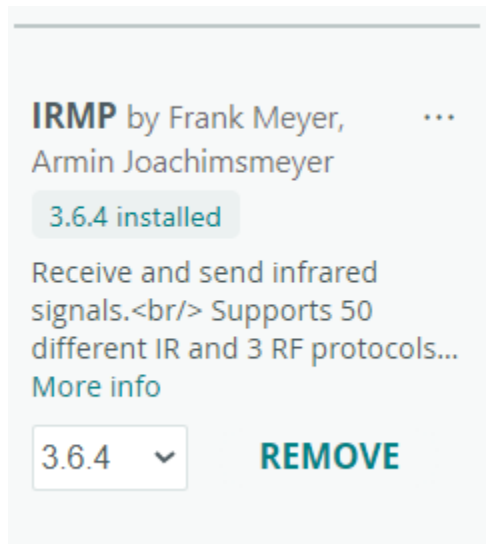
- la sélection de la **couleur du PAV** (3 bits → 4 couleurs : Gris, Jaune, Bleu, Vert),
- la sélection du **type d'équipement** (1 bit → PAV ou Abribus).

Ces interrupteurs permettent une configuration simple et rapide du PAV, sans avoir à modifier le code.

Émetteur infrarouge

Le PAV doit être capable d'émettre un signal infrarouge codé selon le protocole **NEC**. Ce protocole est utilisé car il est simple, fiable et compatible avec la bibliothèque IRMP.

Le BOM doit pouvoir lire ce signal lorsqu'il passe devant le PAV.



III - Planification

II.1 - Outils logiciels

Pour programmer, tester et valider le fonctionnement du PAV, j'ai utilisé plusieurs outils logiciels indispensables. Chacun d'eux m'a servi à une étape précise du développement, que ce soit pour écrire le code, vérifier les trames envoyées ou analyser le comportement du système.

- **L'IDE Arduino**

L'IDE Arduino est l'environnement principal dans lequel j'ai écrit et compilé mon programme en C++. C'est un outil simple à utiliser, qui permet de gérer facilement les bibliothèques, de téléverser le code sur l'ESP32 et d'accéder au moniteur série.

Il m'a permis de structurer mon programme, de tester rapidement des modifications et de corriger les erreurs au fur et à mesure.

La librairie IRMP

La librairie IRMP m'a permis de gérer l'émission infrarouge. Elle prend en charge plusieurs protocoles, dont le protocole **NEC**, qui est celui utilisé dans le projet ESTOM. Grâce à cette librairie, j'ai pu encoder les données du PAV dans une trame IR et l'envoyer de manière fiable au BOM.

Sans IRMP, il aurait fallu gérer manuellement les timings et la modulation, ce qui aurait été beaucoup plus complexe.

Les bibliothèques WiFi.h et WiFiClient.h

Ces deux bibliothèques m'ont servi à connecter l'ESP32 au réseau LMV.

Elles permettent :

- de se connecter automatiquement au WiFi,
- d'ouvrir une connexion TCP vers la supervision,
- d'envoyer des messages JSON,
- et de vérifier en permanence si la connexion est toujours active.

Elles sont essentielles pour que le PAV puisse communiquer avec le serveur et transmettre son état en temps réel.

Le moniteur série

Le moniteur série a été un outil indispensable pendant tout le développement.

Il m'a permis de :

- vérifier les valeurs lues sur les DIP Switch,
- afficher les trames infrarouges envoyées,
- contrôler les messages JSON transmis ou reçus,
- diagnostiquer les problèmes de connexion WiFi ou TCP.

Sans le moniteur série, il serait beaucoup plus difficile de comprendre ce qui se passe réellement dans le programme.

- **ScannaStudio**

ScannaStudio est le logiciel utilisé avec les analyseurs logiques de la marque analogique. Il sert d'interface entre l'ordinateur et l'analyseur, et permet d'observer en temps réel les signaux numériques capturés sur les broches d'un montage électronique. Le logiciel affiche les signaux sous forme d'ondes temporelles, ce qui permet de vérifier la forme des impulsions, la durée des bits, la fréquence du signal et l'ordre exact dans lequel les données sont envoyées. C'est un outil indispensable lorsqu'on travaille avec des protocoles comme l'infrarouge, car il permet de visualiser précisément ce que la LED IR émet réellement.

Le logiciel fonctionne avec les analyseurs logiques analogique de la série SQL, qui se branchent en USB sur l'ordinateur. Une fois connecté, ScanaStudio permet de choisir les canaux à analyser, de régler la fréquence d'échantillonnage, de définir un déclencheur (trigger) et de lancer une acquisition. Les signaux apparaissent ensuite sous forme de courbes numériques (0/1), ce qui permet de zoomer, mesurer des durées, vérifier les fronts montants et descendants, et analyser la chronologie exacte d'un message. Le logiciel propose aussi un système de "workspaces" qui permet de sauvegarder les réglages et les captures pour les réutiliser plus tard.

L'un des points les plus utiles de ScanaStudio est sa capacité à **décoder automatiquement des protocoles de communication**. Pour l'infrarouge, il suffit d'ajouter un décodeur NEC dans le logiciel : ScanaStudio analyse alors la trame capturée et affiche directement les valeurs décodées (adresse, commande, bits, etc.). Cela permet de vérifier très rapidement si le PAV envoie bien la bonne valeur, si la structure du message est correcte et si le BOM reçoit une trame conforme.

II.2 - Tâches planifiées

Pour organiser mon travail, j'ai découpé le développement en plusieurs tâches. Cela m'a permis d'avancer rigoureusement cela va de sois, et de vérifier chaque partie avant de passer à la suivante.

Lecture et gestion des DIP Switch

Cette partie consistait à :

- identifier les broches de l'ESP32 à utiliser,
- programmer la lecture des trois DIP Switch,
- déterminer la couleur du PAV à partir des deux premiers bits,
- déterminer le type (PAV ou Abribus) à partir du troisième bit,
- vérifier le bon fonctionnement via le moniteur série.

Cette étape était importante, car la configuration du PAV dépend entièrement de ces interrupteurs.

II.3 - Horaires de travail

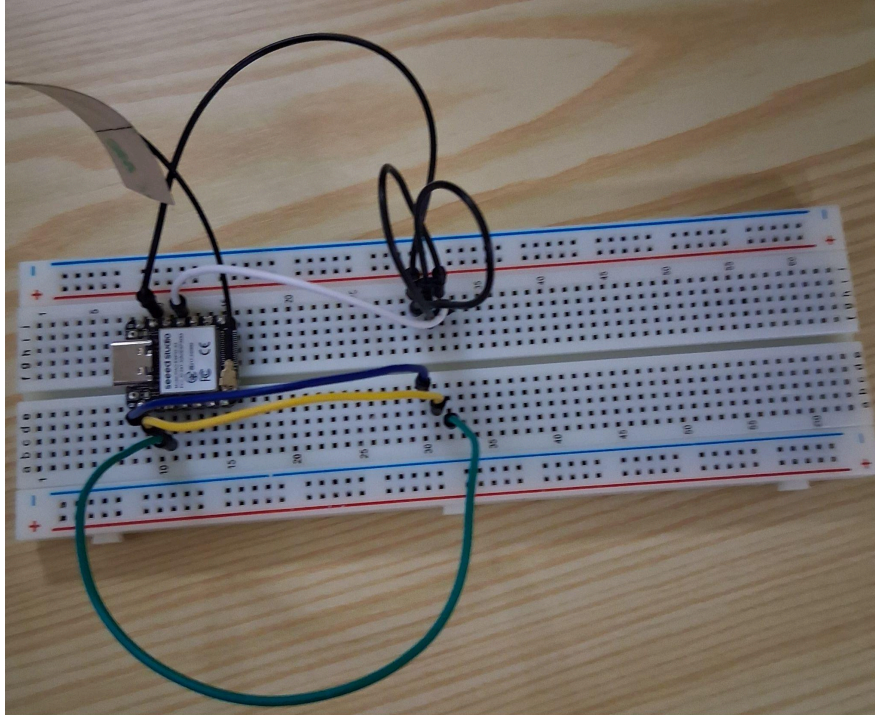
Pour planifier mon travail, j'ai utilisé **Microsoft Project**.

J'ai d'abord créé une estimation globale du déroulement du projet, puis j'ai mis à jour cette planification chaque semaine en fonction de l'avancement réel.

Cela m'a permis de :

- visualiser les tâches déjà réalisées,
- identifier celles qui restaient à faire,
- ajuster mon planning en cas de retard ou de difficulté,
- garder une vue d'ensemble sur le projet.

Nom de la tâche ▼	Durée ▼	Début ▼	Fin ▼
▣ Projet ESTOM	184 h?	Mar 06/01/26	Sam 30/05/26
▣ Commun	13 h?	Mar 06/01/26	<u>Mer 07/01/26</u>
Etude du dossier	4 h	Mar 06/01/26	Mar 06/01/26
Réunion de cadrage	4 h	Mar 06/01/26	Mar 06/01/26
Réunion pour la partie Physique	1 h	Mer 07/01/26	Mer 07/01/26
Etude du rapport 2025	4 h	Mar 06/01/26	Mar 06/01/26
Test du Projet en situation réel	4 h	Mer 01/04/26	<u>Mer 01/04/26</u>
Projet ESTOM devant les clients et public	9 h	Sam 30/05/26	<u>Sam 30/05/26</u>
▣ Activité de Soheybe	76 h	Mar 06/01/26	Mer 04/03/26
Etude du Programme TCP	4 h	Mar 06/01/26	Mar 06/01/26
Etude de la partie Physique	4 h	Mer 07/01/26	Mer 07/01/26
Connecter au Wifi et au Serveur TCP	4 h	Lun 12/01/26	Lun 12/01/26
Pris en main du code DIP Switch	2 h	Mer 14/01/26	Mer 14/01/26
TEMO DIP Switch	2 h	Lun 19/01/26	Lun 19/01/26
Constrution de l'infrarouge	12 h	Lun 19/01/26	Mer 21/01/26
Le code Complet	18 h	Mer 11/02/26	Mer 04/03/26



IV - Présentation de ma partie

II.1 - Généralités

Dans le cadre du projet ESTOM, ma mission principale a été de développer la partie embarquée du PAV (Point d'Apport Volontaire). L'objectif était de rendre ce module capable d'envoyer des informations fiables à la supervision, tout en intégrant un système d'identification basé sur des DIP Switch et une communication infrarouge destinée au BOM (Benne à Ordures Ménagères).

Mon travail s'est donc concentré sur trois éléments essentiels :

- **La lecture des DIP Switch**, qui permettent de définir la couleur du PAV (Gris, Jaune, Bleu ou Vert) ainsi que son type (PAV ou Abribus).
- **L'émission infrarouge**, utilisée pour transmettre un code unique basé sur la configuration du PAV. Ce code doit être lu par le BOM lorsqu'il passe devant.
- **La communication WiFi/TCP**, qui permet d'envoyer à la supervision l'état du PAV (VIDE, PLEIN, VIDAGE) et de recevoir des ordres si nécessaire.

Pour structurer mon programme, j'ai choisi une approche modulaire. J'ai donc utilisé plusieurs classes dédiées : une pour la lecture des DIP, une pour la gestion des LED,

une pour l'infrarouge, et une pour la batterie. Le fichier principal `.ino` sert à coordonner l'ensemble et à gérer la logique générale du PAV.

L'objectif final était d'obtenir un système autonome, capable de fonctionner en continu, de se reconnecter automatiquement au réseau en cas de coupure, et d'envoyer des informations fiables au superviseur sans intervention humaine.

IV.1 - Architecture technique

L'architecture technique de ma partie repose sur trois blocs fonctionnels principaux :

1. l'identification via les DIP Switch,
2. la communication infrarouge,
3. la communication réseau (WiFi + TCP).

1. Lecture des DIP Switch

Les DIP Switch sont connectés sur trois broches GPIO configurées en entrée avec la résistance interne `INPUT_PULLUP`.

Chaque interrupteur représente un bit :

- DIP 1 → poids 1
- DIP 2 → poids 2
- DIP 3 → poids 4

En combinant ces trois bits, le PAV peut générer un identifiant unique compris entre 0 et 7. L'identifiant est ensuite utilisé pour calculer la valeur envoyée en infrarouge.

La lecture des DIP se fait au démarrage du PAV. Cela permet d'éviter les variations intempestives et garantit que la configuration reste stable pendant tout le fonctionnement.

2. Émission infrarouge

L'émission infrarouge utilise la librairie `IRMP`, configurée pour fonctionner avec le protocole **NEC**, un protocole simple et largement utilisé dans les systèmes IR.

Le fonctionnement est le suivant :

- le PAV lit les DIP,
- il calcule la valeur à envoyer,
- il encode cette valeur dans une trame NEC,

- il envoie la trame via l'émetteur IR,
- il affiche les informations dans le moniteur série.

Ce processus se répète régulièrement, notamment lorsque le PAV est en état PLEIN ou VIDAGE, car ce sont les moments où le BOM doit recevoir l'information.

3. Communication WiFi et TCP

Le PAV se connecte automatiquement au réseau WiFi créé par la Raspberry utilisé. Une fois connecté, il ouvre une socket TCP vers la supervision.

Dès le démarrage, il envoie un message "BONJOUR" pour signaler qu'il est en ligne.

Ensuite, à chaque changement d'état, il envoie un message JSON contenant :

- la couleur du PAV,
- son type (PAV ou Abribus),
- son état (VIDE, PLEIN, VIDAGE).

L'ESP32 gère également la reconnexion automatique en cas de perte de réseau. Cela garantit que le PAV reste toujours visible par la supervision.

IV.3 - Sécurité et fiabilité

La partie PAV que j'ai développée doit fonctionner en continu, sans intervention humaine. Pour garantir cela, j'ai intégré plusieurs mécanismes de sécurité et de fiabilité dans le programme. Ils concernent le réseau, les données échangées, l'infrarouge et le fonctionnement général de l'ESP32.

Sécurisation de la connexion réseau

La communication WiFi et TCP est essentielle pour que le PAV puisse dialoguer avec la supervision. Pour éviter les blocages ou les pertes d'informations, plusieurs protections ont été mises en place :

- **Reconnexion automatique au WiFi** : si l'ESP32 ne parvient pas à se connecter dans un délai de 5 secondes, il redémarre automatiquement.
- **Vérification permanente de la connexion TCP** : si la connexion au serveur est perdue, l'ESP32 redémarre immédiatement pour rétablir la communication.
- **Envoi d'un message "BONJOUR" au démarrage** : cela permet au serveur de savoir que le PAV est bien en ligne.

Ces mécanismes assurent que le PAV reste toujours connecté et visible par la supervision.

Fiabilité des données JSON

Les échanges entre le PAV et le serveur se font en JSON. Pour éviter les erreurs :

- chaque trame reçue est vérifiée (présence de la clé “Ordre”),
- les erreurs de désérialisation sont ignorées pour éviter les blocages,
- La structure JSON envoyée reste toujours la même (Type, Couleur, Progression, NbrCollision, Status).

Ces choix garantissent que les données échangées sont cohérentes et exploitables.

Fiabilité de l'émission infrarouge

L'infrarouge transmet un identifiant unique basé sur :

- le type (PAV ou Abribus),
- la couleur,
- la configuration des DIP.

Pour assurer une émission fiable :

- utilisation du protocole NEC,
- encodage compact sur 8 bits,
- émission uniquement dans les états critiques (PLEIN, VIDAGE),
- gestion automatique des timings par la librairie IRMP.

Cela garantit que le message IR est toujours conforme et correctement envoyé.

Robustesse du fonctionnement général

Le programme est conçu pour fonctionner 24h/24 sans intervention :

- utilisation de classes dédiées (CIR, DipSwitch, etc.),
- affichage systématique dans le moniteur série,
- gestion propre des états,
- redémarrage automatique en cas de problème réseau.

Résistance aux erreurs et aux coupures

Pour éviter que le PAV devienne inutilisable :

- redémarrage automatique en cas de perte réseau,
- lecture des DIP uniquement au démarrage,
- LED comme indicateur visuel d'état,
- aucune dépendance à des ressources externes instables.

Le PAV reste donc opérationnel même en cas de perturbations réseau ou électriques.

V - Déroulement du travail

V.0 - Partie physique: l'infrarouge

L'infrarouge (IR) est une technologie de communication qui utilise de la lumière invisible à l'œil humain pour transmettre des informations. Cette lumière se situe juste en dessous du spectre visible, autour de **940 nm** (nanomètres), ce qui la rend totalement invisible mais détectable par des capteurs électroniques.

Dans le projet ESTOM, l'infrarouge sert à transmettre un **identifiant unique** du PAV vers le BOM. Le BOM lit ce signal lorsqu'il passe devant le PAV, ce qui lui permet de savoir quel conteneur il est en train de collecter.

1. À quoi sert l'infrarouge dans le projet ?

L'infrarouge est utilisé pour une raison simple :

c'est un moyen de communication direct, rapide et fiable entre le PAV et le BOM.

Il permet :

- d'envoyer un identifiant unique du PAV,
- d'indiquer la couleur du conteneur (Gris, Jaune, Bleu, Vert),
- de signaler le type (PAV ou Abribus),
- de transmettre une information sans fil, sans WiFi, sans Bluetooth,
- et surtout **sans risque d'interférences radio**.

Le BOM n'a qu'à passer devant le PAV pour recevoir la trame IR.

2. Comment fonctionne un signal infrarouge ?

Pour transmettre une information, l'infrarouge utilise une LED spéciale qui émet une lumière invisible.

Cette LED ne reste pas allumée en continu : elle **clignote très rapidement** selon un motif précis.

2.1 Conversion de l'information

Tout commence par une donnée numérique :

un identifiant, une couleur, un type, etc.

Cette donnée est convertie en **binaire** (suite de 0 et de 1).

2.2 Modulation du signal

Chaque bit (0 ou 1) est représenté par un comportement de la LED :

- un **1** = la LED clignote à une fréquence,
- un **0** = la LED reste éteinte ou clignote différemment.

C'est la fréquence standard utilisée par la plupart des télécommandes et capteurs IR.

2.3 Le protocole utilisé : NEC

Le PAV utilise le protocole **NEC**, un protocole très répandu dans les télécommandes.

Il définit :

- la durée d'un bit,
- la durée d'un "1",
- la durée d'un "0",
- la structure du message,
- le temps entre deux trames.

Durée d'une trame NEC

Une trame NEC complète dure environ **67 ms** (millisecondes).

3.

Le PAV envoie un **octet (8 bits)** construit à partir :

- de la couleur (2 bits),
- du type (1 bit),
- de l'identifiant DIP (3 bits),
- et d'un complément pour compléter l'octet.

Ce code est ensuite encodé en NEC et envoyé via la LED IR.

Le BOM possède un **récepteur infrarouge**.

Ce récepteur détecte :

- la lumière IR,
- la fréquence (38 kHz),
- les variations (0 et 1),
- la structure NEC.

Le microcontrôleur du BOM :

1. récupère la suite binaire,
2. vérifie la structure NEC,
3. décode les bits,
4. reconstruit l'identifiant du PAV,
5. associe cet identifiant à une couleur et un type.

C'est comme une télécommande et une télévision :

Le PAV est la télécommande, le BOM est la TV.

L'infrarouge a plusieurs limites :

- il est sensible à la lumière du soleil,
- il nécessite un alignement approximatif,
- la portée dépend de la luminosité,
- les obstacles bloquent le signal.

C'est pour ça que, lors des tests à Cavaillon, il fallait parfois repasser devant le PAV pour que le BOM reçoive la trame.

- L'infrarouge = lumière invisible à **940 nm**.
- Fréquence de modulation = **38 kHz**.
- Protocole utilisé = **NEC**.
- Durée d'une trame ≈ **67 ms**.
- Données envoyées = **identifiant du PAV (8 bits)**.
- Rôle = permettre au BOM d'identifier le PAV lors du passage.
- Limites = sensibilité à la lumière, portée variable.

V.1 - Le travail réalisé

Dans le code de l'infrarouge

```
void connectToWiFi() {
    Serial.print("Connexion au WiFi");
    WiFi.begin(g_ssid, g_password);
    unsigned long startAttemptTime = millis();
    while ( (WiFi.status() != WL_CONNECTED) && (
(millis()-startAttemptTime)<10000) ) {
        delay(500);
        Serial.print(".");
    } //wh

    if (WiFi.status() == WL_CONNECTED) {
        Serial.println("\nConnecté au WiFi");
        Serial.print("Adresse IP locale : ");
        Serial.println(WiFi.localIP());
        adrIpLocale = WiFi.localIP();
        g_lastOctet = adrIpLocale[3]; // dernier octet
        Serial.print("\ndernier octet :");
        Serial.println(g_lastOctet);
    } else {
        Serial.println("\nÉchec de connexion WiFi. RESTART !");
        ESP.restart(); // redémarrage logiciel

    } // else

} // fonction
```

Connexion WiFi : Le programme essaie de se connecter au réseau WiFi en utilisant le SSID et le mot de passe fournis.

Boucle d'attente : Pendant **10 secondes maximum**, il vérifie toutes les 500 ms si la connexion est établie.

Pendant ce temps, il affiche des points . pour montrer qu'il attend.

Connexion réussie :

Si le WiFi se connecte :

- Il affiche un message de succès.
- Il montre l'**adresse IP locale** obtenue.
- Il récupère le **dernier octet** de cette adresse IP (souvent utile pour identifier l'appareil sur le réseau).

Échec de connexion —

Si après 10 secondes il n'est toujours pas connecté :

- Il affiche un message d'erreur.
- Il **redémarre automatiquement l'ESP** pour réessayer.

Dans le code de l'infrarouge

Lors de l'exécution de la fonction `setup()`, le PAV réalise plusieurs étapes essentielles pour être opérationnel :

- connexion automatique au réseau WiFi LMV,
- ouverture d'une connexion TCP vers la supervision,
- lecture des DIP Switch pour identifier la couleur et le type du PAV,
- initialisation du bandeau LED,
- initialisation du module infrarouge,
- envoi d'un message "BONJOUR" au serveur pour signaler que le PAV est en ligne.

Cette séquence garantit que tout est correctement configuré avant que le PAV commence son fonctionnement normal.

Tests réalisés à Cavaillon

Nous sommes ensuite partis à Cavaillon pour effectuer des tests directement chez LMV.

Ces tests étaient importants, car ils permettaient de voir comment le PAV réagit dans des conditions réelles, et pas seulement dans la salle de TP.

Sur place, on avait :

- le vrai BOM,
- les distances réelles,
- la lumière du soleil,
- les perturbations extérieures impossibles à simuler en classe.

Ces tests nous ont permis de confirmer que le système fonctionnait, mais aussi d'identifier plusieurs points à améliorer.

Résultats des tests

L'infrarouge fonctionnait globalement bien :

Le BOM arrivait à recevoir la trame, mais **pas toujours du premier coup**.

Parfois, il fallait :

- repasser une deuxième fois devant le PAV,
- ou se rapprocher un peu plus.

C'est normal, car l'infrarouge est très sensible à la lumière ambiante, et à Cavaillon il faisait particulièrement clair.

Malgré cela, les tests ont été concluants : le système fonctionnait, mais nécessitait quelques ajustements pour être parfaitement fiable.

Améliorations apportées après les tests

Suite aux tests, j'ai décidé d'apporter plusieurs mises à jour au PAV, notamment au niveau du bandeau LED.

Avant, lorsque le PAV détectait le BOM, le bandeau clignotait en continu, ce qui donnait un effet peu clair, presque comme un bug visuel.

J'ai donc modifié l'animation pour qu'elle soit :

- plus propre,
- plus visible,
- plus cohérente avec l'état du PAV.

Ces modifications rendent le comportement du PAV plus lisible.

VI - Conclusion

Pour conclure, ce projet m'a vraiment permis de voir ce que c'est que de travailler sur un système embarqué complet, du début à la fin. J'ai dû comprendre comment tout s'enchaînait, comment chaque partie communiquait avec les autres, et surtout comment faire en sorte que le PAV fonctionne de manière fiable, que ce soit en infrarouge ou en réseau. Il y a eu des moments où tout marchait du premier coup, et d'autres où il fallait chercher, tester, corriger, recommencer... mais c'est justement ça qui m'a fait progresser.

La partie infrarouge m'a demandé pas mal de temps, surtout pour bien comprendre comment la trame était envoyée et comment le BOM la recevait. Les tests à Cavaillon ont été importants, parce qu'on a pu voir le comportement du PAV en conditions réelles, avec la lumière, la distance, et tout ce qu'on ne peut pas simuler en salle. Ça a montré que le système fonctionnait, même si parfois il fallait ajuster ou repasser devant pour que le BOM capte bien le signal.

J'ai aussi travaillé sur la communication réseau, les messages JSON, la gestion des DIP et tout ce qui permet au PAV de tourner tout seul sans intervention. J'ai ajouté des sécurités pour éviter les blocages, des vérifications pour garder une connexion stable, et des améliorations pour rendre le fonctionnement plus propre et plus clair.

Nous avons pu faire une démonstration du projet ESTOM à Cavaillon devant des touristes qui jouaient à notre stand. Ces derniers étaient plutôt heureux lorsqu'ils essayaient notre projet ESTOM, même s'il y a eu un problème avec l'infrarouge, qui était trop faible, et que les BOM devaient se coller au PAV pour être identifiés.

Au final, même si tout n'a pas été simple, le PAV fait ce qu'on attend de lui : il envoie son identifiant en infrarouge, il communique avec la supervision, il gère ses états et il reste stable dans le temps. Ce projet m'a appris à être plus autonome, à mieux organiser mon travail et à comprendre comment un vrai système embarqué se construit et s'améliore.

Partie Personnelle IR3: Supervision

HACENE Fatma

I - Introduction personnelle

Pour le projet de 2ème année de BTS CIEL, j'ai choisi de travailler sur le projet ESTOM (Espace de Sensibilisation au Tri des Ordures Ménagères). J'ai fait un stage chez Luberon Monts de Vaucluse agglomération donc c'était un choix impératif pour moi de choisir ESTOM.

Ma mission est de s'occuper de la Supervision, le cœur du projet. Elle consiste d'une interface Web, d'un serveur TCP et d'une base de données. Ce superviseur doit communiquer avec les BOM (Benne à Ordures Ménagères) et les PAV (Point d'Apport Volontaires). Tout cela est piloté grâce à une Raspberry Pi.

Puisque le projet qui n'a pas été terminé, j'ai récupéré le travail du précédent IR3, Clément VOLLE, et je complète ou change le travail déjà fait en se basant sur le cahier des charges final.

II - Objectifs et Exigences

II.1 - Objectifs fonctionnels

Voici la partie du diagramme des exigences qui me concerne:

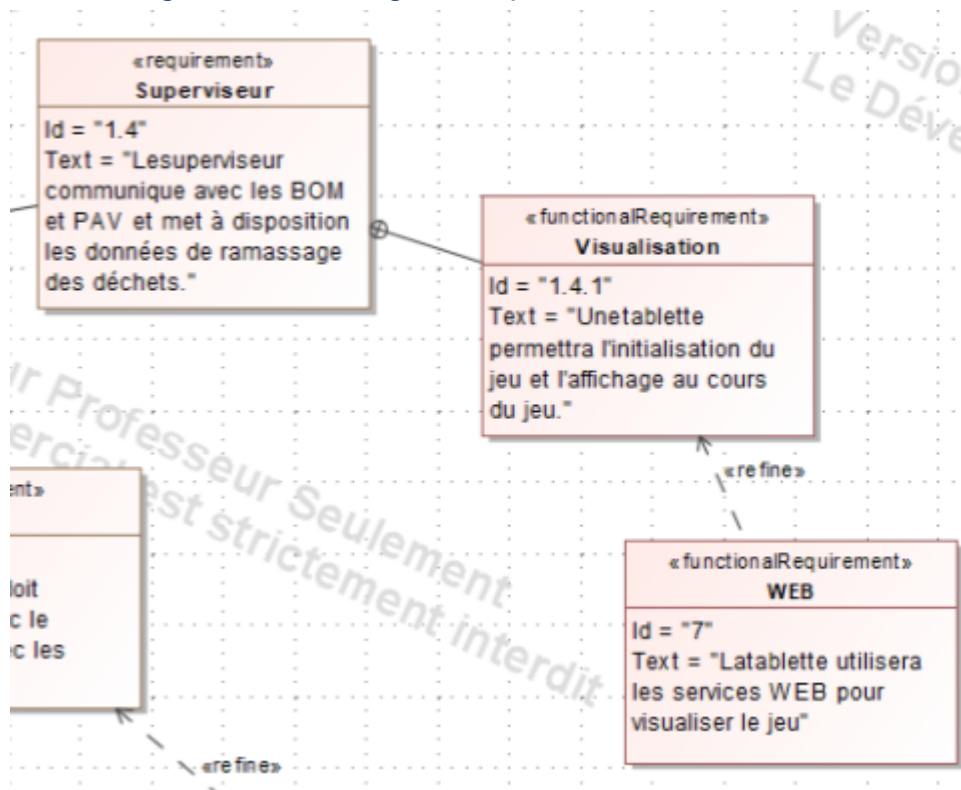


Diagramme des exigences reqESTOM (réduit)

Ceci présente en tout les deux grandes parties que je dois traiter pour le côté supervision de ce système.

L'interface Web de configuration:

Pour contrôler le jeu, une interface Web graphique est fournie par la Raspberry Pi. Elle permet de:

Paramètrer:

- Définir le nombre de PAV par couleur
- Prévoir combien de BOM sont présent pour la partie,
- Configurer les points mais aussi les malus de collision, de 0.2 à 5 points.

Afficher:

- L'écran de la partie en cours, avec le temps imparti
- Les vainqueurs de la partie

Contrôler:

- Lancer la partie
- Réinitialiser la partie

Le serveur TCP:

Un serveur TCP est impératif pour la communication entre les BOM et les PAV pour contrôler la partie. Elle envoie des ordres, chacune numérotée en fonction de leur contenu. Elle reçoit aussi des réponses des BOM et PAV.

Ordres envoyés:

- | | |
|---------------------------|-------|
| • 0 = Initialiser | - ALL |
| • 1 = Départ de la partie | - ALL |
| • 2 = Fin de la partie | - ALL |
| • 10 = Autoriser transfer | - PAV |
| • 11 = Fin transfer | - PAV |
| • 12 = Annuler transfer | - PAV |

Messages reçues:

- | | |
|------------------------|-------|
| • 0 = Information | - ALL |
| • 1 = Détection | - BOM |
| • 2 = Fin transfer | - BOM |
| • 3 = Annuler transfer | - BOM |
| • 4 = Collision | - BOM |

II.2 -Exigences techniques

Sécurité:

L'accès Wi-Fi est protégé avec WPA2-PSK et contient un pare-feu configuré pour bloquer tous les ports inutiles sauf pour les ports 22 pour accès à distance, 5005 pour le serveur TCP et 3306 pour la base de données.

Seulement 21 équipements peuvent se connecter à l'accès WiFi pour ne pas autoriser d'autres appareils inconnus à se connecter.

L'interface Web contient un certificat SSL et fonctionne avec le protocole HTTPS.

Performance:

Temps de communication suffisamment rapide pour un jeu fluide et sans attente remarquable.

Fiabilité:

Affichage continu de la partie, limiter les bugs visuels possible. Vérification d'une réception certaine des données. S'assurer de la bonne communication entre le serveur TCP et l'interface Web avec la base de données.

Compatibilité:

Grâce à l'implémentation d'une page configuration en tant que site Web, elle peut être accédée par les navigateurs Chromium et Firefox. Elle doit aussi être (web/device responsive) pour s'afficher correctement sur un grand écran ou une tablette.

Documentation:

Un manuel utilisateur va être fourni avec le système pour que le client puisse connaître comment ce système fonctionne et diagnostiquer les problèmes éventuels qu'on peut prévoir. Écrire des fiches recettes complètes pour vérifier le bon fonctionnement du système.

III - Planification

III.1 - Outils et méthodes

Les outils:

Pour compléter la mission, j'ai utilisé quelques outils à ma disposition

Serveur TCP:

- Qt Creator est l'environnement de développement intégré utilisé pour créer l'application qui lance le serveur TCP. Cet outil de développement est pour le C++ et disponible sur Raspbian, le système d'exploitation de la Raspberry Pi.

Interface Web:

- WinSCP et son éditeur de texte intégré servent à modifier les pages .php du site. La Raspberry Pi n'est pas assez puissante pour modifier les fichiers nativement. Ceci amène un confort pour tester les pages sans perdre trop de temps à attendre la réponse de la Raspberry Pi.
- phpMyAdmin est une interface de configuration Web qui fait partie du paquet LAMP. Elle sert à afficher ou même modifier la base de données.

Documentation:

- Project est un outil créé par Microsoft. Elle sert à planifier les tâches dans le temps pour avoir une idée du travail à faire.
- MagicDraw est un programme qui sert à créer des diagrammes UML (Unified Modeling Language) SysML (System Modeling Language). On s'en sert pour créer les diagrammes pour voir d'une manière visuelle comment le système va fonctionner en théorie.

La méthode:

Comme le précédent IR3, je vais aussi procéder en 3 itérations mais avec des priorités un peu différentes.

Partie 1: Mise en place

L'installation des outils LAMP (Linux, Apache2, MySQL/MariaDB, PHP) pour mettre en route le serveur Web, suivi d'une mise en place de l'accès Wi-Fi avec dhcpcd, hostapd et dnsmasq puis cloner le répertoire appEstom de Github à la Raspberry Pi. Vérification que tout a été bien installé et le bon fonctionnement du travail de l'an dernier et installation de la version Debian de Qt Creator. Mise en place de MariaDB (MySQL) et de phpMyAdmin pour la base de données impérative pour le bon fonctionnement du système.

Partie 2: Le serveur TCP et le programme Qt/C++

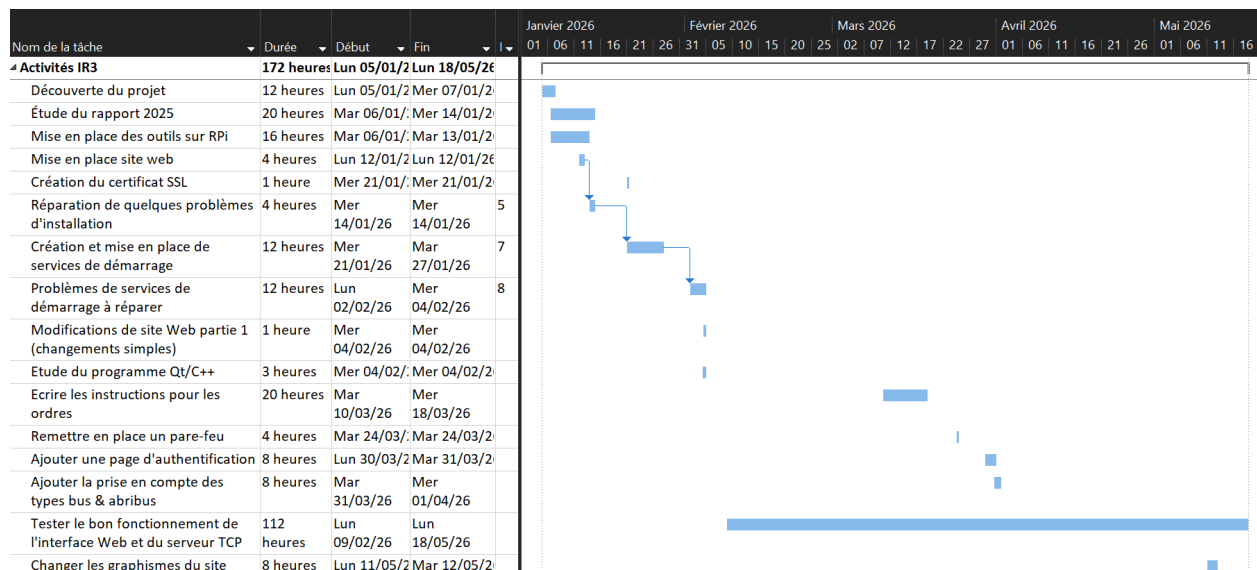
Récupération du programme Qt/C++ simulateur qui était pour tester le bon fonctionnement du jeu. La conversion de ce programme de simulateur à un vrai serveur avec la construction des trames JSON à envoyer au BOM et au PAV.

Partie 3: L'interface Web

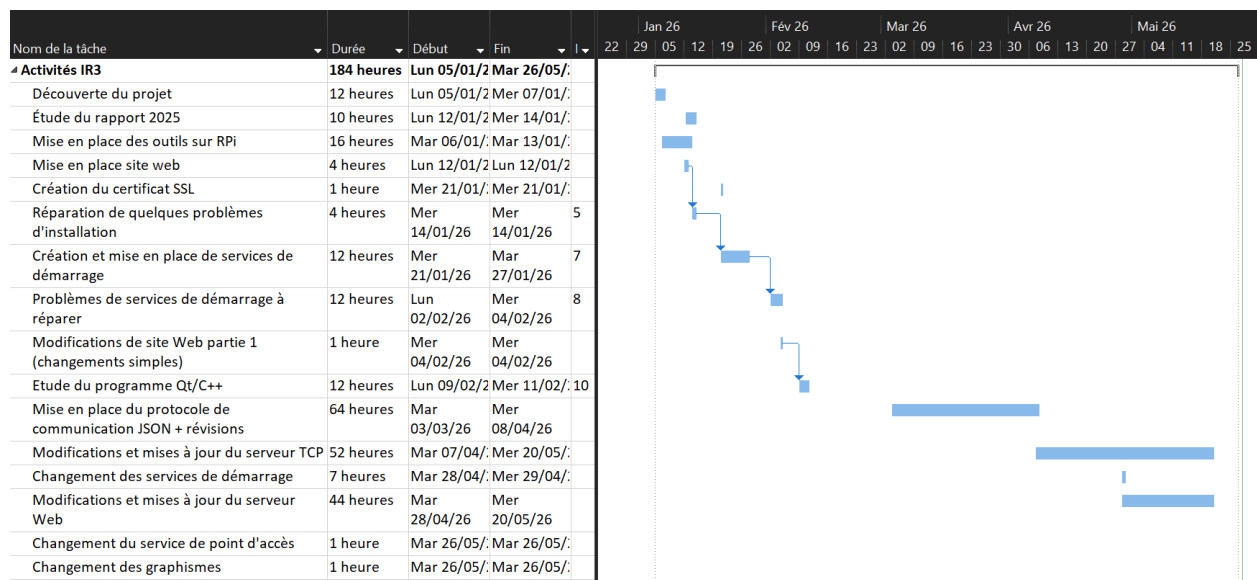
Récupération du frontend et backend du site en changeant les termes (l'ancien terme pour le PAV qui était TOM (Tour à Ordures Ménagères), sécurisation du site grâce au protocole HTTPS et un mot de passe défini dans la base de données.

III.2 - Tâches planifiées

Planification prévisionnelle:



Planification réelle:



IV - Présentation de ma partie

IV.1 - Généralités

Ma mission se porte sur la supervision, l'utilisation d'une interface Web pour contrôler et visualiser le jeu mais aussi pouvoir communiquer avec les BOM/BUS et les PAV/ABRIBUS et au final, montrer le nombre total des points. Ceci consiste de 3 parties:

- **Le serveur TCP écrit en Qt/C++** qui gère la communication entre les véhicules et leur (goals) en utilisant les classes CDBReader et TCPListener pour pouvoir se connecter en utilisant le port 5005. Elle est fournie à un accès à la base de données par un utilisateur spécifique avec le port 3306..
- **La base de données MySQL** contient toutes les données comme les configurations du jeu par une table **Config**, et les données des véhicules et (bruh) sous les tables **BOM** pour les BOM et le bus et **PAV** pour les PAV et les abribus. Elle est accessible par un administrateur
- **L'interface Web** graphique accessible par le point d'accès Wi-Fi de la Raspberry Pi. Elle est sécurisée par le protocole HTTPS avec un certificat auto-signé. Son adresse est 'estom.fr'. Elle est munie d'une page pour afficher et contrôler la partie mais aussi pour configurer le jeu. Elle utilise la base de données pour pouvoir configurer mais aussi connaître les états des BOM et PAV.

IV.2 - Architecture technique

- **Frontend:** PHP avec des requêtes SQL, JavaScript (AJAX) pour un rafraîchissement rapide
- **Backend:** Raspbian avec le paquet LAMP (Apache2, PHP, MariaDB/MySQL), un point d'accès Wi-Fi WPA2-PSK
- **Base de données:** contenant que 3 tables (Config, BOM et PAV) donc la table Config qui doit impérativement contenir une ligne pour le fonctionnement du serveur TCP et de l'interface Web
- **Intégration temps réel:** la classe CDBReader de Qt Creator effectue un polling, c'est à dire de vérifier l'état de la base de données, à chaque seconde afin de mettre à jour la variable status (qui est l'état du jeu)

IV.3 - Sécurité et fiabilité

Le point d'accès Wi-Fi est protégé par le protocole WPA2-PSK et contiendra un pare-feu iptables pour limiter le nombre d'appareils qui peuvent se connecter mais aussi bloquer les ports inutiles.

Pour les requêtes SQL, la validation côté serveur des requêtes par PHP pour protéger le backend des injections SQL.

V - Déroulement du travail

V.0 - Partie physique: le Wi-Fi

C'est quoi le Wi-Fi?

Le Wireless Fidelity, connu en tant que Wi-Fi, est utilisé pour la communication entre des appareils dans un réseau local de façon **sans fil**. Pour accéder au réseau Wi-Fi, un **point d'accès** (access point) est important pour être le relais entre les appareils mais aussi le réseau câblé Ethernet. Il fonctionne grâce à **la propagation des ondes électromagnétiques**.

Le Wi-Fi utilise les **ondes radioélectriques**, aussi connues en tant qu'**ondes hertziennes**, qui sont caractérisées par leur fréquence, longueur d'onde et puissance. Les ondes sont émises par une antenne Wi-Fi et se propagent, pouvant être absorbées par certains matériaux comme des murs ou avoir des interférences avec d'autres signaux comme le Bluetooth.

Le WiFi utilise quelques bandes de fréquences principalement:

- **2.4GHz**, meilleur pour les grandes distances mais un débit lent, utilisé pour des tâches simples et a une compatibilité universelle
- **5GHz**, meilleur débit mais sa portée est réduite, un compromis entre la distance et la performance
- **6GHz**, le Wi-Fi 6E, la meilleur performance possible mais avec une portée plus réduite et n'est pas pris en charge par quelques anciens appareils

Étude du point d'accès

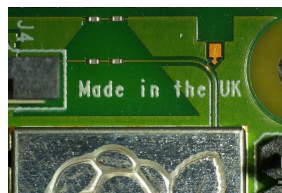
On utilise une **Raspberry Pi 3 modèle B+**. Elle contient une carte Wi-Fi à **double bande** donc elle utilise les fréquences 2.4GHz et 5GHz. Dans le cadre de notre travail, **on utilise la bande 2.4GHz avec la norme IEEE 802.11n**. La Raspberry Pi peut théoriquement atteindre jusqu'à 150Mbit/s.

Donc pour calculer sa longueur d'onde, on divise la célérité avec la fréquence:

$$\lambda = c/f = 3 \cdot 10^8 / 2.4 \cdot 10^9 \approx 0.125m \text{ soit } 12.5cm$$

Donc sa longueur d'onde est de 12.5cm.

L'antenne de la Raspberry Pi est intégrée dans la carte, comme on le voit ici:



Le triangle que vous voyez est l'antenne intégrée provenant de ProAnt AB.

Elle a une puissance d'émission de 16 dBm qu'on peut convertir en Watts:

$$P_w = 10^{16/10} \approx 39.81 \text{ mW} \text{ arrondi à } 40 \text{ mW}$$

On va aussi travailler avec un gain typique de 2dBi.

Puissance Isotrope Rayonnée Équivalente

La Puissance Isotrope Rayonnée Équivalente, ou le PIRE, sert à montrer la puissance du système si l'antenne est parfaitement isotrope (qui signifie que l'antenne émet dans toutes les directions de manière égale).

Le PIRE s'exprime en dBm. Voici le calcul:

$$\text{PIRE} = \text{Puissance d'émission } P_e + \text{Gain de l'antenne } G$$

Donc avec $P_e = 16 \text{ dBm}$ et $G = 2 \text{ dBi}$:

$$\text{PIRE} = 16 \text{ dBm} + 2 \text{ dBi} = 18 \text{ dBm}$$

$$P_w = 10^{18/10} \approx 63.09 \text{ mW} \text{ arrondi à } 63 \text{ mW}$$

La réglementation européenne dit que le PIRE doit être en dessous de 100mW, soit 20dBm. Grâce au calcul du PIRE, on peut donc observer qu'on est bien en dessous de la limite.

L'estimation de la portée du signal

Pour estimer la distance maximale de communications, on utilise l'équation de Friis:

$$P_r = P_t + G_t + G_r - 20\log(d) - 20\log(f) - 32.45$$

avec

P_r : la puissance reçue, on choisit -80dBm car c'est le seuil pour un bon signal

P_t : la puissance transmise, donc la puissance de 16dBm

G_t & G_r : les gains des antennes, donc notre choix de 2dBi

f : la fréquence en MHz, donc 2.4 devient 2400

d : la distance en km

On cherche d donc on retourne l'équation de Friis:

$$20\log(d) = -P_r + P_t + G_t + G_r - 20\log(f) - 32.45$$

Pour annuler le log, on utilise $10^{}$ ce qui nous donne:

$$d = 10^{80+16+2+2-20\log(2400)-32.45/20} \approx 0.994 \text{ km}$$

La distance maximale de la portée du signal dans un espace libre est d'environ 1km.

V.1 - Partie 1: Mise en place

Préparation de la Raspberry Pi et configuration

Pour commencer, j'ai pris une Raspberry Pi 3 Model B+ où j'ai installé Raspbian Bookworm. Ceci sera mon environnement de travail principal. J'ai activé le SSH et le

VNC pour utiliser la Raspberry Pi en mode headless (c'est-à-dire, sans brancher un écran et un clavier directement dessus).

Installation et récupération du travail du précédent IR3

Le travail du précédent IR3 réside dans un répertoire Github sous le nom de ESTOM. Pour le récupérer, il suffit de taper:

```
git clone https://github.com/PhASnBenoit/ESTOM/
```

Ce répertoire contient l'interface Web, l'application Qt/C++ pour le serveur TCP et la base de données ESTOM.

Installation des outils

Pour installer les outils sur les systèmes basés sur Debian comme Raspbian, on doit passer par le package manager apt. Avant de procéder à l'installation, il faut mettre à jour la liste de répertoires de apt avec

```
sudo apt update ou sudo apt upgrade
```

Dès que apt est à jour, on peut commencer à installer les paquets qu'ils nous faut. On installe le paquet LAMP avec

```
sudo apt install apache2 mariadb php -y
```

L'argument **-y** indique à apt à installer directement sans demander notre avis sur la taille qu'ils vont prendre.

Puis on installe Qt Creator avec

```
sudo apt install qtcreator -y
```

Dans mon cas particulier, je n'avais ni hostapd ni dnsmasq installés par défaut.

```
sudo apt install hostapd dnsmasq -y
```

Maintenant que tous les outils sont installés, je commence avec la configuration du point d'accès.

Configuration des services point d'accès

Je vais modifier 3 fichiers:

```
/etc/dhcpd.conf
```

```
/etc/hostapd/hostapd.conf
```

```
/etc/dnsmasq.conf
```

Ceci sont pour configurer le point d'accès.

Je commence par dhcpd.conf:

```
interface wlan0
```

```
static ip_address=192.168.0.10/24
```

```
nohook wpa_supplicant
```

On demande de configurer wlan0, donc le Wi-Fi, pour qu'il ait une IP statique de 192.168.0.10 avec un masque de 24. wpa_supplicant sert pour utiliser le protocole WPA2 mais nohook est pour que le hook de WPASupplicant ne soit pas appelé.

Pour dnsmasq.conf:

```
interface=wlan0
```

```
dhcp-range=192.168.0.11,192.168.0.73,255.255.255.0,24h
```

On demande de configurer wlan0, le Wi-Fi, et on lui donne une plage DHCP de 192.168.0.11 à 192.168.0.73 avec le masque /24 et le 24h est la durée du bail.

Pour hostapd.conf:

```
interface=wlan0 #L'interface Wi-Fi
```

```
driver=nl80211 #Pilote standard
```

```
ctrl_interface=/var/run/hostapd
```

```
ctrl_interface_group=0
```

```
ssid=AP-ESTOM #Nom du point d'accès
```

```
hw_mode=g #Mode réseau (g -> 802.11g)
```

```
channel=7 #Canal Wi-Fi pour ne pas perturber d'autres réseaux
```

```
wmm_enabled=0 #Qualité de service désactivée
```

```
macaddr_acl=0 #Aucun whitelist MAC
```

```
auth_algs=1 #Algorithme d'authentification
```

```
ignore_broadcast_ssid=0 #Rend le SSID visible
```

```
wpa=2 #Protocole WPA2
```

```
wpa_passphrase=ESTOM2025 #Mot de passe
```

```
wpa_key_mgmt=WPA-PSK #Méthode de gestion de clés
```

```
rsn_pairwise=CCMP #Chiffrement WPA2
```

```
country_code=FR
```

Hostapd est le service qui va faire tourner le point d'accès. Il ne faut pas oublier de modifier le fichier /etc/default/hostapd et ajouter la ligne

DAEMON_CONF="/etc/hostapd/hostapd.conf" pour terminer la configuration.

Maintenant il reste à lancer les services:

```
sudo systemctl unmask hostapd
```

```
sudo systemctl enable hostapd dnsmasq
```

```
sudo systemctl start hostapd dnsmasq
```

Le point d'accès Wi-Fi est fonctionnel et joignable.

Configuration du serveur Web

Après l'installation d'Apache 2, je déplace les fichiers .php dans /var/www/html pour que, dès qu'on écrit l'adresse, on tombe directement sur la page d'accueil nommée

`index.php`. Je mets en place MariaDB en donnant un mot de passe pour root mais aussi la création d'un compte superviseur pour l'accès à la base de données pour le serveur TCP et le serveur Web. Pour activer le protocole HTTPS, je crée un certificat auto-signé. Pour commencer, on active le SSL:

```
sudo a2enmod ssl
```

Maintenant que le SSL est activé, on peut faire une requête de la création du certificat avec OpenSSL:

```
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout
/etc/ssl/private/[nom].key -out /etc/ssl/certs/[nom].crt
```

Après avoir rempli le formulaire, on part configurer le site pour prendre en compte le SSL. On va dans le fichier de configuration des fichiers hôtes. Puisque je n'ai pas besoin d'avoir de multiples sites, j'utilise le fichier par défaut.

```
sudo nano /etc/apache2/sites-available/000_default.conf
```

Puis on va ajouter ou modifier le contenu du fichier pour avoir la configuration SSL:

```
<VirtualHost *:443>
```

```
    ServerName your_domain_or_ip
```

```
    DocumentRoot /var/www/html
```

```
    SSLEngine on
```

```
    SSLCertificateFile /etc/ssl/certs/[nom].crt
```

```
    SSLCertificateKeyFile /etc/ssl/private/[nom].key
```

```
</VirtualHost>
```

Maintenant si on part sur le site avec le nom de domaine, 'estom.fr', on reçoit un avertissement du navigateur, ce qui est normal puisque c'est un certificat auto-signé. On peut ignorer l'avertissement et on peut accéder à la version HTTPS de l'interface Web.

Mise au point de services de démarrage

Même si tout tourne bien, j'ai rencontré un problème où `hostapd` doit toujours être redémarré pour faire apparaître le réseau. Je suis aussi arrivée à la conclusion que le client doit avoir le jeu prêt dès le branchage de la Raspberry Pi.

Pour répondre à ce besoin, j'ai créé 2 services dans le dossier `/etc/systemd/system`.

Pour le `hostapd`, j'ai créé le service `hostaprststrt.service`:

```
[Unit]
```

```
Description=Restart hostapd service
```

```
Wants=network-online.target
```

```
After=network.target network-online.target
```

```
[Service]
ExecStart=/etc/hostaprstrt.sh
WorkingDirectory=/home/pi
StandardOutput=append:/home/pi/hostaprstrt.log
StandardError=append:/home/pi/hostaprstrt.err
Restart=on-failure
RestartSec=1
User=root
```

```
[Install]
WantedBy=multi-user.target
```

Pour le serveur TCP, j'ai créé le service `estomtcp.service`:

```
[Unit]
Description=Autostart appEstomTCP
After=graphical.target
```

```
[Service]
Type=simple
User=pi
Environment=DISPLAY=:0
ExecStart=/usr/bin/lxterminal -e
"/home/pi/ESTOM/build-appEstom2025-Desktop-Debug/appEstom2025"
Restart=on-failure
```

```
[Install]
WantedBy=graphical.target
```

Grâce à ces 2 services, je peux assurer que le point d'accès et le serveur TCP vont se lancer dès que la Raspberry Pi est branchée à une alimentation.

Pour les activer, on lance les commandes suivantes dans le terminal

```
sudo systemctl daemon-reload
sudo systemctl enable hostaprstrt estomtcp
sudo systemctl start hostaprstrt estomtcp
```

et Raspbian prendra en compte ces deux services pour le prochain démarrage.

V.2 - Partie 2: Le serveur TCP et le programme Qt/C++

Pour la communication avec tous les appareils, un serveur TCP est mis en place. Les ESP32-S3 des appareils accèdent au serveur par le point d'accès Wi-Fi et par le port

5005. Ce serveur a été écrit sur Qt avec C++ et contient quelques classes à part le `main.cpp` qui sert seulement à lancer le serveur. Il ne contient pas d'interface graphique donc il se lance par le terminal.

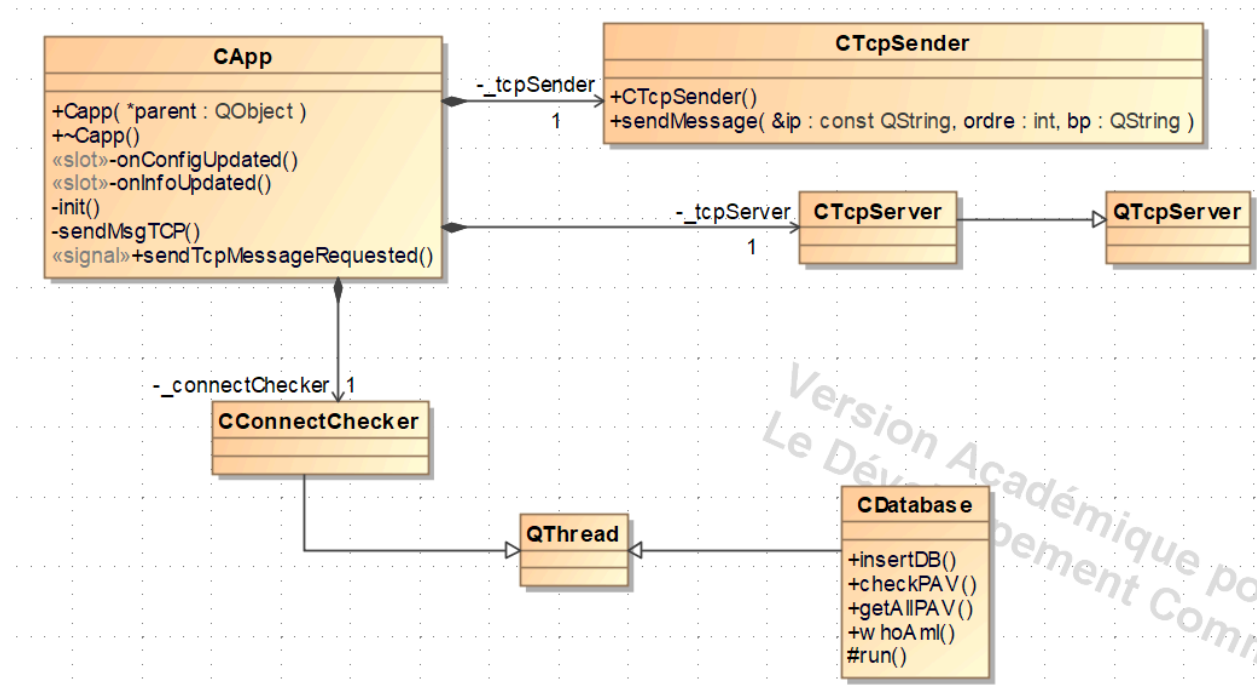


Diagramme de classe (whoAml() est pb())

- **CDatabase:** Cette classe, qui hérite de QThread, gère l'accès et la modification de la base de données. Elle est capable d'insérer des données dans la base de données.
- **CConnectChecker:** Cette classe, qui hérite de QThread, vérifie l'état des appareils pour voir s'ils sont bien connectés ou pas. Elle sert si un BOM ou PAV se déconnecte en plein milieu de la partie.
- **CTcpSender:** Cette classe fabrique des données en format JavaScript Object Notation (JSON) à envoyer aux appareils. Chaque ordre est construit de la même façon pour assurer une compatibilité totale. Le JSON contient le numéro d'ordre et les informations nécessaires à envoyer. Cette classe est surtout conçue pour envoyer les ordres de 'DÉPART', 'PAUSE' et 'FIN' à tous les appareils mais aussi communique surtout au PAV pendant la partie.
- **CTcpServer:** Cette classe est le serveur en mode écoute. Dès que les appareils connectent, le serveur reçoit une trame 'BONJOUR' qui envoie ses informations par défaut. L'écoute est indispensable pour les BOM car ils vont communiquer le plus avec le superviseur.
- **CApp:** C'est la classe démarrée par `main.cpp`. Elle met en route le serveur TCP dans `CTcpServer` pour l'écoute en le poussant vers un thread. Puis elle fait

appel à CConnectChecker pour vérifier les connexions. C'est elle qui envoie les ordres pour CTcpSender à envoyer dès qu'il y a une modification dans la table Config car cette table est celle qui contrôle le jeu. Elle envoie aussi des données pour insérer dans la base de données en utilisant CDatabase. Cette classe est comme la classe principale de ce programme.

Le programme que j'ai reçu est un simulateur qui contient les bases mais pas toutes les communications. Mon travail ici est de créer tous les ordres pour envoyer aux BOM et PAV avec un JSON mais aussi traiter les actions que m'envoie les BOM et PAV. Je dois aussi communiquer avec mes collègues pour avoir les bons ordres à envoyer et qu'ils soient tous compatibles avec des données différentes mais avec la même structure.

La structure de ces messages est simple. Les messages envoyés par le superviseur sont des ordres tandis que les messages envoyés par le BOM et le PAV sont des status.

La trame BONJOUR que je dois recevoir est de la structure suivante:

```
{  
  "status": "0",  
  "type": "BOM" (ou "BUS"/"PAV"/"ABB")  
  "couleur": "X" (valeur du DIP SWITCH)  
}
```

Le status 0 indique que c'est le BONJOUR et que le BOM ou PAV est initialisé. D'après l'état de partie, le serveur va répondre différemment mais la structure reste la même.

Voici la réponse du superviseur:

Pour BOM/BUS:

```
{  
  "ordre": "0",  
  "etatJeu": "X",  
  "collisions": "X"  
  "leds": "X"  
}
```

Pour PAV/ABRIBUS:

```
{  
  "ordre": "0",
```

```

"etatJeu": "X"
"etatPAV": "X"
}

```

Ceci est la communication pour le démarrage, que nous pouvons retrouver ici:

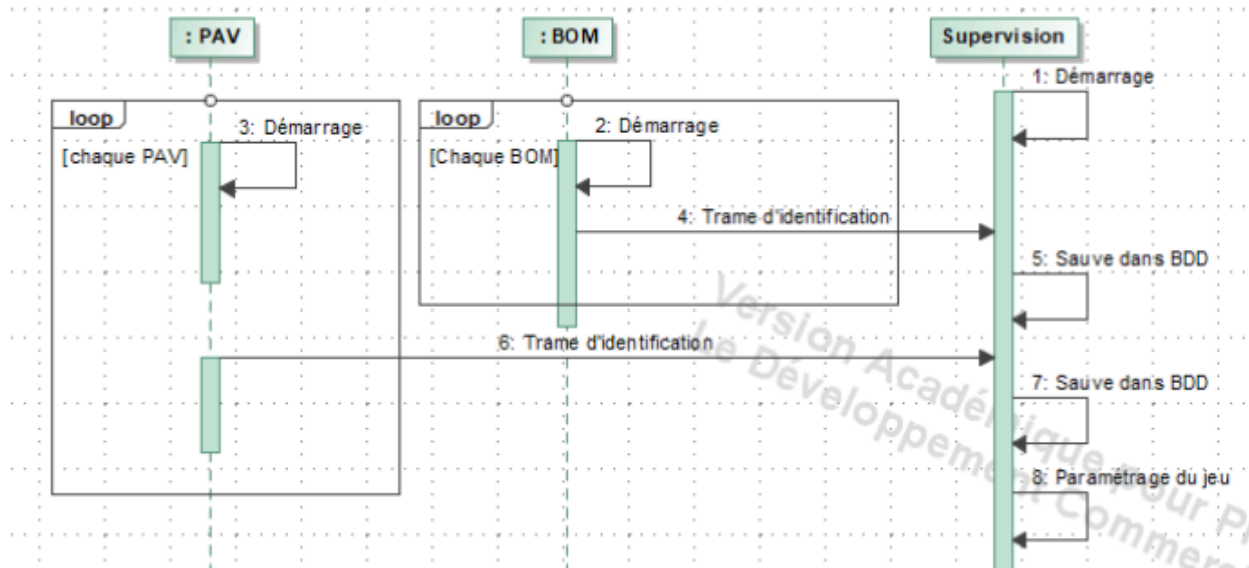


Diagramme de séquence sdTramesInit

La supervision communique beaucoup entre la base de données et les PAV et BOM pour que le jeu soit cohérent.

Dans le serveur, ceci nécessite CApp, CDatabase et CTcpSender.

Voici l'extrait pour le cas du message du BONJOUR, nous arrivons dans la méthode `on_infoUpdated()` et les cases 0:

```

T_SEND toSend;
if (infos.type == "BOM") {
    toSend.pb = "B";
    switch (infos.status.toInt()) {
        case 0: //BONJOUR
            qDebug() << "BONJOUR de BOM (" << ip << ")";
            _dbReader->insertDB("BOM", QVariantList{ip,
infos.status, infos.couleur});
            sendMsgTCP(ip, 0, toSend);
            break;
        ...
    }
    if (infos.type == "PAV") {
        toSend.pb = "P";
        switch (infos.status.toInt()) {
            case 0: // BONJOUR

```

```

        qDebug() << "BONJOUR de PAV (" << ip << ")";
        _dbReader->insertDB("PAV", QVariantList{ip,
infos.status, infos.couleur});
        sendMsgTCP(ip, 0, toSend);
        break;

```

CApp prend les infos qu'il doit envoyer. Il retrouve le type de l'appareil, si c'est le BOM ou le PAV et le stocke dans `pb`. Puis, il envoie l'IP de l'appareil, l'ordre et les données à la classe `CTcpSender` par la méthode `sendMsgTCP()` et on atterrit sur `on_sendTcpMessage()` !

Voici l'extrait:

```

QJsonObject jsonObj;
jsonObj["ordre"] = QString::number(ordre);
switch(ordre) {
case 0:
    qDebug() << "Ordre0: INIT;";
    if (toSend.pb == "B") {
        jsonObj["etatJeu"] = toSend.etatJ;
        jsonObj["collisions"] = toSend.collisions;
        jsonObj["leds"] = toSend.leds;
    } else {
        jsonObj["etatPAV"] = toSend.etatP;
        jsonObj["etatJeu"] = toSend.etatJ;
    } // else
    break;
...
    QJsonDocument doc(jsonObj);
    QByteArray message = doc.toJson(QJsonDocument::Compact) +
'\n';

    socket->write(message);
    socket->flush();

```

Grâce à `QJsonObject`, on crée l'objet JSON qui va être transformé en document JSON grâce à `QJsonDocument`. D'abord, on déclare l'ordre qui est déjà donné par CApp. Puis `pb` est utilisé pour identifier le type d'appareil qui va recevoir ce message. L'objet JSON est construit d'après des données que l'appareil a besoin et puis on sort du switch. Le document JSON est créé puis envoyé par le socket.

Tous les ordres sont conçus de cette façon. Le serveur écoute, reçoit un status, le traite dans CApp et l'envoie par CTcpSender. Mais pour que tout cela fonctionne, le serveur interroge la base de données grâce à CDatabase.

Si vous observez le diagramme sdTramesInit, le superviseur sauvegarde dans la base de données toutes les informations envoyées et reçues grâce à la méthode `insertDB()`.

CDatabase interroge la base de données dès l'initialisation du serveur et cherche la table de configuration Config avant que le procès d'initialisation des appareils commence.

D'abord, elle nettoie les tables BOM et PAV en utilisant une connexion temporaire à la base de données et met le status du jeu à 0 pour dire au serveur Web qu'on n'a pas de jeu en cours car le serveur vient de démarrer (ou redémarrer en cas de problème technique):

```
void CDatabase::resetTables(QSqlDatabase &db)
{
    QSqlQuery query(db);

    QStringList tables = {"BOM", "PAV"};
    for (const QString &table : tables) {
        //qDebug() << "Réinitialisation de la table :" << table;

        if (!query.exec("DELETE FROM " + table)) {
            qDebug() << "Erreur lors de la suppression des
données de" << table << ":" << query.lastError().text();
        } // if delete

        if (!query.exec("ALTER TABLE " + table + "
AUTO_INCREMENT = 1")) {
            qDebug() << "Erreur lors de la réinitialisation de
l'ID de" << table << ":" << query.lastError().text();
        } // if alter
    } // for

    if (!query.exec("UPDATE Config SET Status = 0")) {
        qDebug() << "Erreur lors du changement de status de
config :" << query.lastError().text();
    } // if delete
    query.finish();
    qDebug() << "Tables BOM et PAV réinitialisées avec succès.";
} // method
```

Puis elle récupère la ligne dans la table Config pour savoir la configuration du jeu pour l'instant:

```

    T_CONFIG cfg;
    while (!isInterruptionRequested()) {
        QSqlQuery query(threadDb);
        if (!query.exec("SELECT * FROM Config")) {
            qDebug() << "Erreur SELECT Config :" <<
query.lastError().text();
            QThread::sleep(1);
            continue;
        } // if select

        if (!query.next()) {
            qDebug() << "La table Config est vide ou aucune
ligne trouvée.";
            QThread::sleep(1);
            continue;
        } // if next
        cfg.options = query.value("Options").toString();
        cfg.ptsRecolte = query.value("PtsRecolte").toString();
        cfg.nbrPAV = query.value("NbrPAV").toString();
        cfg.status = query.value("Status").toString();
        cfg.luminosite = query.value("Luminosite").toString();
        emit sig_configUpdated(cfg);
        QThread::sleep(1);
    } // wh

```

A chaque lecture, CDatabase prend une petite pause avant de re-récupérer la ligne Config pour être synchronisée avec le serveur Web sans abuser la connexion avec la base de données.

Le reste des fichiers est en annexe!

V.3 - Partie 3: Le serveur Web

Le site Web



Page d'accueil du serveur Web index.php

Le serveur Web est conçu avec du HTML, CSS, PHP et JavaScript en AJAX.

L'HTML et le CSS sert tout simplement pour l'affichage basique de la page et donc ne sera pas abordé ici.

Le PHP est le langage qui fait que l'affichage fonctionne et qui gère le côté Web. Le serveur interroge beaucoup la base de données pour être le plus synchronisé possible avec le serveur TCP.

Le serveur Web contient 3 pages graphiques accessibles. La page d'accueil **index.php**, l'affichage **gestion-de-la-partie.php** et la configuration **parametre-du-jeu.php**.

Il contient aussi son propre "CDatabase", c'est-à-dire un fichier .php qui peut accéder à la base de données et qui est utilisé par le reste pour mettre à jour les données.

D'abord, le serveur Web fait une remise à zéro du score et de l'affichage:

```
<?php
```

```
require 'private-estom/db_connect.php';
```

```
$data = json_decode(file_get_contents("php://input"), true);
```

```

$stmt = $conn->prepare("UPDATE BOM SET Score = 0, Progression =
0, Remplissage = 0, NbrCollision = 0, Collisions = '');

if ($stmt->execute()) {
    echo json_encode(["success" => true]);
} else {
    echo json_encode(["error" => "Erreur lors de la mise à jour
du score."]);
}
$stmt->close();
$conn->close();
?>

```

Il prépare une requête SQL pour remettre tous les éléments utilisés pour l’affichage à 0 pour redémarrer une partie avec un affichage propre.

Pour le paramétrage du jeu, il utilise un formulaire pour la modification de la table Config.

Il récupère les données déjà présente sur la base de données avec la requête SQL `SELECT Options, PtsRecolte, NbrPAV, MalusCollision, NbrBOM_V, NbrBOM_J, NbrBOM_N, NbrBOM_B, duree FROM Config WHERE Id = 1;` et puis les affichent dans le formulaire.

Les paramètres sont seulement mis à jour si le formulaire est soumis avec `if`

`($_SERVER['REQUEST_METHOD'] === 'POST')` puis écrit les nouvelles valeurs dans la base de données. Voici un extrait:

```

$sqlUpdate = "UPDATE Config SET Options = ?, PtsRecolte = ?,
NbrPAV = ?, MalusCollision = ?, NbrBOM_V = ?, NbrBOM_J = ?,
NbrBOM_N = ?, NbrBOM_B = ?, duree = ? WHERE Id = 1";
$stmt = $conn->prepare($sqlUpdate);
$stmt->bind_param("iiiiiii", $nouveauOptions,
$nouveauPtsRecolte, $nouveauNbrPAV,
$nouveauMalusCollision, $nouveauNbrBOM_V, $nouveauNbrBOM_J,
$nouveauNbrBOM_N, $nouveauNbrBOM_B, $nouveauTempsPartie);

```

La requête SQL est préparée avec des `?` puis on utilise `bind_param` avant d’exécuter la requête. Cette méthode de préparation de requête sert surtout à protéger des attaques d’injections SQL.

Gestion de la partie accède aussi beaucoup à la base de données mais au lieu d’être isolée à la table Config, elle utilise toutes les tables de la base de données.

Notamment la table BOM qui contient les clés pour l’affichage comme la progression qui définit comment l’image du BOM se translate sur la route et le score qui s’additionne ou se divise par les points et le malus définis dans le paramétrage.

La base de données

La base de données, nommée ESTOM, contient 3 tables: BOM, Config et PAV.

Cette base de données n'a aucun lien nécessaire entre les tables donc un modèle MCD (Modèle Conceptuel de Données) avec les cardinalités n'est pas possible.

Voici l'ancienne base de données:

Bdd : ESTOM

TOM	BOM	Config
ID_TOM INT	ID_BOM INT	Id INT
Stockage TINYINT(4)	Couleur TINYINT(4)	Option TINYINT
Couleur TINYINT(4)	Progression DECIMAL(5,2)	PtsRecolte INT
Connected TINYINT	Status TINYINT(1)	NbrTOM INT
IPAddr VARCHAR(45)	Connected TINYINT	duree INT
	NbrCollision INT	Status INT
	IPAddr VARCHAR(45)	NbrBOM_V INT
	Joueur VARCHAR(45)	NbrBOM_J INT
	Collisions TEXT	NbrBOM_N INT
		NbrBOM_B INT
		MalusCollision VARCHAR(45)
		Temps_restant INT
		Heure_depart DATETIME
Indexes	Indexes	Indexes
PRIMARY	PRIMARY	PRIMARY

Et la nouvelle base de données:

Bdd : ESTOM

BOM	Config	PAV
ID INT(11)	Id INT(11)	ID INT(11)
Progression DECIMAL(5,2)	Options TINYINT(1)	Stockage TINYINT(4)
Remplissage INT(11)	PtsRecolte INT(11)	Couleur TINYINT(4)
Couleur TINYINT(4)	NbrPAV INT(11)	Connected TINYINT(...)
Joueur VARCHAR(20)	MalusCollision VARCHAR(4...)	IPAddr VARCHAR(20)
Status TINYINT(4)	NbrBOM_V INT(11)	Status TINYINT(4)
Connected TINYINT(4)	NbrBOM_J INT(11)	
NbrCollision TINYINT(4)	NbrBOM_N INT(11)	
IPAddr VARCHAR(20)	NbrBOM_B INT(11)	
Collisions TEXT	Temps_restant INT(11)	
	duree INT(11)	
	Status INT(11)	
	Heure_depart DATETIME	
	password VARCHAR(5)	
	Luminosite INT(11)	
Indexes	Indexes	Indexes
PRIMARY	PRIMARY	PRIMARY
IPAddr		IPAddr

Premier changement pour la base de données est que la clé **IPAddr** est unique, pour que le serveur TCP n'écrive pas plusieurs fois la même ligne pour le même appareil. On a aussi ajouté une clé **Luminosite** pour que la luminosité des LEDs puissent être changés par le site Web dans les paramètres.

V.4 - Quelques changements

Le démarrage automatique du serveur TCP

Le service `estomtcp` que j'ai créé au début ne fonctionnait jamais. Le serveur ne démarrait pas automatiquement.

Donc j'ai créé un dossier dans `/home/pi/.config` qui se nomme `autostart`. Puis je crée un fichier nommé `terminal.desktop` qui contient:

```
[Desktop Entry]
Type=Application
Name=ESTOMServer
Exec=lxterminal -e /opt/estom/appEstom2026
```

Mais j'ai observé qu'il avait du mal à se démarrer automatiquement puisqu'il avait besoin des graphismes, donc du bureau, et la Raspberry Pi a besoin d'un mot de passe pour afficher ce bureau.

J'ai pris un risque en allant sur `raspi-config`, puis `System Options` puis `Auto Login`. Ceci fait que la Raspberry Pi ne demande plus le mot de passe à l'utilisateur qui est un grand risque de sécurité et qu'il faut trouver d'autres moyens de la protéger des attaques possibles.

Le changement du point d'accès... mais un retour à l'ancien service

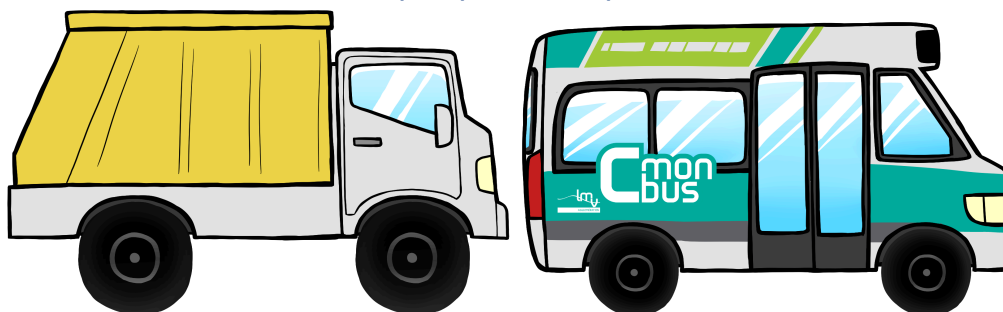
Le point d'accès Wi-Fi était géré par le service `hostapd` mais dans nos tests, surtout chez les clients, nous avons vu qu'on avait des problèmes pour se connecter au point d'accès. Nous avons donc décidé de changer de service de point d'accès en utilisant `NetworkManager`. Mais nous avons vite rencontré des problèmes pendant notre démonstration à LMVélo.

Non seulement `NetworkManager` est moins sécurisé (pas de WPA2 et utilise le chiffage TKIP) mais aussi qu'il a moins d'options de configuration pour le point d'accès. Donc on a décidé de retourner sur `hostapd` par le biais d'une Raspberry Pi 5 pendant toute la démonstration. Pour la limite de connexion, on a décidé de peut-être essayer une clé Wi-Fi qui pourrait gérer les connexions pour `hostapd`.

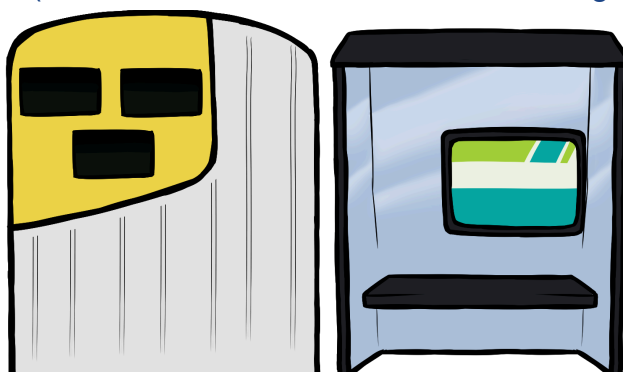
Les graphismes du site Web

Les images de l'ancien site Web étaient des images prises des sites et dont la bannière et l'arrière-plan étaient générés par l'IA. Les couleurs étaient saturées et l'image n'avait aucune cohérence.

En tant que dessinatrice, j'ai décidé de redessiner les icônes et autres mais aussi ajouter le bus et les abribus, voici quelques exemples de mon travail:



Le BOM (seulement la couleur de la benne change) et le bus.



Le PAV (la couleur, comme le plastron du réel, change) et l'abribus.



Les cadres de scores qui rappelle quelle couleur sert pour quel type de déchet
+ le cadre du bus qui est un mini abribus!



La bannière qui fait référence aux mécanismes du jeu.



Et finalement, le nouveau logo!

VI - Conclusion

C'était un plaisir de travailler sur le projet ESTOM avec mes collègues mais aussi avec LMV Agglomération. De voir le projet passer de simulateur à un vrai jeu où les enfants puissent s'amuser avec et utiliser les compétences que j'ai appris en cours et de les employer dans le projet.

Le projet est bientôt terminé et j'ai hâte de voir le produit final. Malgré les petites corrections à faire, j'ai pu approfondir mes connaissances et apprendre de nouvelles méthodes.

Le début était difficile, de prendre le travail d'une autre personne mais c'était intéressant de voir comment l'étudiant avant moi à procédé avec son travail. Après avoir pris en main par le biais d'un diagramme de classes, j'ai pu ajouter les modifications nécessaires pour le fonctionnement réel du projet.

Je voudrais remercier mes professeurs pour m'avoir aider à naviguer ce projet et à m'améliorer dans les domaines comme l'administration d'un serveur Web ainsi que la programmation en C++ du serveur TCP.

Je voudrais aussi remercier mes collègues avec lesquels j'ai bien communiqué avec eux et mis les pièces que chaque partie en place comme la désignation des ordres et états pour la bonne communication entre la supervision et les appareils.

Et surtout un grand merci à LMV Agglomération de nous avoir fait confiance avec un projet magnifique.

Partie Personnelle ER1: BOM/Bus

BLANC Thomas

I - Introduction personnelle

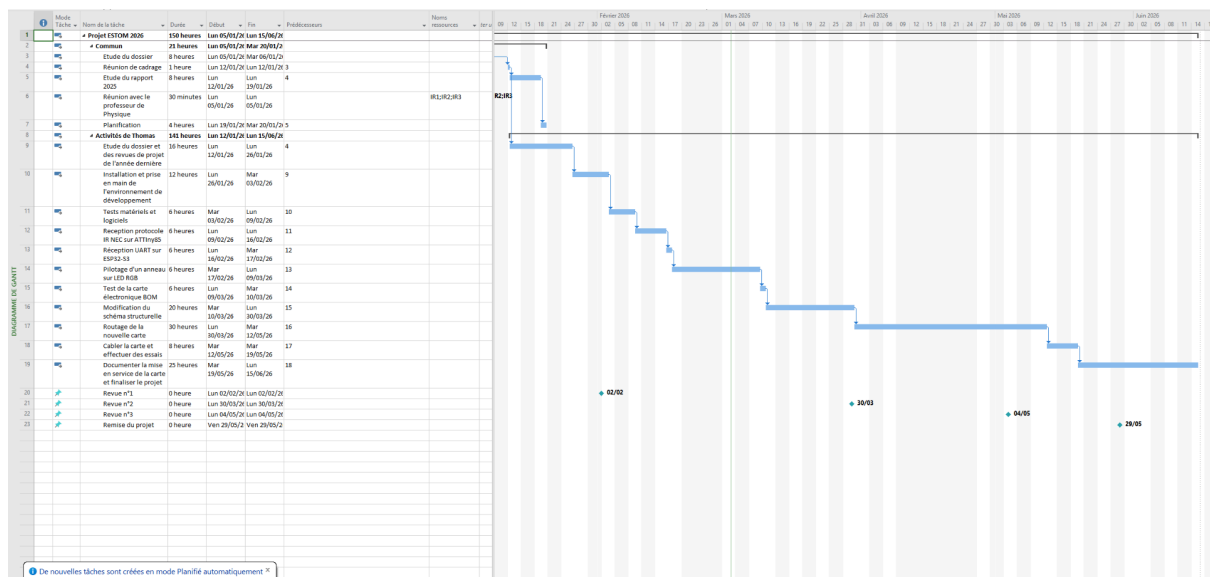
On m'a assigné au projet nommé Espace de Sensibilisation au Tri d'Ordures Ménagères (**ESTOM**) demandé par Luberon Mont de Vaucluse (**LMV**). Dans ce projet, j'ai la charge de la **BOM** (Benne à Ordures Ménagères) ainsi que du **BUS**.

J'ai commencé par analyser le cahier des charges afin de comprendre le fonctionnement global du système et définir précisément mes missions. Je vais vous présenter les activités que j'ai effectuées durant ce début de projet, à savoir :

- **Préparation de l'environnement de travail** : notamment la mise en place du WiFi et la configuration des outils nécessaires au développement (ESP32 et ATtiny85).
- **Réception des signaux infrarouges** : analyse d'une trame en protocole NEC envoyée par les PAV à l'aide d'un analyseur logique, afin de pouvoir décoder les informations reçues.
- **Pilotage d'une matrice de LED** : permettant d'indiquer au conducteur la couleur du PAV concerné, puis de simuler une phase de chargement via une animation lumineuse.
- **Sécurité et fiabilité** : réalisation de tests sur des capteurs anti-choc et mise en place d'un système d'anti-rebond (debounce) afin de fiabiliser l'ensemble de son fonctionnement.
- **Conception et évolution matérielle** : Étude et modification du **schéma structurel** sous KiCad pour intégrer les nouvelles protections (notamment la diode D5 contre les inversions de polarité) et optimisation du **routage** du circuit imprimé (PCB) pour garantir une isolation optimale entre la partie puissance (LEDs) et la partie commande (ESP32).
- **Fabrication du prototype : Brasage** des composants montés en surface (CMS) et traversants sur la carte ESTOM-2026. Cette étape a nécessité une grande précision, particulièrement pour le XIAO ESP32-S3 et les connecteurs des modules de signalisation.

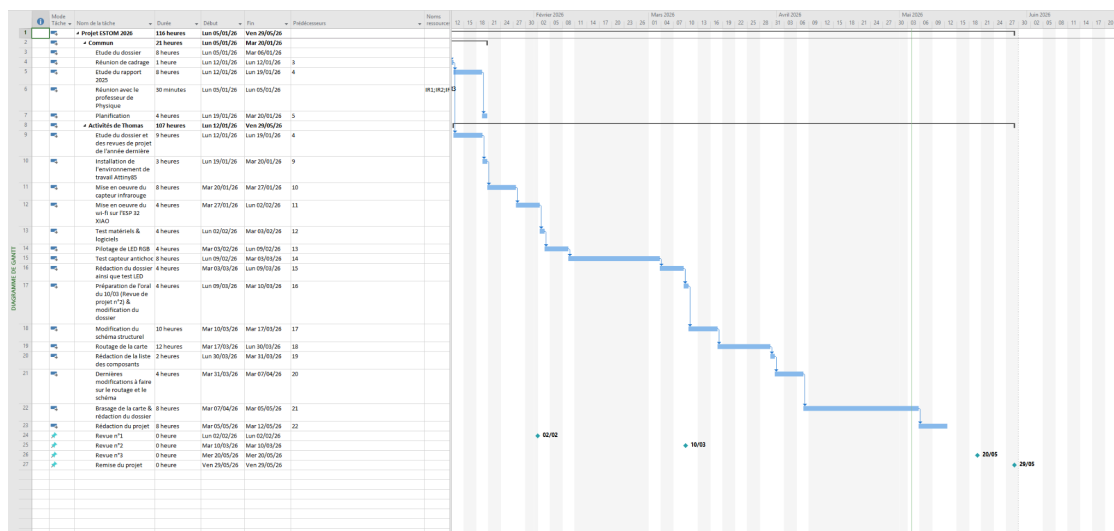
II - Organisation du projet – Diagramme de Gantt prévisionnel et réel

Prévisionnel :



La réalisation de ce diagramme de Gantt prévisionnel m'a permis de réaliser la difficulté d'estimer avec précision la durée des tâches en amont. En pratique, certaines étapes techniques ont nécessité un temps de recherche plus important que prévu, tandis que d'autres ont été finalisées plus rapidement.

Réel :

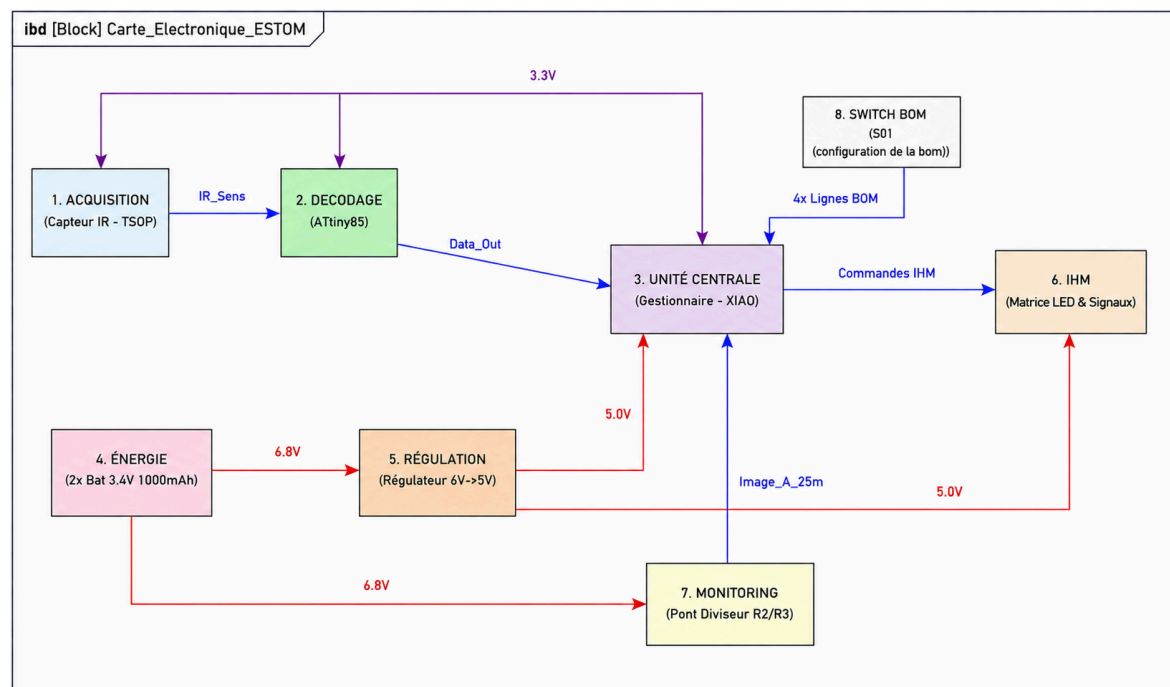


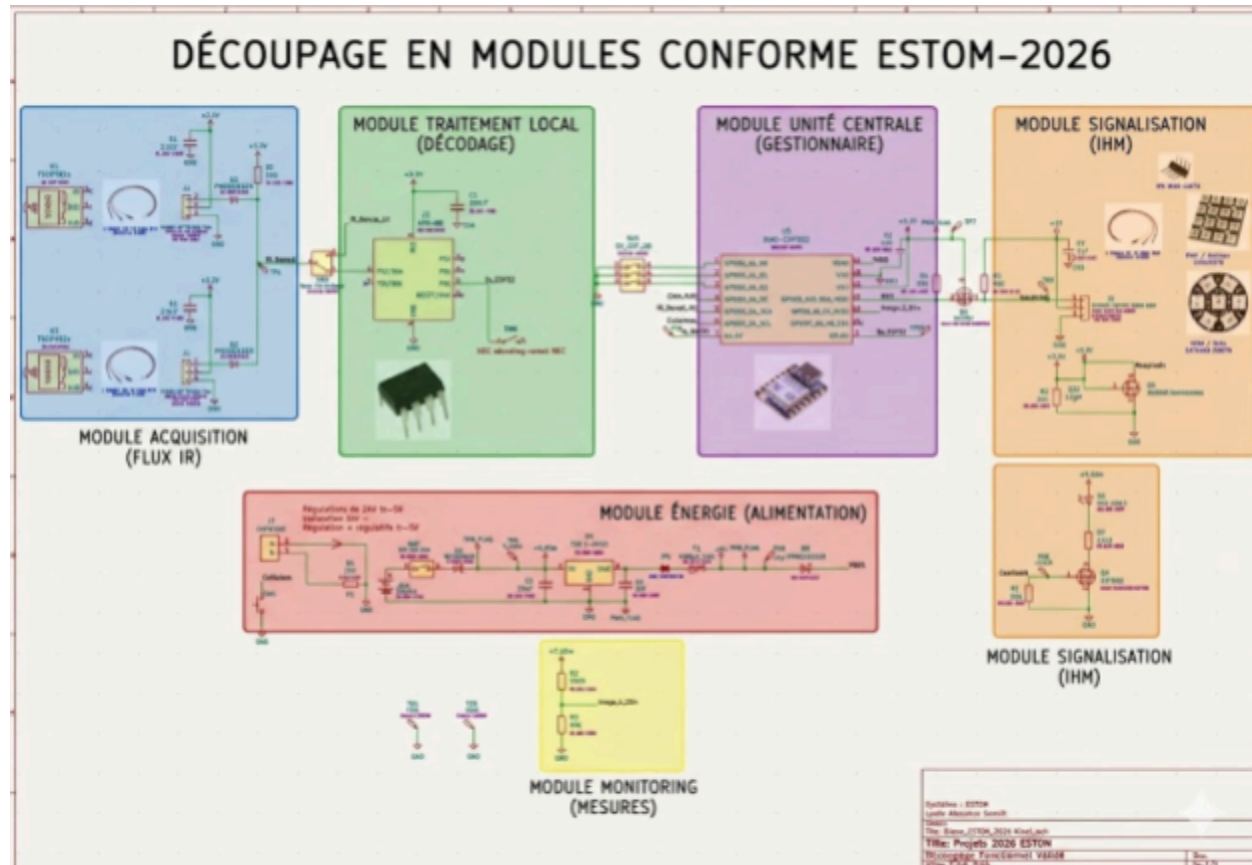
III - Présentation du Travail

Fiche de suivi

Date	Activités	Détail	heures				
05/01	Découverte du projet	Lecture de la revue de projet de l'année dernière ainsi qu'un résumé avec le professeur	3				
09/01	Formation Microsoft Project		1				
09/01	Création de fiche suivis et de mon planning	Fiche de suivis avec google sheets et planning avec Project	3				
12/01	Finalisation du diagramme de Gantt et de l'estimation de temps travail	Estimation du temps de travail du projet	4				
13/01	Certification PIX	Examen de la certification PIX	2,5				
	Test de l'application mobile pour piloter le BOM		0,5				
	Rédaction du dossier		1				
19/01	Suite de la découverte du projet et de l'analyse des revues précédentes		1,5				
	Installation de l'environnement de travail ATtiny85	Résolution de problème inérant à l'installation, le site pour télécharger le lecteur de carte était down plus le driver ne fonctionnait pas j'ai du chercher une solution Installation de l'environnement de travail ATtiny85 Difficulté à faire fonctionner l'analyseur logique les informations retranscrivent, données corrompues	2,5				
20/01	Mise en oeuvre de capteur infrarouge		4				
26/01	Suite de la mise en oeuvre du capteur infrarouge		4				
26/01	Mise au point avec le professeur sur le projet		1				
	Mise en oeuvre du wi-fi sur l'ESP 32 XIAO S3	Beaucoup de difficulté à mettre en oeuvre le wifi complètement vu le jeudi de la dite semaine	3				
02/02	Test bootloadeur ESP32-S3 XIAO	Ajoute vu sur reddit pour corriger le bug du bootloadeur que je vais tester aujourd'hui. Test non concluant	1				
	Test wifi sur la carte du projet	La carte est correctement connectée au wifi ! Test concluant + rédaction sur le gdoc du projet	1,5				
	Pilotage de matrice de LED avec Kevin	Observation de comment piloter la matrice de LED du PAV	1,5				
03/02	Suite du pilotage de LED avec mon binôme	Suite du pilotage de LED avec but de setup les couleurs avec le switch monté sur la carte	4				
09/02	Rédaction partie LED	Mettre sur feuille mon activité sur le pilotage de LED fait avec mon binôme	1				
	Test sur capteur antihoc avec Mohammed	Les capteurs ont été installés avec succès sur le camion maintenant il faut mesurer le delta T, afin de modifier le code pour supprimer les rebonds éventuelles (les 5 capteurs n'auront pas la capacité)	3				
10/02	Continuation de l'activité précédente avec Moha	Monsieur Hortoland nous a aidés à faire un code qui fonctionne avec comme base un code exemple simple d'utilisation	4				
03/03	Rédaction du dossier du projet ainsi que TEST LED avec KEVIN		4				
09/03	Préparation de l'oral du 10/03 (Revue de projet n°2) & modification du dossier		4				
16/03	Modification du schéma électronique	Je vais modifier le schéma de 2025 avec les modifications apportées par monsieur Hortoland sur le schéma modifié en pdf. Normalement si tout ce passe bien j'ai terminé la saisie de schéma et je vais commencer le routage de la carte demain le 17/03 Il me reste quelques éléments à modifier avant de pouvoir attaquer le routage de la carte. J'ai également changé ma méthodologie de travail suite à une explication du prof de comment bien ranger et trier son travail afin d'être efficace.	4				
17/03	Suite et fin de la saisie de schéma & début du routage de la carte		4				
23/03	Suite du routage de la carte		4				
24/03	Suite du routage de la carte		4				
30/03	Fin du routage de la carte	Le routage est quasiment terminé il restera 2,3 modifications future à faire	1,5				
	Fichier excel & calcul du prix des composants d'une carte électronique complète pour le bom	Avec l'aide du schéma de saisie j'ai remplis un excel avec le cout des composants qui pour l'instant s'élève à environ 29€ TTC	2,5				
31/03	Modification du schéma structurel et du routage		4				
27/04	Mise au point avec le professeur sur les prochaines activités		1			Total d'heures	95
	Réunion au tour de la table pour préparer les composants à braser le lendemain		1				
	Accompagnement de Kevin à l'infirmerie (et attente de l'infirmerie)		1				
	Rédaction du dossier pendant la demire heure		1				
28/04	Brassage de la carte avec la pate à braser et le four à refusion		2,5				
	Rédaction du dossier du projet		1,5				
04/05	Rédaction du dossier du projet		4				
05/05	Rédaction du dossier du projet		4				

Voici ma fiche de suivi du projet de BTS qui va me permettre de retracer toute mon activité durant l'année 2026.





Nous avons procédé à quelques modifications au niveau du schéma structurel :

- -Ajout de connecteurs JST (J2, J3, J4) pour les capteurs IR et les LED, facilitant le montage et la maintenance.
- - Connexion directe de la sortie régulée vers la broche VBUS de l'ESP 32 et ajout d'une diode de protection (D5).
- - Réduction du bloc "Collision/Gâchette" à un seul bouton (SW1) pour plus de clarté.

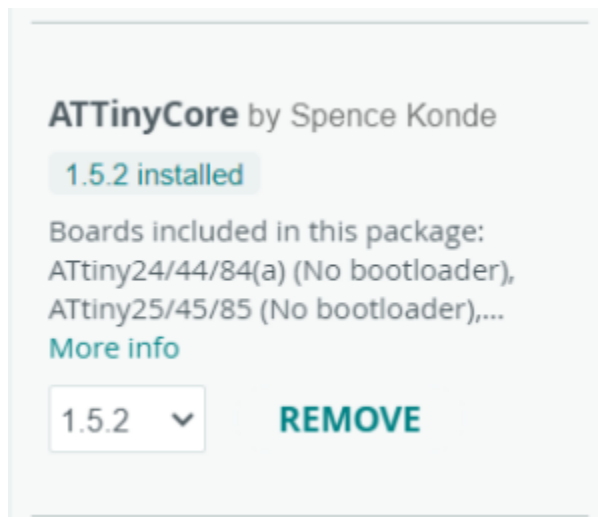
III.2 - Mise en place de l'environnement de travail

Rôle de l'ATtiny85 dans le système ESTOM

Avant de procéder à la configuration matérielle, il est important de préciser le rôle du microcontrôleur **ATtiny85**. Au sein de la **BOM**, il fait office de processeur esclave dédié à une tâche critique : la gestion du flux infrarouge.

- **Déchargement du processeur principal** : Il permet à l'ESP32-S3 de se concentrer sur la communication Wi-Fi et le pilotage du système
- **Réactivité** : L'ATtiny85 assure une écoute en temps réel des signaux IR (protocole NEC) sans latence.

Préparation de l'environnement de développement L'utilisation de ce microcontrôleur spécifique nécessite une configuration particulière de l'IDE Arduino via la plateforme **ATTinyCore**.



Installation dans l'IDE Arduino :

- URL à ajouter dans les Préférences : http://drazzy.com/package_drazzy.com_index.json
- Puis ajouter la carte ATTinyCore

Résolution d'un problème d'installation : Lors de la mise en place, j'ai rencontré l'erreur suivante :

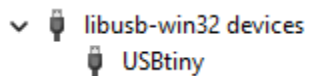
```
Failed to install platform: 'ATTinyCore:avr:1.5.2'. Error: 2
UNKNOWN: connectex: No connection could be made because the target
machine actively refused it.
```

Cette erreur de connexion au serveur de téléchargement a été résolue par l'installation manuelle d'une **version antérieure stable** de la plateforme, permettant ainsi de finaliser l'intégration des cartes ATtiny dans l'interface de développement.

Configuration du programmeur SparkFun dATtiny85 : Le Tiny AVR Programmer (programmeur SparkFun d'ATtiny85) permet d'établir la connexion entre le PC et l'ATtiny85 afin de le configurer et de graver son bootloader.

Pour que le PC communique avec le **Tiny AVR Programmer**, j'ai dû installer manuellement le driver pour le périphérique initialement reconnu sous le nom de **FabISP**.

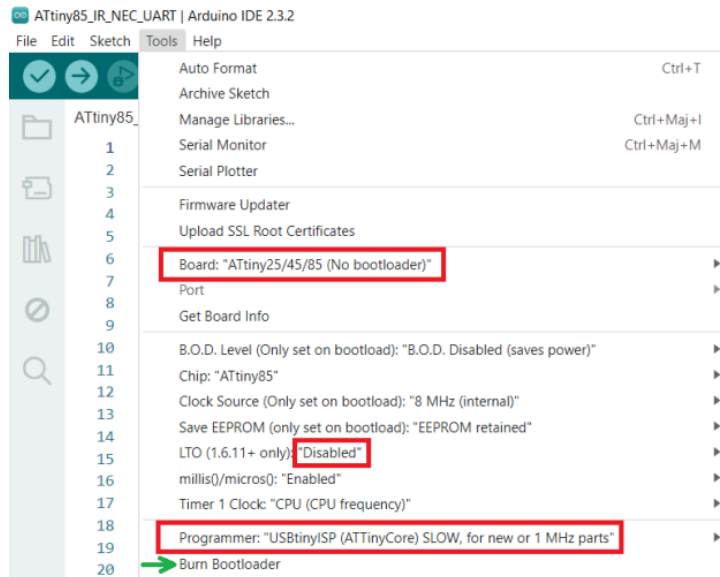
- *Procédure :* Gestionnaire de périphériques > Mettre à jour le pilote > Recherche manuelle du dossier téléchargé.
- *Résultat :* Le périphérique est désormais correctement identifié sous le nom de **USBtinyISP**.



La partie installation n'est pas terminée ! Il faut maintenant préparer le microcontrôleur à être utilisé il faut faire un burn bootloader :

La fonction Burn Bootloader (Graver le bootloader) sert à configurer le microcontrôleur en programmant ses fuses (horloge, options de démarrage) afin de le préparer à recevoir du code.

Pour se faire il faut régler les paramètres suivants (IMPORTANT) :



Pour finaliser la préparation du microcontrôleur, j'ai dû paramétrer l'IDE Arduino avec les options spécifiques à notre matériel. Cette étape est cruciale pour que le code soit compilé avec les bonnes vitesses d'exécution.

Paramètres de flashage :

- **Board :** ATtiny85 (No bootloader)
- **LTO (Link Time Optimization) :** Disabled (pour garantir la stabilité de la compilation).
- **Programmer :** USBtinyISP (ATTinyCore) **SLOW**. *Note : Le mode SLOW est utilisé pour assurer une communication stable lors du premier flashage.*

Une fois ces paramètres réglés, j'ai exécuté la fonction **"Burn Bootloader"**.

Validation par test "Blink" Avant de passer au code complexe de décodage IR, j'ai effectué un test de "Blink" (clignotement de LED) pour valider la chaîne de programmation :

1. Ouverture de l'exemple : File -> Examples -> Basics -> Blink.
2. Téléversement via la commande spécifique : Sketch -> Upload Using Programmer.

Ce test a permis de confirmer que le PC, le programmeur et le microcontrôleur communiquent parfaitement.

Passage au prototype de réception IR Une fois le matériel validé, j'ai réalisé le câblage sur plaque d'essai (Breadboard) en m'appuyant sur un schéma **Fritzing**. Ce circuit intègre le récepteur infrarouge nécessaire à l'analyse du protocole NEC.

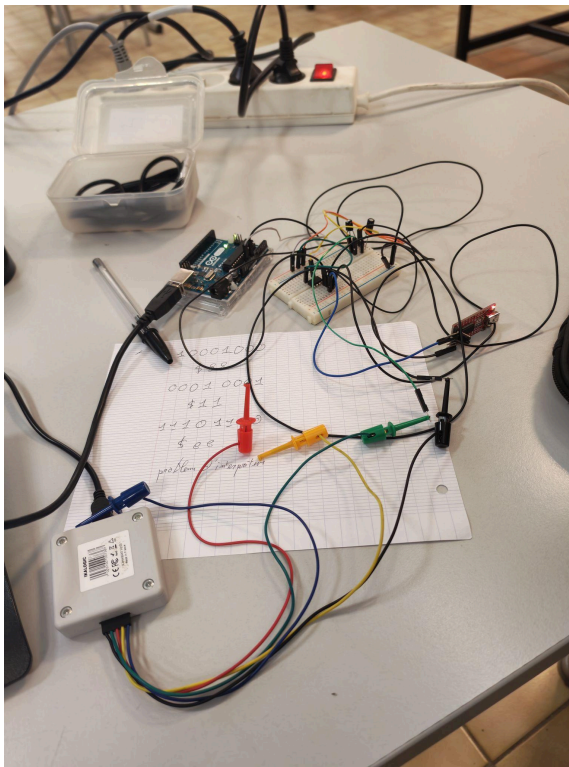
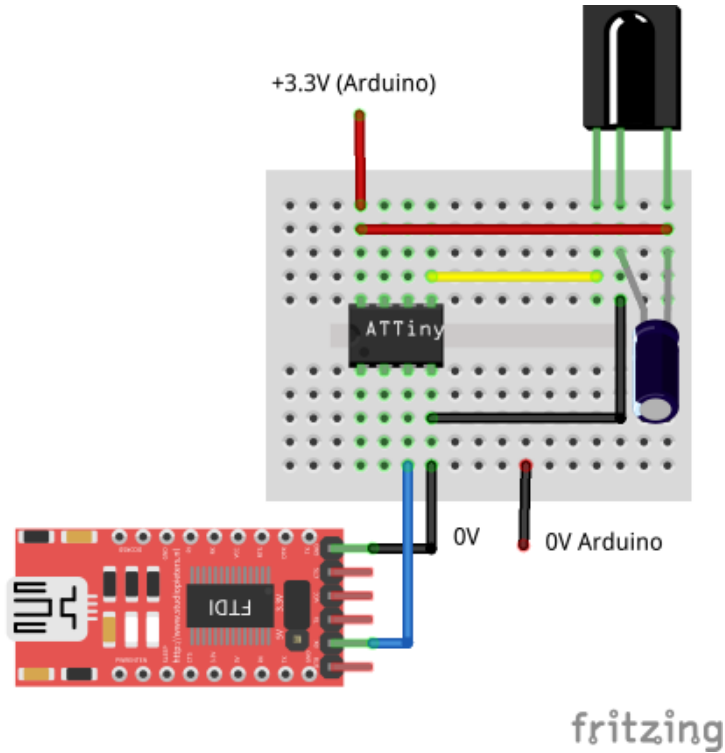


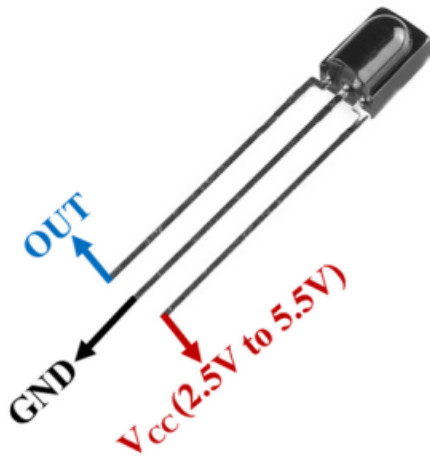
Schéma réalisé sur Fritzing qui m'a aidé à réaliser le montage ci-dessus :



Fritzing est un logiciel de CAO électronique intuitif qui permet de documenter rapidement des prototypes grâce à une vue "planche à essai" réaliste.

III.3 - Réception et analyse des signaux infrarouges

Une fois l'environnement de développement prêt, l'étape suivante a consisté à valider la communication entre les **PAV** (émetteurs) et la **BOM** (récepteur). Pour cela, nous avons utilisé une télécommande simulant l'émission d'une tour et un démodulateur infrarouge raccordé à l'ATtiny85.



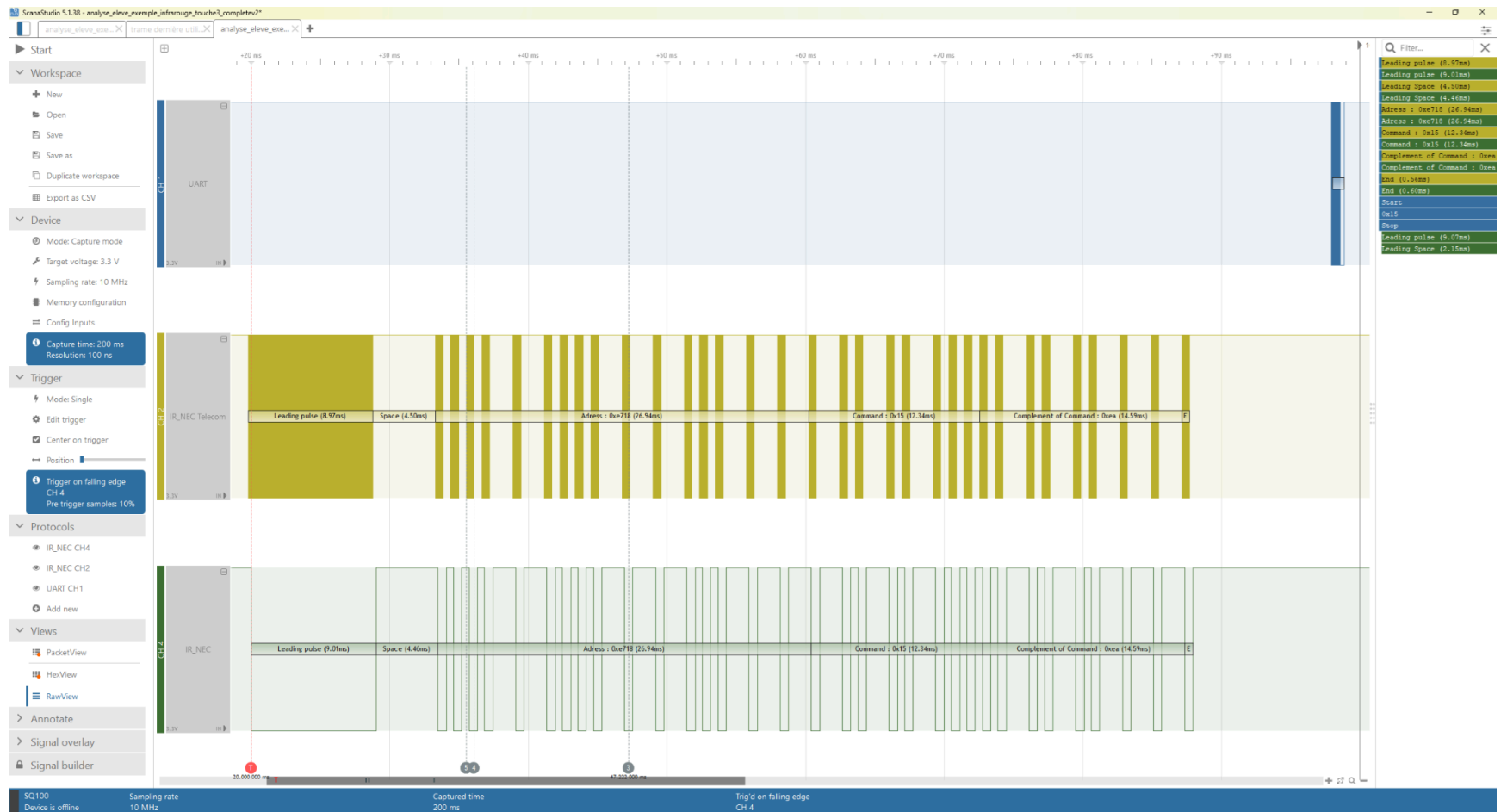
Pour observer le signal en temps réel, j'ai connecté un analyseur logique **SQ100** directement sur la broche *OUT* du récepteur infrarouge. Mon binôme, Kevin, s'est chargé de la configuration de la télécommande pour l'émission des trames.



Le signal infrarouge est émis sur une porteuse de 38 kHz. Le rôle du démodulateur est d'extraire l'enveloppe du signal (les données utiles) et de les envoyer vers le microcontrôleur.

Une fois que l'**ATtiny85** a identifié l'adresse de la tour (par exemple 0x01 pour la TOM Jaune), l'information est transmise à l'**ESP32-S3** (processeur maître) via une communication série.

- **Fonctionnement** : L'ATtiny85 utilise ses broches de sortie série (broche 3 dans notre montage) pour envoyer une trame UART asynchrone. Ce mode de communication est privilégié car il permet de transmettre des données logiques (octets) plutôt que des signaux électriques bruts.
- **Rôle de "filtre"** : Ce transfert permet à l'ESP32 de ne pas être sollicité par le décodage complexe du signal infrarouge. Il reçoit directement l'identifiant de la tour, ce qui optimise les ressources du processeur principal.



1. Mesure de la porteuse et robustesse (Canal IR_NEC Telecom)

Le signal émis par la télécommande de simulation est modulé pour garantir une transmission fiable dans un environnement industriel (souvent perturbé par l'éclairage ou d'autres rayonnements).

- Analyse fréquentielle : Comme mesuré sur la capture, la période T est de $26,3 \mu\text{s}$, ce qui correspond à une fréquence porteuse de 38 kHz ($F = 1/T$).
- Rôle du démodulateur : Les récepteurs TSOP (U1, U2) filtrent cette fréquence pour isoler le signal utile du bruit ambiant. Cela garantit que seule l'intention de vidage simulée est enregistrée par le système, ignorant les scintillements des éclairages environnants.

2. Fonctionnement du protocole NEC (Canal IR_NEC)

Une fois le signal démodulé, on obtient une trame propre suivant le standard NEC. Ce protocole repose sur une modulation de distance entre impulsions (Pulse Distance Encoding).

Contrairement à d'autres protocoles, ce n'est pas la durée de l'impulsion qui définit le bit, mais le temps de repos (l'espace) qui la suit.

- Codage des bits : Chaque bit commence par un pulse de $562,5 \mu\text{s}$.
 - Logique 0 : Un espace court de $562,5 \mu\text{s}$ suit l'impulsion.
 - Logique 1 : Un espace long de $1,6875 \text{ ms}$ suit l'impulsion.
- Identification de la donnée : Sur la capture, on identifie la commande $0x15$. Cela correspond à la touche "5" de la télécommande.
- Sécurité par inversion : Le protocole NEC envoie chaque octet suivi de son complément logique (son inverse). L'ATtiny85 compare les deux : si la somme de la commande et de son inverse ne donne pas $0xFF$, la donnée est jugée corrompue et rejetée. Cette sécurité est cruciale pour éviter de fausser les statistiques de collecte de déchets à cause d'un obstacle passager devant le capteur.

3. Traitement et communication série (Canal UART)

L'ATtiny85 sert de processeur d'interface dédié. Il décharge le microcontrôleur principal (XIAO ESP32) de la gestion critique du timing infrarouge, qui demande une précision à la microseconde.

- Conversion de protocole : Une fois la trame NEC validée, l'ATtiny85 convertit l'information en une trame UART. On constate sur la capture que la donnée transmise est bien $0x15$, prouvant l'intégrité du transfert d'information.
- Chronologie du système : On remarque que la trame UART ne commence qu'après la réception complète de la trame IR. Ce délai correspond au temps nécessaire à l'ATtiny pour recevoir le dernier bit de contrôle ("Command inversée") et effectuer la vérification logique avant d'initier la communication avec l'ESP32.

III.4 - Pilotage de la matrice de LED

1. Analyse de l'adressage matériel

L'objectif de cette activité est de concevoir l'interface visuelle de la BOM. La matrice de LED doit servir de guide à l'utilisateur en indiquant, par un code couleur et une animation, vers quel PAV (Point d'Apport Volontaire) il doit diriger son véhicule pour effectuer la collecte. Ce travail a été réalisé en collaboration avec mon binôme, Kevin.

La matrice d'un PAV est composée de 16 LEDs RGB. En analysant le cheminement des données, nous avons constaté que l'adressage physique suit une structure en "serpentin".

Initialement, le code de base de Kevin permettait un allumage séquentiel simple (de 0 à 15). Pour répondre aux besoins du projet, j'ai dû faire évoluer ce programme afin de pouvoir manipuler la matrice non plus comme une ligne de LEDs, mais comme des formes géométriques distinctes (carrés).

Notre matrice de LED sur les PAV possède 16 leds numérotés de manière suivante :

12	13	14	15
11	10	9	8
4	5	6	7
3	2	1	0

2. Création de l'animation de réception

Le but est de créer une réaction visuelle dynamique dès que le PAV commence son envoi de données. Nous avons mis au point un algorithme de "désagrégation" de la matrice. L'animation suit ce cycle :

1. **Allumage total** : La matrice s'allume complètement à la couleur du PAV détecté (ex: Jaune).
2. **Désagrégation extérieure** : Le carré extérieur (le contour) s'éteint.
3. **Désagrégation intérieure** : Le carré central s'éteint à son tour.
4. **Boucle** : La matrice se rallume totalement et recommence le cycle tant que la réception est active.

```
// Tableau répertoriant les indices du carré extérieur (contour)
int contour[] = {0, 1, 2, 3, 4, 7, 8, 11, 12, 13, 14, 15};

// Boucle pour éteindre sélectivement le contour
for (int i = 0; i < 12; i++) {
    pixels.setPixelColor(contour[i], pixels.Color(0, 0, 0));
}
pixels.show();
```

3. Problématique de la restitution des couleurs

La dernière étape consistait à définir les codes couleurs `pixels.Color(R, G, B)` pour chaque type de déchet. Nous avons rencontré une difficulté majeure : la matrice ne se règle pas par "teinte" mais par intensité lumineuse.

Le système de tri repose sur trois couleurs principales pour identifier chaque catégorie de déchets :

- **Jaune** — pour les emballages recyclables (plastiques, métaux, cartons légers).
- **Bleu** — pour les papiers et cartons.
- **Vert** — pour le verre.

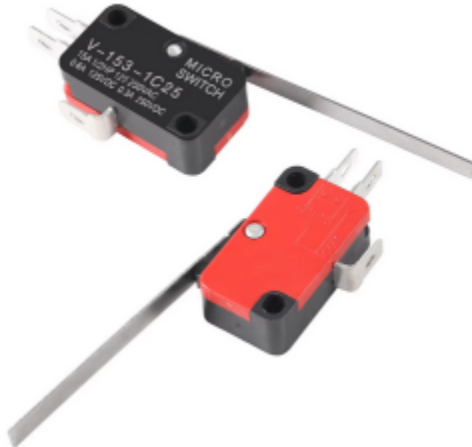
Pour les **ordures ménagères**, deux options restent possibles selon les contraintes techniques de la matrice LED :

- **Marron** : difficile à reproduire sur une LED, car il s'agit d'un orange à très faible luminance ; nous utilisons donc un **orange très sombre** pour s'en approcher.
- **Gris** : simulé par un **blanc de faible intensité**, permettant un rendu visuel proche du gris.

Enfin, la **couleur de la LED du BUS** n'est pas encore déterminée.

III.5 - Test Anti-choc et Rebond

Afin de sensibiliser les jeunes utilisateurs à la sécurité routière, les camions (BOM) sont équipés de capteurs de choc de type **interrupteurs de fin de course**. Chaque impact détecté doit entraîner une pénalité de points. Mon objectif a été de rendre cette détection fiable malgré les contraintes physiques des capteurs mécaniques.



1. Mise en évidence du phénomène de rebond (Bouncing)

Pour tester la réactivité du système, j'ai développé un programme simple d'incrémentation d'une variable à chaque appui sur le GPIO 6.

- **Le problème constaté :** Lors d'un seul appui physique, la variable compteur s'incrémente de plusieurs dizaines d'unités sur le moniteur série.

```

Output: Serial Monitor
Not connected. Select a board and a port to connect automatically.
Choix détecté : Compteur : 1
Choix détecté : Compteur : 2
Choix détecté : Compteur : 3
Choix détecté : Compteur : 4
Choix détecté : Compteur : 5
Choix détecté : Compteur : 6
Choix détecté : Compteur : 7
Choix détecté : Compteur : 8
Choix détecté : Compteur : 9
Choix détecté : Compteur : 10
Choix détecté : Compteur : 11
Choix détecté : Compteur : 12
Choix détecté : Compteur : 13
Choix détecté : Compteur : 14
Choix détecté : Compteur : 15
Choix détecté : Compteur : 16
Choix détecté : Compteur : 17
Choix détecté : Compteur : 18
Choix détecté : Compteur : 19
Choix détecté : Compteur : 20
Choix détecté : Compteur : 21
Choix détecté : Compteur : 22
Choix détecté : Compteur : 23
  
```

Analyse technique : Ce phénomène est dû aux lamelles métalliques du capteur qui, lors du contact, vibrent et créent des micro-coupures électriques extrêmement rapides. L'ESP 32 étant performant, il interprète chaque vibration comme un nouveau choc.

2. Solutions et perspectives : L'anti-rebond (Debounce)

Avec Mohamed, nous avons cherché une solution pour stabiliser la lecture.

- **Utilisation de `millis()`** : Nous avons choisi d'utiliser `millis()` pour créer un délai de garde "non-bloquant". Contrairement à `delay()`, cette fonction permet au reste du programme (gestion des moteurs et des LEDs) de continuer à tourner pendant que l'on attend que le signal du capteur se stabilise.
- **Problématique du relâchement** : Un second rebond apparaît souvent lorsque l'interrupteur "remonte" (le relâchement). Ce signal parasite doit également être filtré.
- **Analyse critique de la solution "2 chocs = 1 impact"** : J'ai proposé cette idée pour diviser l'erreur, mais elle s'avère insuffisante. Si le délai de l'utilisateur est trop long ou si l'on manipule l'interrupteur, le comptage devient imprécis.

3. Analyse comparative des stratégies de détection

À ce stade du projet, deux approches sont envisageables pour la gestion des collisions. Le choix final dépendra du comportement souhaité pour la sensibilisation des utilisateurs :

- **Option A : Stratégie de "Tolérance" (2 chocs = 1 impact)**
 - **Principe** : Le programme nécessite deux détections successives pour valider une pénalité.
 - **Avantage pédagogique** : Cela permet de ne pas pénaliser les enfants pour de simples effleurements ou de petites erreurs de manipulation. Seul un "gros" choc (générant plusieurs rebonds ou une double collision) est sanctionné.
 - **Inconvénient** : Moins de précision technique.
- **Option B : Stratégie de "Précision" (1 choc = 1 impact)**
 - **Principe** : Grâce à un code d'anti-rebond (Debounce) totalement fonctionnel via `millis()`, chaque action mécanique réelle est isolée.
 - **Avantage technique** : C'est la solution la plus rigoureuse. Le système est parfaitement fiable : un contact physique égal une pénalité, sans aucune erreur de comptage due aux vibrations.
 - **Inconvénient** : Peut être perçu comme très sévère par les utilisateurs, la moindre erreur étant immédiatement sanctionnée.

Conclusion de l'étude : Si l'objectif est la précision pure, l'**Option B** est la plus aboutie techniquement. Cependant, l'**Option A** offre une dimension "jeu" plus souple. Pour la Revue 3, nous finaliserons ce choix après une phase de tests réels avec les utilisateurs pour déterminer laquelle de ces deux logiques est la plus adaptée au cahier des charges.

III.6 - Brasage et câblage de la nouvelle carte

Étape d'assemblage et de brasage

Une fois le routage terminé et les cartes reçues, nous passons à l'étape du câblage et de l'assemblage. Pour ce faire, nous utiliserons le matériel suivant :

- **De la pâte à braser** : Un mélange de micro-billes d'étain et de flux qui permet de fixer les composants CMS. (Pâte à Souder 183°C)



- **Un stencil (écran de sérigraphie)** : Une fine plaque métallique découpée au laser, utilisée pour appliquer la pâte à braser avec précision uniquement sur les pastilles (pads) du circuit.



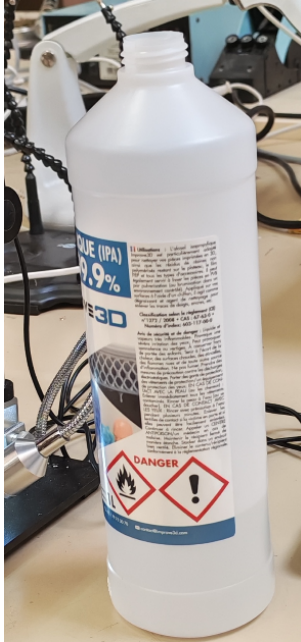
- **Une pince de précision (brucelles)** : Pour déposer les composants délicatement sur la pâte.



- **Une loupe électronique (ou microscope binoculaire) :** Indispensable pour contrôler la qualité des joints de soudure et l'alignement des composants miniatures.



- **Un poste de soudage (fer à souder) :** Utilisé principalement pour souder les **composants traversants** (connecteurs, supports de puces...).
- **alcool isopropylique :** Pour nettoyer la carte ainsi que le stencil



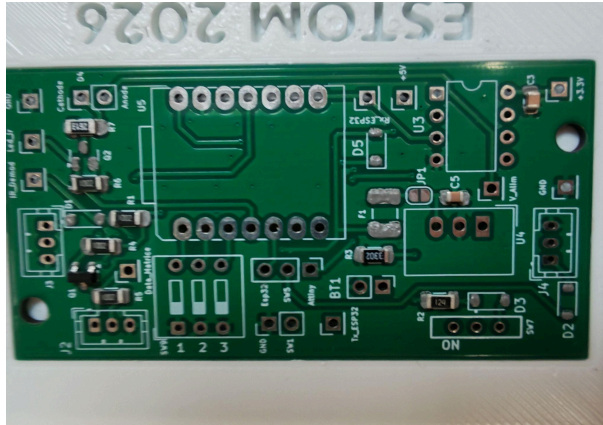
Distinction des composants

Nous distinguons deux types de technologies sur la carte :

1. **Les composants CMS (Composants Montés en Surface)** : Ils sont posés directement sur la surface du circuit imprimé (condensateurs, diodes, résistances).
2. **Les composants traversants** : Leurs broches traversent la carte pour être soudées de l'autre côté (comme l'ATtiny85 ou les connecteurs JST).

Le procédé de brasage

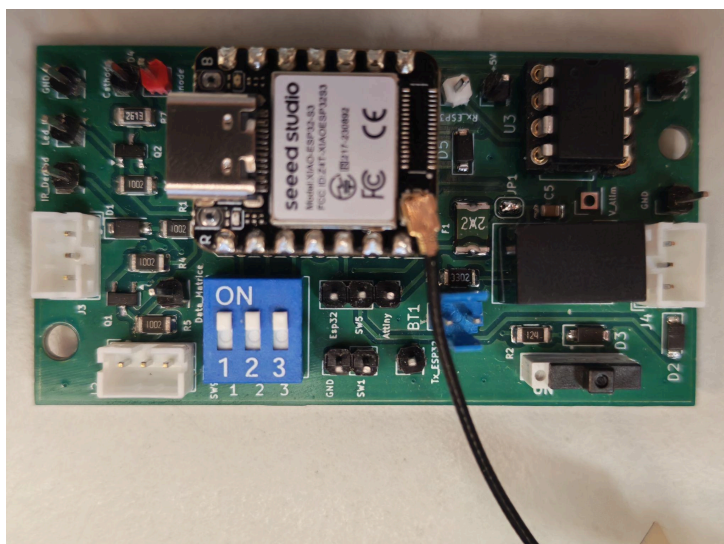
Ces composants sont assemblés par **brasage**. C'est un procédé permanent qui consiste à lier deux pièces métalliques à l'aide d'un métal d'apport (souvent un alliage d'étain). Ce métal d'apport à une température de fusion inférieure à celle des métaux à assembler, ce qui permet de créer une liaison électrique et mécanique solide sans fondre les pistes du circuit ni les pattes des composants.



Le cycle de refusion Une fois les composants CMS déposés avec précision sur la pâte à braser, la carte est placée dans un **four à refusion**. Ce processus permet de liquéfier l'alliage de manière homogène sur l'ensemble du circuit. Pour notre carte ESTOM-2026, nous avons appliqué ce profil thermique spécifique :

- **Température de départ** : 90°C (phase de préchauffage pour évaporer les solvants du flux).
- **Température de pic** : Une montée progressive jusqu'à **275°C** sur une durée totale de **2 minutes 50**.

La carte une fois câblée ressemble à ça :



III.7 Instructions de Câblage et d'Assemblage des Composants

Les composants doivent être câblés et soudés selon les instructions suivantes :

1. Les Circuits Intégrés et Modules

- **U3 (ATtiny85)** : Il est impératif de câbler un support de circuit intégré au préalable. Vérifiez soigneusement le sens du détrompeur du support et de l'ATtiny85 avant l'insertion pour éviter toute inversion.
- **U4 (Régulateur de tension)** : À souder selon l'empreinte dédiée.
- **U5 (XIAO ESP32-S3)** : Comme pour l'ATtiny85, une attention particulière doit être portée au sens d'orientation.

Astuce de soudure : Pour garantir que les connecteurs femelles (headers) soient parfaitement droits et stables, insérez d'abord les picots mâles (pins) dans l'ESP32, emboîtez-les dans les connecteurs femelles, placez le tout sur le PCB, puis soudez.

2. Les Connecteurs et Éléments de Commande

- **J1, J2, J3 (Embases JST)** : * **J1 et J3** : Permettent de connecter les modules démodulateurs infrarouges TSOP38238.
 - **J2** : Permet de raccorder la matrice de LED, que ce soit celle du PAV ou celle de la BOM.
- **SW9 (DIP Switch)** : Interrupteur de configuration servant à paramétrer le comportement du PAV et de la BOM.
- **SW7 (Interrupteur ON/OFF)** : Switch d'alimentation générale. Il permet d'isoler la batterie pour couper complètement le système et éviter toute consommation résiduelle à l'arrêt.
- **F1 (Fusible)** : Composant de protection thermique/surintensité. Il n'a pas de sens de montage (non polarisé).

3. Les Composants Passifs (Condensateurs & Résistances)

Tous les condensateurs et résistances de cette liste sont **non polarisés** et peuvent être soudés dans n'importe quel sens.

- **Condensateurs** :
 - **C1, C2** : 2,2μF
 - **C3** : 100nF
 - **C4** : 10nF
 - **C5, C6, C7** : 1μF
- **Résistances** :

- **R1, R4, R5, R6, R8** : 10k Ω
- **R2** : 120k Ω
- **R3** : 33k Ω
- **R7** : 274k Ω

4. Les Semi-conducteurs Polarisés (Diodes, LED, Transistors)

Ces composants ont un sens de montage strict. Une inversion peut bloquer le signal ou endommager le circuit.

- **Diodes (D1, D2, D3, D6)** : Repérez la bague sérigraphiée (Cathode) sur le composant :
 - **D1** : Cathode orientée vers le haut.
 - **D2** : Cathode orientée vers la droite.
 - **D3** : Cathode orientée vers le haut.
 - **D6** : Cathode orientée vers la gauche.
- **D4 (LED Infrarouge)** : Utilisée par le PAV pour émettre les signaux IR. La patte la plus longue (Anode) va sur le pôle positif.
- **Q1, Q2, Q3 (Transistors MOSFET)** : À souder en respectant le sens du boîtier (méplat ou détrompeur selon le modèle choisi).

5. Correspondance des Points de Test (TP) et Jumpers (SW)

Pour faciliter le débogage et le routage, les cavaliers (SW1 à SW5) et les points de test (TP1 à TP11) sont assignés aux signaux du circuit dans l'ordre alphabétique/numérique suivant :

Composant(s)	Signal Associé	Fonction
SW1 / TP1	+3.3V	Alimentation logique des capteurs IR et de l'ATtiny85.
SW2 / TP2	+5V	Alimentation principale de la matrice LED et du XIAO.

SW3 / TP3	Image_A_25m	Signal de mesure du niveau de la batterie (via le pont diviseur).
SW4 / TP4	Data_Matrice	Ligne de données numériques envoyée à la matrice LED.
SW5 / TP5	GND	Masse de référence (0V) de la carte.
TP6	IR_Demod	Signal infrarouge brut reçu par les TSOP.
TP7	Inter_fin_course	Entrée du capteur de fin de course pour détecter les chocs.
TP8	Led_Ir	Commande de la LED de puissance émettrice IR.
TP9	Rx_ESP32	Signal série envoyé par l'ATtiny85 vers le XIAO ESP32.
TP10	Select. dest. IR Demod	Sert d'aiguillage pour choisir le chemin du signal infrarouge sur la carte.
TP11	Tx_ESP32	Sortie de transmission série du XIAO ESP32

III.8 Mise en Œuvre et Configuration de la Carte

Une fois le soudage de la carte entièrement finalisé, suivez rigoureusement les étapes suivantes pour valider son bon fonctionnement et la configurer selon l'application finale.

1. Procédure de Premier Allumage et Vérification (Étape Commune)

1. **Programmation initiale** : Configurez et téléversez les programmes respectifs dans l'ATtiny85 et le XIAO ESP32-S3 au préalable sur votre ordinateur.
2. **Insertion des composants** : Installez les deux microcontrôleurs dans leurs emplacements dédiés (**U3** pour l'ATtiny85 et **U5** pour l'ESP32). Veillez à respecter scrupuleusement le sens des détrompeurs.
3. **Alimentation de la carte** : Raccordez une source d'alimentation sur l'entrée de la carte.
Note technique : Pour que le régulateur de tension (**U4**) fonctionne correctement et fournisse un signal stable, la tension d'entrée brute (V_{Alim}) doit être **constante et supérieure à 6 V**.
4. **Contrôle des tensions (Sécurité)** : À l'aide d'un multimètre configuré en voltmètre continu, utilisez les points de test (**TP**) pour valider les tensions du circuit :
 - Mesurez entre **TP2** +5V et **TP5** (GND) pour valider le rail 5V.
 - Mesurez entre **TP1** (+3.3V) et **TP5** (GND) pour valider le rail 3,3V.
5. **Extinction** : Une fois les tensions validées, coupez immédiatement l'alimentation générale via l'interrupteur **SW7** avant de procéder aux raccordements spécifiques.

2. Configuration Spécifique selon l'Application

Basculez et câbler la carte selon l'un des deux modes ci-dessous en fonction de votre besoin :

Appliquée à une BOM (Benne à Ordures Ménagères)

Dans cette configuration, la carte agit principalement comme un récepteur de signal et un afficheur compact.

- **Raccordement des récepteurs** : Connectez les modules démodulateurs infrarouges TSOP38238 sur les embases **J3** et **J4**.
- **Raccordement de l'affichage** : Connectez la matrice de **7 LED** dédiée à la BOM sur l'embase **J2**.
- **Routage du signal** : Positionnez le cavalier (jumper) sur le commutateur **SW5** afin d'aiguiller correctement la liaison entre la broche de l'ATtiny85 et le signal démodulé.

- **Paramétrage** : Ajustez le DIP switch **SW9** sur la position correspondante au profil de la BOM.

Appliquée à un PAV (Point d'Apport Volontaire)

Dans cette configuration, la carte gère l'émission du signal et pilote un affichage plus étendu.

- **Gestion des embases** : Laissez les embases **J3** et **J4** vides (elles ne sont pas utilisées dans ce mode).
- **Raccordement de l'affichage** : Connectez la matrice de **16 LED** dédiée au PAV sur l'embase **J2**.
- **Émission infrarouge** : Assurez-vous de la présence de la LED émettrice infrarouge **D4**, qui sera utilisée par le système pour envoyer les données.
- **Paramétrage** : Ajustez le DIP switch **SW9** sur la position correspondante au profil du PAV.

3. Mise en marche finale

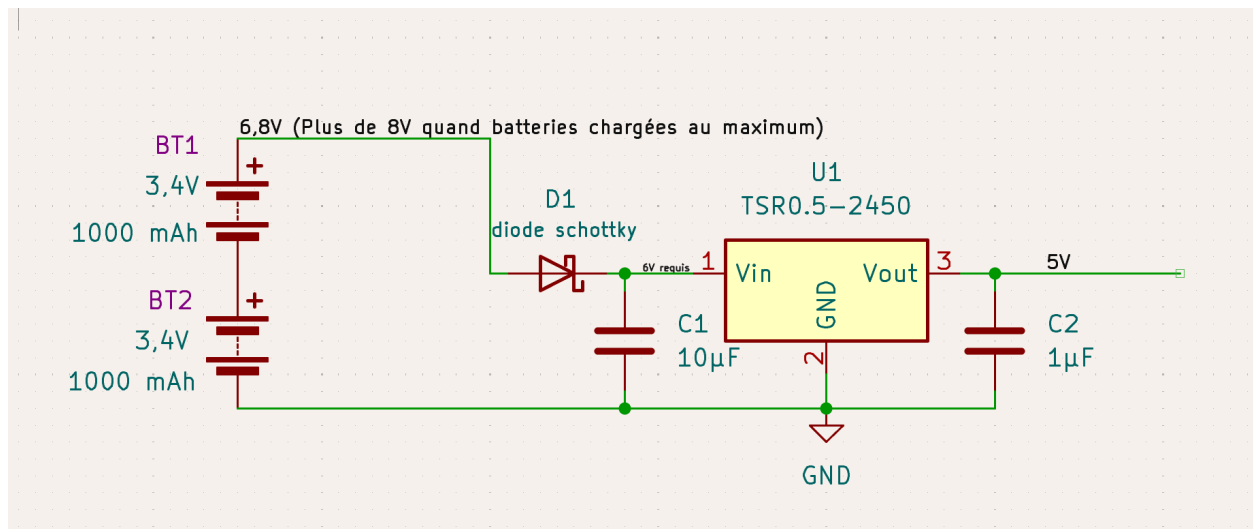
Une fois les périphériques connectés selon votre mode (PAV ou BOM), basculez l'interrupteur général **SW7** sur **ON**. Le système est prêt à fonctionner.

III.9 - Le choix de la batterie et les problèmes rencontrés

Les tests de la solution initiale ont révélé que la pile 9V (6LR61) n'était pas adaptée aux besoins de la BOM. Bien que sa tension soit correcte, sa **capacité très limitée (250 mAh)** fait en sorte que la pile se décharge très rapidement lors des tests.

Solution retenue : Pack LiPo 2S (6,8V / 1000 mAh) Pour garantir un fonctionnement fiable, nous avons opté pour un assemblage de deux cellules Lithium-Polymère (LiPo) montées en série.

- **Capacité** : Avec **1000 mAh**, nous multiplions par **4 fois** l'autonomie du système par rapport à la pile 9V.
- **Tension et sécurité de régulation** : Chaque cellule fournit une tension de **3,4V**, soit un total de **6,8V** pour le pack. Ce choix est stratégique pour le fonctionnement du régulateur **TSR 1-2450 (U4)** :
 - Le régulateur a besoin d'un minimum de **6,0V** en entrée pour garantir une sortie stable à **5,0V**.
 - Une batterie voit sa tension chuter durant sa décharge. En partant de **6,8V**, même lorsque les cellules s'épuisent et perdent quelques dixièmes de volt, la tension totale reste au-dessus du seuil critique de **6,0V**.
 - Si nous avions utilisé des cellules d'une tension plus faible, le régulateur "décrocherait" avant même que la batterie ne soit vide, rendant une partie de l'énergie (les mAh) inutilisable.



III.10 Guide de Dépannage et Diagnostic (Troubleshooting)

Si la carte ne fonctionne pas après l'allumage, suivez cette procédure de diagnostic par étapes pour identifier et corriger la panne.

Étape 1 : Vérification de la chaîne d'alimentation générale

La majorité des pannes électroniques proviennent d'un problème d'alimentation.

1. **Contrôle visuel du switch** : L'interrupteur général SW7 est-il bien positionné sur ON ?
2. **Mesure de la batterie** : À l'aide d'un multimètre, mesurer la tension aux bornes de la batterie ou sur le point de test TP4 (+V_Alimentation). Si la tension est inférieure à $3,3\text{ V}$, le régulateur ne pourra pas fonctionner : rechargez ou remplacez la batterie.
3. **Contrôle du fusible** : Si la batterie est bonne mais qu'aucune tension n'est présente après SW7, vérifiez le fusible F1. S'il est coupé (fondu), cherchez un court-circuit sur la carte avant de le remplacer.
4. **Validation des rails de tension** : Mesurez les tensions de référence sur les points de test dédiés :
 - TP2 ou TP5 pour le 5 V : Vous devez mesurer une valeur proche de 5 V . Si vous avez 0 V , le régulateur U4 (TSR 1-2450) est peut-être mal soudé ou défectueux.
 - TP1 pour le $+3,3\text{ V}$: Vous devez mesurer $3,3\text{ V}$. Cette tension est générée par le XIAO ESP32 (U5). Si le 5 V est présent mais que le $3,3\text{ V}$ est absent, vérifiez l'orientation de l'ESP32 ou s'il est correctement enfoncé dans ses rails.

Étape 2 : Dysfonctionnement de la BOM (Problème de Réception IR)

Si la carte s'allume mais ne réagit pas aux commandes de la télécommande infrarouge :

1. **Vérification mécanique** : Assurez-vous que les démodulateurs TSOP38238 sont bien connectés dans les embases J3 et J4 (et non pas inversés avec J2).
2. **Test du signal brut (TP6)** : Placez la pointe d'un oscilloscope ou d'un analyseur logique sur TP6 (IR_Demod). Pressez un bouton de la télécommande.
 - *Si le signal reste plat à $3,3\text{ V}$* : Le problème vient du TSOP ou de la télécommande (pile HS).
 - *Si le signal descend vers 0 V lors des appuis* : Le TSOP fonctionne, passez à la suite.

3. **Contrôle du Cavalier SW5** : Vérifiez la présence et la bonne position du cavalier sur le sélecteur mécanique SW5 (Select. dest. IR Demod). Sans ce cavalier, le signal du TSOP n'atteint jamais l'ATtiny85.
4. **Vérification de la liaison ATtiny85 ESP32** : Utilisez un analyseur sur TP9 (Rx_ESP32) pour voir si l'ATtiny transmet bien les ordres traduits à l'ESP32. Si rien ne sort, l'ATtiny85 (U3) est peut-être mal programmé ou monté à l'envers sur son support.

Étape 3 : Dysfonctionnement du PAV (Problème d'Émission ou d'Affichage)

Si la carte est configurée pour le PAV mais que rien ne s'affiche ou n'est émis :

1. **Problème d'émission IR (D4)** : Si le PAV n'émet rien, testez le signal de commande sur TP8 (Led_Ir).
 - Si le signal oscille mais que la LED reste éteinte, vérifiez le sens de soudure de la LED infrarouge D4 (Anode/Cathode) et du transistor MOSFET Q2 (SN7002).
 - *Astuce* : Regardez la LED D4 à travers l'appareil photo d'un smartphone, vous devriez voir des lueurs violettes lors de l'émission.
2. **Problème d'affichage de la matrice (J2)** : Si la matrice de LED (7 ou 16 LED) reste éteinte :
 - Vérifiez la tension sur la broche 1 de J2 ($+5\text{V}$). Les matrices consomment beaucoup et dépendent directement de la qualité du rail 5V .
 - Suivez le signal sur TP4 (Data_Matrice) pour vous assurer que l'ESP32 envoie bien les données de commande à la matrice via le transistor de décalage de niveau Q1.

Étape 4 : Comportement erratique ou mauvais profil (DIP Switch)

Si la carte fonctionne mais n'adopte pas le bon comportement visuel ou applicatif :

1. **Vérification du SW9** : Contrôlez les mini-interrupteurs du DIP Switch SW9. Une mauvaise combinaison peut forcer l'ESP32 à exécuter le code de la BOM à la place du PAV, ou inversement.
2. **Vérification des résistances de Pull-up** : Si l'ESP32 ignore la position de SW9, vérifiez à l'aide d'un ohmmètre (carte éteinte) que les résistances de rappel R1, R4, R5, R6 mesurent bien 10k et qu'il n'y a pas de court-circuit avec la masse.

III.11 - Fiche de recette

Nom	Référence	Quantité	Value	Code-Commande	Prix par qdt (TTC)					1,1
Support batterie Embase	BT1	1	Battery	RS 681-1871	0,968					
Condensateurs céramique multicouche 2.2µF	C1,C2	2	2.2µF	RS 242-1135	0.49					
Condensateurs céramique multicouche 100µF	C3	1	100µF	RS 264-4422	0,265					
Condensateurs céramique multicouche 10µF	C4	1	10µF	RS 103-4164	0,614					
Condensateurs céramique multicouche 1µF	> C5-C7	3	1µF	RS 264-4450	0,156					
Diode Nexperia Diode Schottky Simple	> D1-D3	3	PMEG3010ER	RS 816-6855	0,789					
Emetteur infrarouge, Haute puissance, 950 nm	D4	1	SFH 4547	Farnell 4332518	0,894					
Fusible réarmable Littelfuse	F1	1	400mA 0.2A	RS 517-7105	0,497					
B2B-PH-K-S	J1	1	Embase JST PH 2mm 2cts	Farnell 9491856	0,10176					
	> J2, J4	2	Embase JST PH 2mm 3cts	-- valeurs mixtes --	?					
	J3	1	Embase JST PH 2mm 3 cts	?	?					
	JP1	1	SolderJumper_2_Open	?	?					
SN7002NH6327XTSA2	> Q1, Q2	2	SN7002	Mouser 726-SN7002NH6327XTSA	0,46					
Résistance de 10 kΩ	> R1, R4-R6	4	10K	RS 125-1192	0,116					
Résistance de 120 kΩ	R2	1	120K	RS 807-1110	0,028					
Résistance de 33 kΩ	R3	1	33K	RS 679-2055	0,061					
Résistance de 274 kΩ	R7	1	274K	RS 679-1960	0,014	(pas disponible)				
Connecteur HE14 MH100	SW1	1	Inter_fn_course	Gotronic 08000	0,6					
Connecteur HE14 MH100	SW5	1	Select_dest_IR_Demod	Gotronic 08000	0,6					
Commutateur SIP glissière EOZ	SW7	1	SIP 1K2 EAO	RS 204-7865	1,88					
Connecteur HE14 MH100	SW9	1	SW_DIP_x03	Gotronic 07222	0,6					
Connecteur HE14 MH100	TP1	1	IR_Demod	Gotronic 08000	0,6					
Connecteur HE14 MH100	> TP2, TP3	2	GND	Gotronic 08000	0,6					
Connecteur HE14 MH100	TP4	1	V_Alim	Gotronic 08000	0,6					
Connecteur HE14 MH100	TP5	1	Tx_ESP32	Gotronic 08000	0,6					
Connecteur HE14 MH100	TP6	1	5V	Gotronic 08000	0,6					
Connecteur HE14 MH100	TP7	1	3.3V	Gotronic 08000	0,6					
Connecteur HE14 MH100	TP8	1	Data_Matrice	Gotronic 08000	0,6					
Connecteur HE14 MH100	TP9	1	Led_Ir	Gotronic 08000	0,6					
Connecteur HE14 MH100	TP10	1	Rx_ESP32	Gotronic 08000	0,6					
Démodulateur INFRAROUGE	> U1, U2	2	TSOP382xx	RS 708-5086	1,652					
Microcontrôleur ATtiny85	U3	1	ATtiny85	RS 133-1672	1,8					
Régulateur à découpage TRACOPOWER	U4	1	TSR 1-2450	RS 666-4379	1,364					
Carte XIAO ESP32-S3 113991114	U5	1	XIAO-ESP32S3	Gotronic 38192	8,6					
Prix de la carte à commander chez JLCPCB					5					
						Total en € TTC	32,94976			

Le tableau ci-dessus récapitule l'ensemble des composants nécessaires à la fabrication d'une unité de la carte ESTOM-2026.

Analyse financière :

- **Coût des composants** : Le total des composants électroniques s'élève à **27,95 € TTC**. On note que les éléments les plus onéreux sont les microcontrôleurs (**XIAO ESP32** à 8,60 € et **ATtiny85** à 1,80 €) ainsi que les capteurs infrarouges **TSOP**.
- **Fabrication du PCB** : Le prix de fabrication de la carte chez le prestataire (JLCPCB) est de **5,00 €**.
- **Prix de revient total** : Le coût final de la carte équipée est de **32,95 € TTC**.

Partie Personnelle ER2: PAV/Abribus

PIRES SALVADOR Kevin



I - Présentation générale

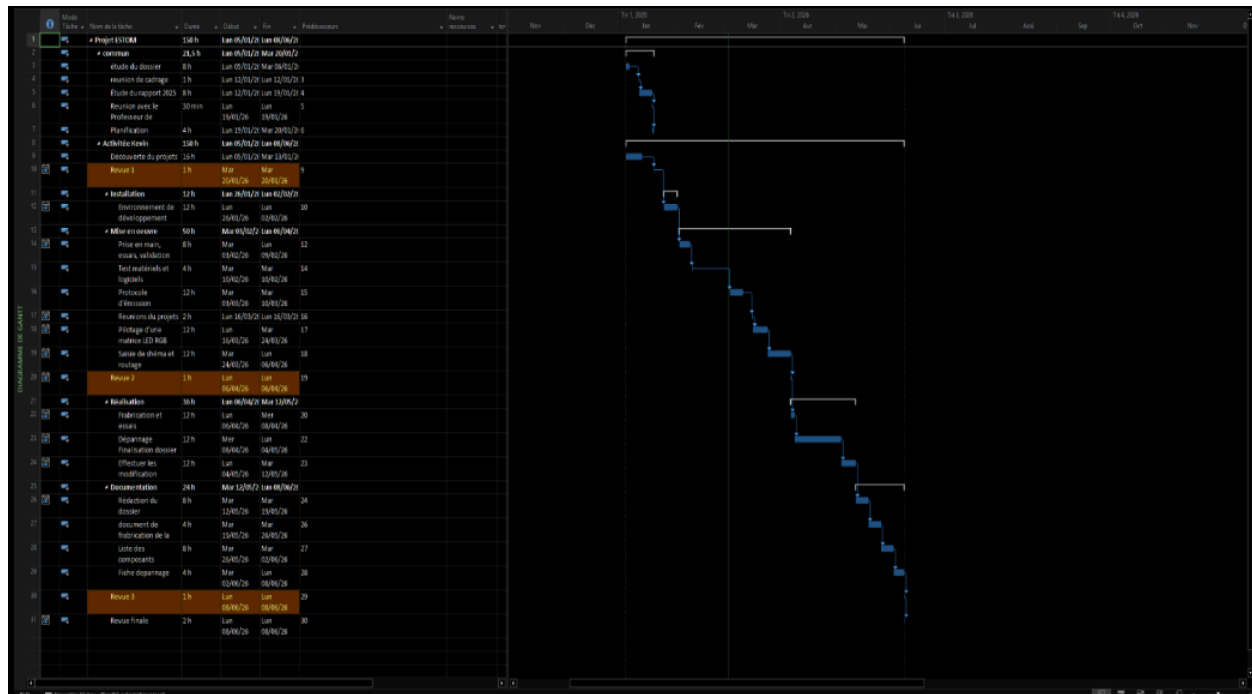
Pour cette deuxième année de BTS, j'ai été affecté au domaine de la réalisation physique des PAV (Points d'Apport Volontaire) pour le projet ESTOM (Espace de Sensibilisation au Tri des Ordures Ménagères). Je suis chargé de la partie électronique des PAV et de l'abribus. Mon objectif est d'optimiser l'autonomie de la batterie pour qu'elle tienne toute la journée et de m'assurer que les PAV communiquent correctement avec les camions-bennes à ordures ménagères (BOM). Enfin, je dois programmer l'extinction des LED lorsqu'une partie se termine ou dès que le camion concerné reste plus de 5 secondes devant le PAV. Ce même objectif est dédié à l'abribus avec le passage du bus.

I.1 - Objectif technique

- J'ai pu réaliser l'allumage des LED pour vérifier le bon fonctionnement de la carte et de l'ESP32-S3.
- J'ai effectué des tests sur un prototype de PAV pour valider le fonctionnement des LED, leur ordre de câblage ainsi que la matrice de LED.
- Je vais devoir retirer quelques composants du routage ainsi que dans le schéma de câblage pour optimiser la solution.

II - Organisation du travail

II.1 - Diagramme de Gantt prévisionnel



Pour garantir le respect des échéances de la revue de projet n°1, j'ai mis en place un planning prévisionnel à l'aide du logiciel Microsoft Project. Cet outil me permet de visualiser l'enchaînement des tâches et d'identifier le "chemin critique" pour ne pas prendre de retard sur la fabrication de la carte électronique.

II.2 - Observation et compréhension de la carte

L'exploitation du dossier de fabrication de l'année précédente a été essentielle pour assurer la continuité du projet. Cette étape de rétro-conception a permis de repérer les ressources matérielles disponibles (brochage, bus de données) et de planifier les modifications nécessaires sur le nouveau circuit.

II.3 - Partie teste des LED avec Arduino IDE

```
for (int i = 0; i < NUMPIXELS; i++) {  
    pixels.setPixelColor(i, pixels.Color(0, 1, 0)); // vert  
}  
pixels.show();  
delay(1000);
```

```
// Éteindre le contour
int contour[] = {0,1,2,3,4,7,8,11,12,13,14,15};
for (int i = 0; i < 12; i++) {
    pixels.setPixelColor(contour[i], pixels.Color(0, 0, 0));
}
pixels.show();
delay(1000);

// Éteindre le centre
int centre[] = {5,6,9,10};
for (int i = 0; i < 4; i++) {
    pixels.setPixelColor(centre[i], pixels.Color(0, 0, 0));
}
pixels.show();
delay(2000);
}
```

J'ai fait plusieurs code différents

Le premier et le code pour éteindre le contour en premier puis l'extérieur

Cette partie du code permet d'éteindre le contour donc les led 0,1,2,3,4,7,8,11,12,13,14,15
Avec `int contour[] = {toute les led qu'on veut éteindre}` ceci sert a creer une variable

```
pixels.setPixelColor(contour[i], pixels.Color(0, 0, 0));
```

La boucle `for (int i = 0; i < 12; i++)` va parcourir le tableau de la case 0 à la case 11. À chaque tour, elle prend le numéro de la LED stockée dans `contour[i]` pour lui donner la couleur (0,0,0), c'est-à-dire "éteint".

```
pixels.show();
```

La fonction `pixels.show()` est indispensable pour transférer les données de couleur stockées dans la mémoire tampon (buffer) de l'ESP32 vers la matrice physique. Elle permet une mise à jour instantanée et synchronisée de l'ensemble des 16 LEDs.

```
for (int i=0; i<NUMPIXELS; i++) {
```

L'utilisation d'une boucle `for` permet d'automatiser le traitement de la matrice de LEDs. En incrémentant l'indice `i`, le programme applique la configuration de couleur à chaque pixel successivement de manière optimisée, ce qui rend le code plus court et plus facile à maintenir.

II.4 - Code pour le DIP SWITCH

```
#define PIN
```

```

#define DIP1 1
#define DIP2 2
#define DIP3 3
#define NUMPIXELS 16

pixels.begin();
pinMode(DIP1, INPUT_PULLUP);
pinMode(DIP2, INPUT_PULLUP);
pinMode(DIP3, INPUT_PULLUP);

}

uint32_t getColorFromDip() {

    bool s1 = !digitalRead(DIP1); // inversion à cause du pull-up
    bool s2 = !digitalRead(DIP2);
    bool s3 = !digitalRead(DIP3);

    int value = (s3 << 2) | (s2 << 1) | s1;

    switch (value) {
        case 0: return pixels.Color(0, 0, 0); // eteint
        case 1: return pixels.Color(4, 0, 0); // Rouge
        case 2: return pixels.Color(0, 4, 0); // Vert
        case 3: return pixels.Color(0, 5, 2); // Bleu
        case 4: return pixels.Color(9, 1, 0); // Orange
        case 5: return pixels.Color(10, 0, 10); // Magenta
        case 6: return pixels.Color(0, 0, 0); // Cyan
        case 7: return pixels.Color(0, 0, 0); // Blanc
    }
    return pixels.Color(0, 0, 0);
}

```

return pixels.Color(0, 0, 0);

Pour configurer le comportement des LEDs, j'utilise un DIP Switch relié aux entrées de l'ESP32. Ces entrées sont configurées en mode Pull-up interne.

Le Pull-up : Cette configuration impose un niveau logique haut (1) par défaut lorsque l'interrupteur est ouvert. Lorsqu'on actionne le switch, le niveau bascule à 0.

L'inversion logique (!) : Afin de faciliter la manipulation des données dans le programme, j'ai utilisé l'opérateur de négation.

La ligne `bool s1 = !digitalRead(DIP1);` permet donc de traduire l'état physique en logique positive :

Si le switch est activé, s1 devient True (1).

Si le switch est désactivé, s1 devient False (0).

Cela rend le traitement binaire qui suit (`s3 << 2 | s2 << 1 | s1`) beaucoup plus intuitif.

`case 0: return pixels.Color(0, 0, 0);`

Pour associer chaque combinaison binaire des interrupteurs à une couleur spécifique, j'ai implémenté une structure de contrôle switch. Cette méthode est plus performante et plus lisible qu'une suite de conditions if...else.

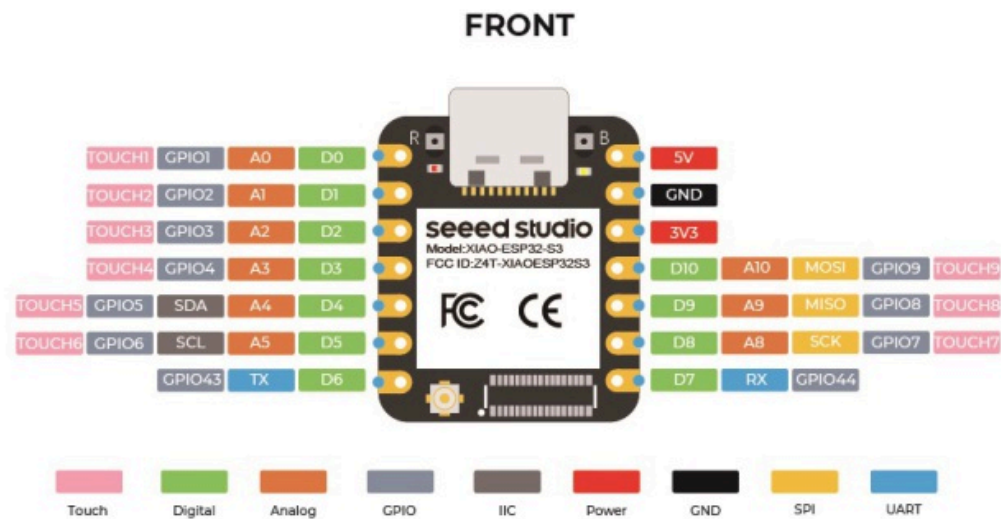
L'instruction case 0 : Elle correspond à l'état où la valeur décimale issue des entrées numériques est nulle (soit la combinaison binaire 000).

L'action `return pixels.Color(0, 0, 0)` : Dans ce cas précis, la fonction renvoie la valeur chromatique "Noir".

Résultat concret : Lorsque l'utilisateur place tous les switches sur la position OFF, la matrice de LEDs s'éteint totalement. C'est la configuration de repos du système.

II.5 - Présentation de l'équipement

Une présentation rapide de L'esp-32S3



Le choix de L'esp a ces avantages. Son prix est très abordable entre 5-8€, Il a un nombre de GPIO nombreux, Le mode deep sleep permet à L'ESP d'économiser sur la durée de vie de la batterie en une journée car il s'éteint quand il est inactif, un mode sans fils et bluetooth est intégrée directement à l'esp

Matrice de LED 4x4



Le choix de la matrice de led 4x4 a l'avantage de moins consommer mais nous pouvons aussi faire comme le code la montrer faire en sorte que l'extérieur s'éteigne puis l'intérieur.... Ce fait beaucoup ressemblance à un camion qui vide une poubelle

Les LED sont positionnés comme ceci

12	13	14	15
11	10	9	8
4	5	6	7
3	2	1	0

Annexe

ANNEXE IR1

Annexe 1: Code estomBOM (estomBOM.ino):

```

////////////////////////////////////
//
//  PROGRAMME BOM (ESTOM 2026)
//  V5 - NeoPixel Adafruit & Protocole JSON v1.6
//  v2.0 Réseau OK, IR ok Choc à tester
//  v2.1 Version avec affichage debug
//  v2.2 Modif champs JSON et INIT
//  v2.3 Amélioration CCapteurChocs (gestion des rebonds)
//  v2.4 Amélioration durée transfert
//  v2.5 Intégration classe CBatterie
//  v2.6 Corrections bug transferts
//  v2.7 Ajout JSON de debug
//  v2.8 Ajustement de l'affichage LED
//  v2.9 Changement password WIFI ESTOM2026
//  v3.0 19/04/2026 Corrections bug affichage LED
//  v3.1 28/04/2026 Correction affichage LED départ
//  v3.2 05/05/2026 Corr bug couleur BOM
//  18/05/2026 v3.3 Corr couleur orange devient rouge
//  19/05/2026 v3.4 Couleur du bus cyan, ajustement
//
////////////////////////////////////
#define VER "3.3"

#include <Arduino.h>
#include <WiFi.h>
#include <WiFiClient.h>
#include <ArduinoJson.h>

#include "definitions.h"
#include "ccapteurchocs.h"
#include "cdipswitch.h"
#include "cneopixel.h"
#include "cbatterie.h"

////////////////////////////////////
// VARIABLES ET INSTANCIATION GLOBALES
////////////////////////////////////
CDipSwitch ds;
CCapteurChocs cc;
CBatterie batt; // pas de setup
CNeoPixel afficheur(NUM_LEDS, LED_PIN, DELAYVAL);

```

```

WiFiClient clientTcp;

// PARAMS WiFi
const char *ssid = "AP-ESTOM";
const char *password = "ESTOM2026";
// Adresse IP et port du serveur TCP
const char *tcpAddress = "192.168.0.1";
const uint16_t tcpPort = 5005;

// etats et valeurs de l'application
int g_dsCouleur=0; // valeur des DIP switches
int g_type=0; // type de BOM
int g_adrIpPav; // dernier octet de l'adresse IP du PAV
int g_luminosite=1; // 1 faible ou 2 moyen ou 3 fort
int g_dep_time=0;
int g_dep_dureeEntreDeuxBytes=0;
int g_dep_pausePourReprise=0;
int g_deltaT=0;
bool g_batterie_faible=false;
int g_c=0;
String g_message="";

T_ETATSBOM _etatBOM = S_INIT;

/////////////////////////////////////////////////////////////////
void connectToWiFi() {
    Serial.print("Connexion au WiFi");
    WiFi.begin(ssid, password);
    unsigned long startAttemptTime = millis();

    while (WiFi.status() != WL_CONNECTED && millis() - startAttemptTime < 10000)
    {
        delay(1000);
        Serial.print(".");
    } // wh

    if (WiFi.status() == WL_CONNECTED) {
        Serial.println("\nConnecté au WiFi");
        Serial.print("Adresse IP locale : ");
        Serial.println(WiFi.localIP());
    } else {
        Serial.println("\nÉchec de connexion WiFi. RESTART !");
        delay(2000);
        ESP.restart();
    } // else
}

/////////////////////////////////////////////////////////////////
/

```

```

bool connectToServer() {
    Serial.println("Connexion au serveur TCP...");
    if (clientTcp.connect(tcpAddress, tcpPort)) {
        Serial.println("Connexion TCP réussie.");
        return true;
    } else {
        Serial.println("Échec de connexion TCP.");
        return false;
    } // else
}

///////////////////////////////////////////////////////////////////
/
void sendMessageToServer(T_TYPESTRAMESEND typeTrame) {
    if (!clientTcp.connected()) {
        Serial.println("Connexion perdue pour envoi, tentative de
reconnexion...");
        if (!connectToServer()) {
            Serial.println("Échec de reconnexion pour envoi.");
            return;
        } // if connect
    } // if connect

    DynamicJsonDocument doc(256);
    // TOUTES LES VALEURS DOIVENT ETRE DES STRING (v1.6)
    doc["status"] = String(typeTrame);

    switch(typeTrame) {
        case E_BONJOUR:
            doc["type"] = (g_type==0?"BOM":"BUS");
            doc["couleur"] = String(g_dsCouleur);
            break;

        case E_DEB_TRANSFERT:
            doc["ipPAV"] = String(g_adrIpPav); break;

        case E_FIN_TRANSFERT:
            doc["leds"] = String(afficheur.progression());
            doc["ipPAV"] = String(g_adrIpPav);
            break;

        case E_ANNULATION_TRANSFERT:
            doc["leds"] = String(afficheur.progression());
            doc["ipPAV"] = String(g_adrIpPav);
            break;

        case E_CHOC:
            doc["collisions"] = String(cc.getNbChocs()); break;
    }
}

```

```

        case E_DIVERS: // Pour des messages de debug car pas de voie série sur le
BOM
            doc["texte"] = g_message; break;
        } // sw

        serializeJson(doc, clientTcp); // envoi TCP vers serveur
        clientTcp.println();

        Serial.print("Infos envoyées vers le serveur : ");
        serializeJson(doc, Serial);
        Serial.println();
    }

    ///////////////////////////////////////////////////////////////////
    ///////////////////////////////////////////////////////////////////
void setup() {
    Serial.begin(9600); // vitesse obligatoire pour recevoir le caractère du uC
    Serial.print("estomBOM v");
    Serial.println(VER);

    // lecture des sw au démarrage seulement
    ds.setup();
    g_dsCouleur = ds.getDsCouleur();
    g_type = ds.getType();
    Serial.print("Valeur de couleur détectée : ");
    Serial.println(g_dsCouleur);
    Serial.print("Type détectée : ");
    Serial.println(g_type);

    // batterie
    float nivBatt = batt.getValue();
    g_batterie_faible = (nivBatt<=SEUIL_BATTERIE_FAIBLE?true:false);
    Serial.printf("Batterie : %f\n", nivBatt);

    // init couleur
    Serial.print("Luminosité de départ : ");
    Serial.println(g_luminosite);

    // init et clignotement bandeau LED (NeoPixel)
    afficheur.begin(g_type);
    afficheur.off();
    for(int i=0 ; i<5 ; i++) {
        afficheur.on(g_dsCouleur, g_luminosite, g_batterie_faible);
        delay(500);
        afficheur.off();
        delay(500);
    } // for

    // connexion WIFI

```

```

connectToWiFi();
// connexion au serveur TCP
while (!connectToServer()) {
    if (WiFi.status() != WL_CONNECTED) {
        Serial.println("\nÉchec de connexion WiFi. RESTART !");
        delay(2000);
        ESP.restart();
    } // if Wifi
    delay(2000);
} // wh

sendMessageToServer(E_BONJOUR);

for(int i=0 ; i< NUM_LEDS ; i++) {
    afficheur.setProgression(g_dsCouleur, g_luminosite, i+1,
g_batterie_faible);
    delay(200);
} // for
afficheur.off();
Serial.println("Fin du SETUP");
}

////////////////////////////////////
////////////////////////////////////
void loop() {

    if (!clientTcp.connected()) {
        Serial.println("\nÉchec de connexion au serveur. RESTART !");
        ESP.restart();
    } // if tcp connected

    // test niveau batterie
    float nivBatt = batt.getValue();
    g_batterie_faible = (nivBatt<=SEUIL_BATTERIE_FAIBLE?true:false);

    // =====
    // LECTURE TRAME TCP (SUPERVISION)
    // =====
    if (clientTcp.available()) {
        String trameRec = clientTcp.readStringUntil('\n');
        Serial.println(trameRec);
        StaticJsonDocument<512> data;
        DeserializationError error = deserializeJson(data, trameRec);
        if (error) {
            Serial.print("Erreur JSON : ");
            Serial.println(error.f_str());
            return;
        } // if error
        if (!data.containsKey("ordre")) {

```

```

    Serial.println("Trame invalide (pas de clé 'ordre').");
    return;
} // if data

int ordre = data["ordre"].as<String>().toInt();
Serial.print("Ordre reçu : "); Serial.println(ordre);

int etatJeu;
int nbColls;
int nbLeds;
switch (ordre) {
    case R_INIT:
        Serial.println("Trame INIT reçue");
        etatJeu = (data["etatJeu"].as<String>().toInt());
        _etatBOM = (etatJeu==S_JEUENCOURS?S_JEUENCOURS:S_INIT);
        if (_etatBOM != S_INIT) {
            cc.setup();
        } // if _etatBOM
        nbColls = data["collisions"].as<String>().toInt();
        nbLeds = data["leds"].as<String>().toInt();
        Serial.printf("coll=%d, leds=%d\n", nbColls, nbLeds);
        cc.setNbChocs(nbColls);
        afficheur.setProgression(g_dsCouleur, g_luminosite, nbLeds,
g_batterie_faible);
        break;

    case R_DEBUT:
        Serial.println("Trame DEBUT PARTIE reçue");
        _etatBOM = S_JEUENCOURS;
        if (data.containsKey("luminosite")) {
            g_luminosite = data["luminosite"].as<String>().toInt();
        } // if lum
        cc.setup();
        //afficheur.off();
        afficheur.setProgression(g_dsCouleur, g_luminosite, 0,
g_batterie_faible);
        cc.setNbChocs(0); // RAZ du nombre de chocs
        break;

    case R_FIN:
        Serial.println("Trame FIN PARTIE reçue");
        _etatBOM = S_INIT;
        afficheur.setProgression(g_dsCouleur, g_luminosite, 0,
g_batterie_faible);
        cc.setNbChocs(0); // RAZ du nombre de chocs
        break;

    default:
        Serial.println("Ordre inconnu.");

```

```

        break;
    } // sw
}

// =====
// GESTION DES CHOCS
// =====
if (_etatBOM != S_INIT) {
    if (cc.isChocs()) {
        Serial.print("Chocs = ");
        Serial.println(cc.getNbChocs());
        sendMessageToServer(E_CHOC);
    } // if cc
}

// =====
// RÉCEPTION DU CARACTERE VIA INFRAROUGE (PAV)
// =====
int coulPAV;
int typePAV;
if (Serial.available() > 0) {
    char car = Serial.read();
    Serial.print("Caractère reçu : ");
    Serial.println(car, HEX);
    if ( (_etatBOM!=S_INIT) && (_etatBOM!=S_FIN_TRANSFERT) ) {
        // décodage
        if (car != 0) { // parfois lecture de parasite valeur 0
            coulPAV = (car&COULEUR)>>1;
            typePAV = car&TYPE;
            if ( (coulPAV==g_dsCouleur) && (typePAV==g_type) ) { // Si compatible
                if (_etatBOM == S_JEUENCOURS) {
                    g_dep_time = millis();
                    _etatBOM = S_TRANSFERT;
                    g_adrIpPav = (car & ADRIP) >> 3;
                    sendMessageToServer(E_DEB_TRANSFERT);
                    Serial.println("Debut de transfert...");
                } // _etatBOM
                g_dep_dureeEntreDeuxBytes = millis();
            } // coul
        } // if car
    } // if _etatBOM
} // if available

// =====
// GESTION DU TRANSFERT (5 SECONDES)
// =====
if (_etatBOM == S_TRANSFERT) {
    g_deltaT = millis() - g_dep_time;
    if (g_deltaT > DUREE_TRANSFERT) {

```

```

        g_message = "\n-----\nTemps atteint fin
transfert\n-----\n";
        sendMessageToServer(E_DIVERS);
        _etatBOM = S_FIN_TRANSFERT;
////////////////////////////////////
////////////////////////////////////
        g_dep_pausePourReprise = millis();
        afficheur.setProgression(g_dsCouleur, g_luminosite,
afficheur.progression()+NBLEDS_PAR_TRANSFERT, g_batterie_faible);
        sendMessageToServer(E_FIN_TRANSFERT);
    } // if DUREEE_TRANSFERT

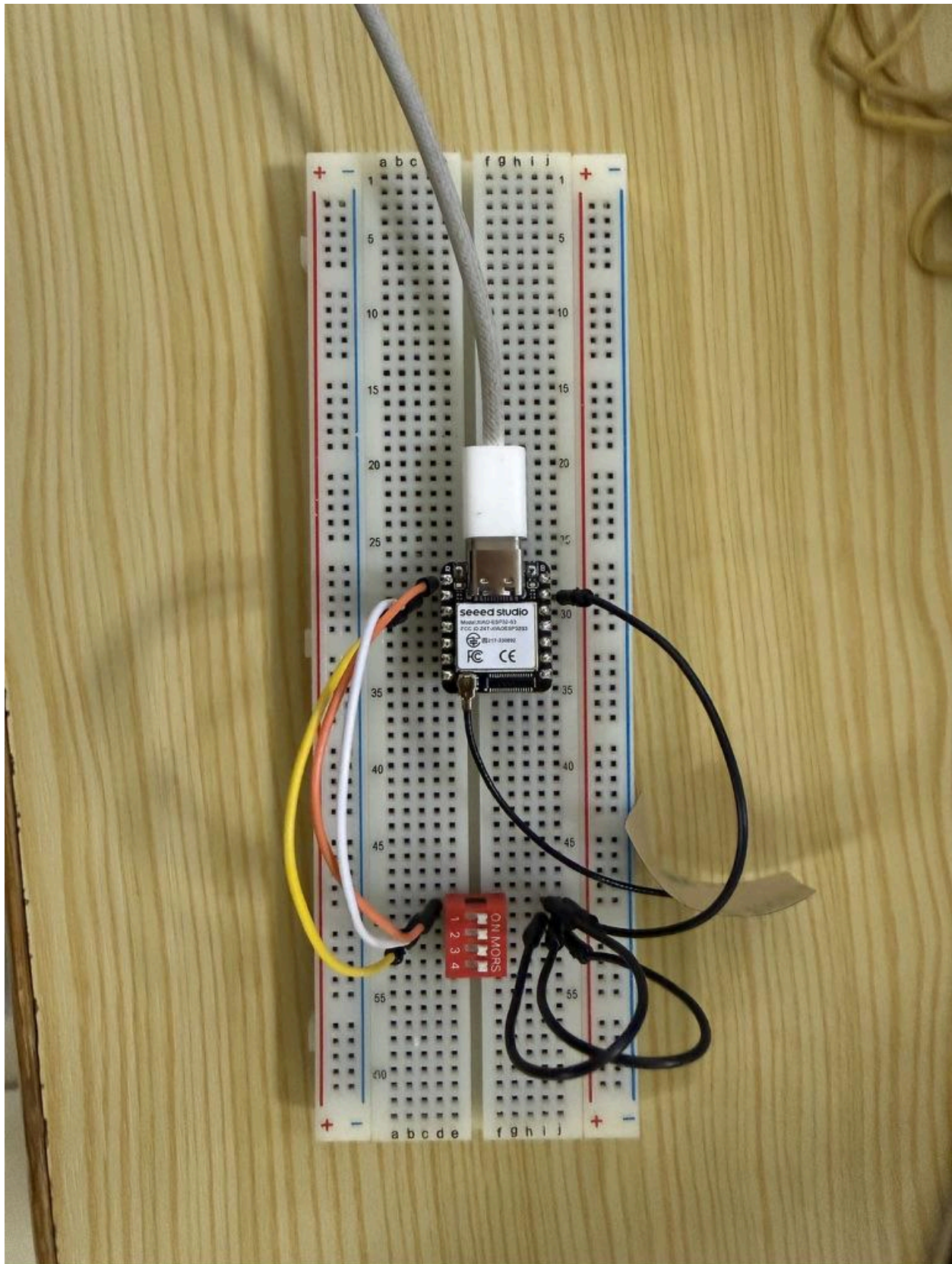
    g_deltaT = millis() - g_dep_dureeEntreDeuxBytes;
    Serial.printf("Début entre 2 octets : %d\n", g_dep_dureeEntreDeuxBytes);
    if (g_deltaT > DUREE_ENTRE_2BYTES) {
        _etatBOM = S_FIN_TRANSFERT;
        g_message = "Temps dépassé entre deux-----";
        sendMessageToServer(E_DIVERS);
        Serial.printf("deltaT = %d\n", g_deltaT);
        g_dep_pausePourReprise = millis();
        sendMessageToServer(E_ANNULATION_TRANSFERT);
    } // if DUREE_ENTRE_2BYTES
} // if _etatBOM

if (_etatBOM == S_FIN_TRANSFERT) {
    if ( (millis()-g_dep_pausePourReprise) > 1000) {
        g_message = "jeuencours\n";
        sendMessageToServer(E_DIVERS);
        _etatBOM = S_JEUENCOURS;
        Serial.println("JEU EN COURS");
    } // if millis
} // if fin_transfert

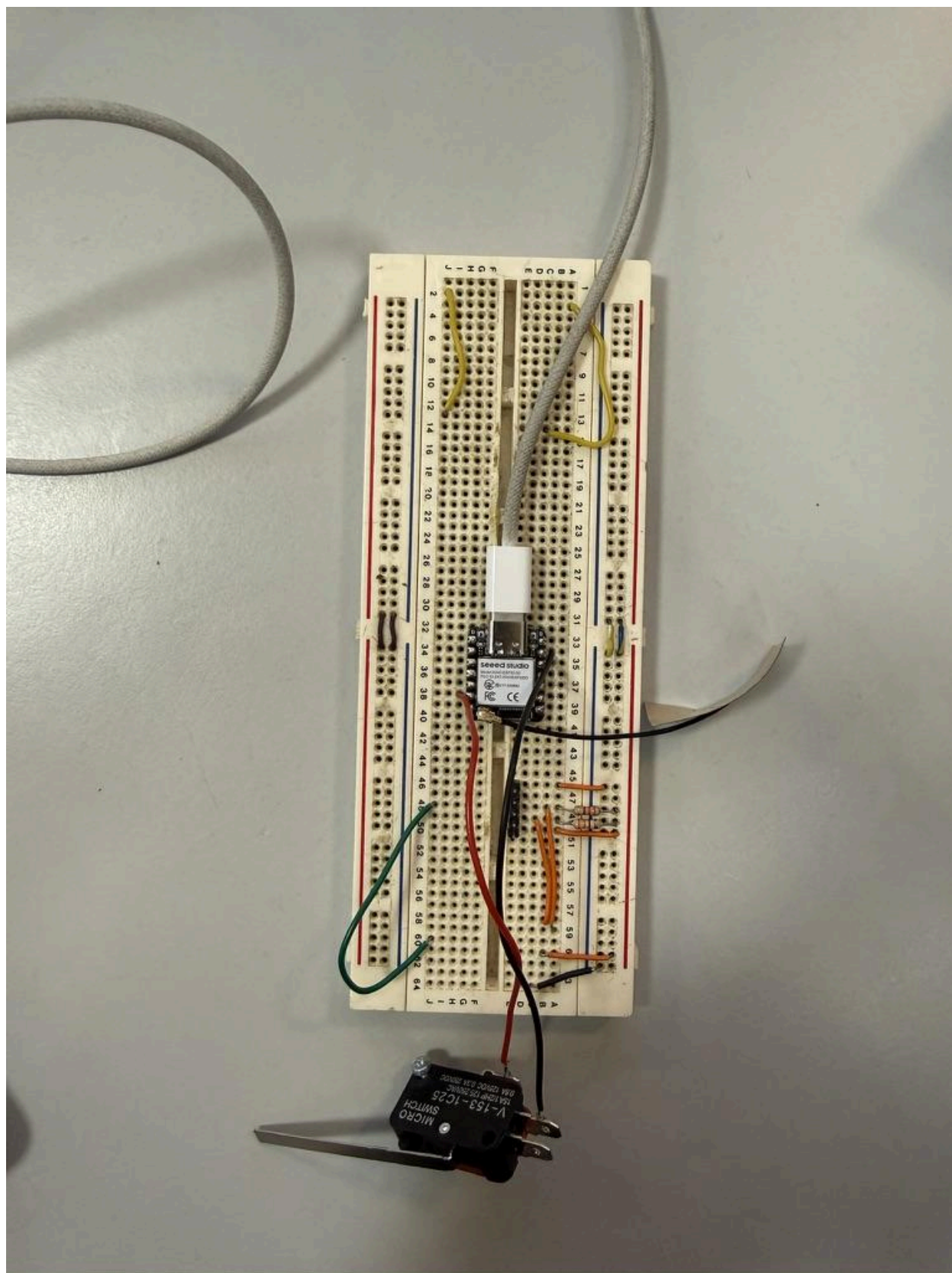
// =====
// AFFICHAGE LED (NeoPixel)
// =====
int nbLeds;
switch((int)_etatBOM) {
    case S_INIT:
        afficheur.off();
        break;
    case S_JEUENCOURS:
        nbLeds = afficheur.progression();
        afficheur.setProgression(g_dsCouleur, g_luminosite, nbLeds,
g_batterie_faible);
        break;
    case S_TRANSFERT:
        afficheur.clignote(g_dsCouleur, g_luminosite, g_batterie_faible);
        break;

```

```
        default:
            break;
    } // sw
} // loop
```

Annexe 2: Photo du montage sur le test des DIP SWITCH

Annexe 3: Cablâge sur breadboard



ANNEXE IR2

```

////////////////////////////////////
//
// PROGRAMME PAV MàJ du 09/04/2026 by PhA
// v2.0 La lecture de la tension batterie n'est pas opérationnelle
// v2.1 Modif luminosité 0-2 devient 1-3 1 faible, 2 moyen, 3 fort
// v2.2 Version Béta (Sans la détection batterie faible)
// v2.3 GPIO6 désactivé, conversion affinée, g_etatPAV, VER
// v2.4 Corrections mineures
// v2.5 Echange etat VIDE et PLEIN et ajout gestion batterie
// v2.6 Changement password WIFI ESTOM2026
// v2.7 18/04/2026 transmission des états du PAV
// v2.8 29/04/2026 Ajustement batterie faible et distinction PAV/ABB
// v2.9 05/05/2026 Corr bug couleur dip sw et vidage PAV
// v3.0 13/05/2026 Corr bug méthode CNeoPixel::vider()
// 18/05/26 v3.1 Corr orange vers rouge pour batterie faible
//
////////////////////////////////////
#define VER "3.1"

#include <Arduino.h>
#include <WiFi.h>
#include <WiFiClient.h>
#include <ArduinoJson.h>

#include "definitions.h"
#include "cdipswitch.h"
#include "cneopixel.h"
#include "cbatterie.h"

////////////////////////////////////
// VARIABLES ET INSTANCIATION GLOBALES
////////////////////////////////////

//
// A mettre à jour selon le contexte d'utilisation
//
const char *g_tcpAddress = "192.168.0.1";
const char *g_ssid = "AP-ESTOM";
const char *g_password = "ESTOM2026";
const uint16_t g_tcpPort = 5005;

```

```

//
// FIN METTRE A JOUR
//

T_ETATSPAV g_etatPAV=VIDE; // par défaut au démarrage état du PAV
//
// instanciations
//
CBatterie batt;
CDipSwitch ds;
CNeoPixel afficheur(NUM_LEDS, LED_PIN, DELAYVAL);
WiFiClient clientTcp;
IPAddress adrIpLocale;
//
// etats et valeurs de l'application
//
byte g_lastOctet = 1; // par défaut, contient derniere octet de l'adr IP
// TODO prévoir si batterie faible alors g_luminosite revient à faible : 0
uint8_t g_luminosite=1; // par défaut faible=1 moyen=2 fort=3
uint8_t g_dsCouleur=1; // no des switch sw2 et 3, donc entre 0-3
int g_type=0; // g_type de device : PAV ou abri bus (sw1 = 1)
bool g_batterie_faible = false;

////////////////////////////////////
////////////////////////////////////
/* IR
 * Set library modifiers first to set output pin etc.
 */
#include "PinDefinitionsAndMore.h"
// #define IR_OUTPUT_IS_ACTIVE_LOW
#define IRSND_IR_FREQUENCY          38000
#define IRSND_PROTOCOL_NAMES        1 // Enable protocol number mapping to
protocol strings - requires some FLASH.
#include <irsndSelectMain15Protocols.h>
// or disable #include <irsndSelectMain15Protocols.h> and use only one
protocol to save programming space
// #define IRSND_SUPPORT_NEC_PROTOCOL 1
// #define IRSND_SUPPORT_NEC_PROTOCOL 1
/* IR
 * After setting the definitions we can include the code and compile it.
 */
#include <irsnd.hpp>

```

```

IRMP_DATA irsnd_data;
#ifdef SEND_SAMSUNG
union WordUnion{
    struct {
        uint8_t LowByte;
        uint8_t HighByte;
    } UByte;
    struct {
        int8_t LowByte;
        int8_t HighByte;
    } Byte;
    uint8_t UBytes[2];
    int8_t Bytes[2];
    uint16_t UWord;
    int16_t Word;
    uint8_t *BytePointer;
};
#endif

////////////////////////////////////
void connectToWiFi() {
    Serial.print("Connexion au WiFi");
    WiFi.begin(g_ssid, g_password);
    unsigned long startAttemptTime = millis();
    while ( (WiFi.status() != WL_CONNECTED) && (
(millis()-startAttemptTime)<10000) ) {
        delay(500);
        Serial.print(".");
    } //wh
    if (WiFi.status() == WL_CONNECTED) {
        Serial.println("\nConnecté au WiFi");
        Serial.print("Adresse IP locale : ");
        Serial.println(WiFi.localIP());
        adrIpLocale = WiFi.localIP();
        g_lastOctet = adrIpLocale[3]; // dernier octet
        Serial.print("\ndernier octet :");
        Serial.println(g_lastOctet);
    } else {
        Serial.println("\nÉchec de connexion WiFi. RESTART !");
        ESP.restart(); // redémarrage logiciel
    } // else
} // fonction

```

```

/////////////////////////////////////////////////////////////////
/
bool connectToServer() {
    Serial.println("Connexion au serveur TCP...");
    if (clientTcp.connect(g_tcpAddress, g_tcpPort)) {
        Serial.println("Connexion TCP réussie.");
        return true;
    } else {
        Serial.println("Échec de connexion TCP.");
        return false;
    } // else
} // fonction
/////////////////////////////////////////////////////////////////
/
void sendTCPMessageToServer(T_TYPETFRAME trame) {
    if (!clientTcp.connected()) {
        Serial.println("Connexion perdue pour envoi, tentative de
reconnexion...");
        if (!clientTcp.connect(g_tcpAddress, g_tcpPort)) {
            Serial.println("Échec de reconnexion pour envoi.");
            return;
        } // if
    } // if
    DynamicJsonDocument doc(256);
    switch(trame) {
        case BONJOUR:
            doc["status"] = "0";
            doc["type"] = (g_type==0?"PAV":"ABB");
            doc["couleur"] = String(g_dsCouleur);
            break;
        case TT_VIDE:
            doc["status"] = "20";
            break;
        case TT_PLEIN:
            doc["status"] = "21";
            break;
        case TT_VIDAGE:
            doc["status"] = "22";
            break;
        default: // A effacer dans le futur pour les tests seulement
            doc["erreur"] = "Type de trame inconnue";
            break;
    }
}

```

```

    } // sw
    serializeJson(doc, clientTcp); // envoi json vers le serveur
    clientTcp.println(); // envoi saut de ligne vers le serveur
    Serial.print("Infos envoyées : ");
    serializeJson(doc, Serial);
    Serial.println();
} // fonction

////////////////////////////////////
////////
////////////////////////////////////
////////

void setup() {
    Serial.begin(115200); // init terminal de communication
    Serial.print("estomPAV v");
    Serial.println(VER);

    // test niveau batterie
    float nivBatt = batt.getValue();
    g_batterie_faible = (nivBatt<=SEUIL_BATTERIE_FAIBLE?true:false);
    Serial.print("Niveau batterie=");
    Serial.printf("%f\n",nivBatt);

    // lecture des sw au démarrage seulement
    ds.setup(); // DIP SW
    g_dsCouleur = ds.getDsCouleur();
    g_type = ds.getType(); // 1 abri bus 0 PAV avec couleur issue de SW1 et
SW2
    Serial.print("Couleur=");
    Serial.println(g_dsCouleur);
    Serial.print("Type=");
    Serial.println(g_type);

    // Just to know which program is running on my Arduino
    Serial.println(F("START " __FILE__ " from " __DATE__ "\r\nUsing library
version " VERSION_IRMP));
    irsnd_init();
    irmp_irsnd_LEDFeedback(true); // Enable send signal feedback at LED_BUILTIN
    Serial.println(F("Send IR signals at pin " STR(IRSND_OUTPUT_PIN)));

#ifdef SEND_SAMSUNG
    /*

```

```

    * Send Samsung32
    */
    irsnd_data.protocol = IRMP_SAMSUNG32_PROTOCOL;
    irsnd_data.address = 0x0707;
    irsnd_data.command = 0xFB04; // For my Samsung, the high byte is the
inverse of the low byte
    irsnd_data.flags = 0; // repeat frame 0 time
#else
    /*
    * Send NEC
    */
    irsnd_data.protocol = IRMP_NEC_PROTOCOL;
    irsnd_data.address = 0x25DA; // Adresse avec octet et octet complémenté
    irsnd_data.command = (g_lastOctet<<3)|(g_dsCouleur<<1)|g_type; // The
required inverse of the 8 bit command is added by the send routine.
    irsnd_data.flags = 1; // repeat frame
#endif

    // true = wait for frame and trailing space/gap to end. This stores timer
state and restores it after sending.
    if (!irsnd_send_data(&irsnd_data, true)) {
        Serial.println(F("Protocol not found")); // name of protocol is printed
by irsnd_data_print()
    }
    irsnd_data_print(&Serial, &irsnd_data);

    // init couleur et afficheur LED
    afficheur.begin(); // require to intialize object
    for(int i=0 ; i<5 ; i++) {
        afficheur.on(g_dsCouleur, g_luminosite, g_batterie_faible);
        delay(250);
        afficheur.off();
        delay(250);
    } // for

    // témoin d'émission IR
    pinMode(21,OUTPUT); // Pilotage USER_LED jaune
    digitalWrite(21, HIGH);

    // connexion WIFI
    connectToWiFi(); // reset si problème connexion
    // connexion au serveur TCP

```

```

while (!connectToServer()) {
    afficheur.clignote(g_dsCouleur, g_luminosite, g_batterie_faible);
    if (WiFi.status() != WL_CONNECTED) {
        Serial.println("\nÉchec de connexion WiFi. RESTART !");
        ESP.restart();
    } // if
    delay(2000);
} // wh
afficheur.off();

sendTCPMessageToServer(BONJOUR); // Envoi initial après connexion
delay(500); // attente réponse du serveur
} // setup

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////

void loop() {
    // test Connexion WIFI et au serveur ESTOM sinon restart
    if (!clientTcp.connected()) {
        Serial.println("\nÉchec de connexion au serveur. RESTART !");
        ESP.restart();
    } // if

    // test si trame reçue et lecture trame TCP from serveur
    if (clientTcp.available()) {
        String trameRec = clientTcp.readStringUntil('\n');
        Serial.println(trameRec);
        StaticJsonDocument<512> data;
        DeserializationError error = deserializeJson(data, trameRec);
        if (error) {
            Serial.print("Erreur JSON : ");
            Serial.println(error.f_str());
            return;
        } // if
        if (!data.containsKey("ordre")) {
            Serial.println("Trame invalide (pas de clé 'Ordre').");
            return;
        } // if

        int ordre = data["ordre"].as<String>().toInt();

```

```

int etatJeu;
int etatPAV;
Serial.print("Ordre reçu : ");
Serial.println(ordre);
switch (ordre) {
  case 0: // trame init
    etatPAV = (T_ETATSPAV)data["etatPAV"].as<String>().toInt();
    etatJeu = data["etatJeu"].as<String>().toInt();
    g_etatPAV = VIDE; // par défaut
    if ( (etatJeu==ENCOURS) && (etatPAV==PLEIN) ) g_etatPAV = PLEIN;
    if ( (etatJeu==ENCOURS) && (etatPAV==VIDAGE) ) g_etatPAV = VIDAGE;
    sendTCPMessageToServer((T_TYPETRAVE)g_etatPAV);
    break;
  case 1: // début partie
    g_luminosite = data["luminosite"].as<String>().toInt();
  case 13: // trame annulation transfert
    g_etatPAV = PLEIN;
    sendTCPMessageToServer(TT_PLEIN);
    break;
  case 11: // trame départ transfert
    g_etatPAV = VIDAGE;
    sendTCPMessageToServer(TT_VIDAGE);
    break;
  case 2: // trame fin partie
  case 12: // trame fin transfert
    g_etatPAV = VIDE;
    sendTCPMessageToServer(TT_VIDE);
    break;
  default:
    Serial.println("Ordre inconnu."); break;
} // sw
Serial.print("EtatPAV=");
Serial.println(g_etatPAV);
} // if available

// test niveau batterie
float nivBatt = batt.getValue();
g_batterie_faible = (nivBatt<=SEUIL_BATTERIE_FAIBLE?true:false);
//Serial.print("Niveau batterie=");
//Serial.printf("%f\n",nivBatt);

if (g_batterie_faible == true)

```

```
    g_luminosite = 1; // minimum

// contrôle de l'afficheur LED
switch(g_etatPAV) {
    case VIDE:
        afficheur.off();
        break;
    case PLEIN:
        afficheur.on(g_dsCouleur, g_luminosite, g_batterie_faible);
        break;
    case VIDAGE:
        afficheur.vider(g_dsCouleur, g_luminosite, g_batterie_faible);
        break;
    default: Serial.println("état inconnu."); break;
} // sw

// contrôle de l'IR avec émission data
switch(g_etatPAV) {
    case PLEIN:
    case VIDAGE:
        digitalWrite(21, LOW);
        // Utilisation de la bibliothèque IRMP
        irsnd_data.command=(g_lastOctet<<3) | (g_dsCouleur<<1) | g_type;
        irsnd_send_data(&irsnd_data, true);
        //irsnd_data_print(&Serial, &irsnd_data);
        delay(80);
        digitalWrite(21, HIGH);
        break;
} // sw
} // fonction
```

ANNEXE IR3

Annexe 1: Serveur TCP - CDatabase.cpp

```
#include "CDatabase.h"

CDatabase::CDatabase(QObject *parent) : QThread(parent)
{
    // Connexion temporaire pour reset
    QSqlDatabase resetDb = QSqlDatabase::addDatabase("QMYSQL",
"ResetConnection");
    resetDb.setHostName(MARIADB_HOST);
    resetDb.setPort(MARIADB_PORT);
    resetDb.setDatabaseName(MARIADB_DBNAME);
    resetDb.setUserName(MARIADB_NAME);
    resetDb.setPassword(MARIADB_PASSWORD);

    if (!resetDb.open()) {
        qDebug() << "Erreur : Impossible d'ouvrir la BDD pour le reset.";
        qDebug() << resetDb.lastError().text();
    } else {
        qDebug() << "Connexion temporaire pour reset réussie.";
        resetTables(resetDb);
    } // else

    // Supprimer la connexion temporaire pour éviter les conflits
    resetDb.close();
    resetDb = QSqlDatabase();
    QSqlDatabase::removeDatabase("ResetConnection"); // !!!!!!!

    // La vraie connexion du thread sera ouverte dans run()
}

CDatabase::~CDatabase()
{
    requestInterruption();
    wait();
}

void CDatabase::resetTables(QSqlDatabase &db)
{
    QSqlQuery query(db);

    QStringList tables = {"BOM", "PAV"};
    for (const QString &table : tables) {
        qDebug() << "Réinitialisation de la table :" << table;

        if (!query.exec("DELETE FROM " + table)) {
```

```

        qDebug() << "Erreur lors de la suppression des données de" <<
table << ":" << query.lastError().text();
    } // if delete
    if (!query.exec("ALTER TABLE " + table + " AUTO_INCREMENT = 1")) {
        qDebug() << "Erreur lors de la réinitialisation de l'ID de" <<
table << ":" << query.lastError().text();
    } // if alter
    } // for
    if (!query.exec("UPDATE Config SET Status = 0")) {
        qDebug() << "Erreur lors du changement de status de config :" <<
query.lastError().text();
    } // if delete
    query.finish();
    qDebug() << "Tables BOM et PAV réinitialisées avec succès.";
} // method

void CDatabase::run()
{
    threadDb = QSqlDatabase::addDatabase("QMYSQL", "CDatabaseConnection");
    threadDb.setHostName(MARIADB_HOST);
    threadDb.setPort(MARIADB_PORT);
    threadDb.setDatabaseName(MARIADB_DBNAME);
    threadDb.setUserName(MARIADB_NAME);
    threadDb.setPassword(MARIADB_PASSWORD);
    if (!threadDb.open()) {
        qDebug() << "Erreur : Impossible d'ouvrir la base de données depuis le
thread.";
        qDebug() << threadDb.lastError().text();
        return;
    } // if open

    T_CONFIG cfg;
    while (!isInterruptionRequested()) {
        QSqlQuery query(threadDb);
        if (!query.exec("SELECT * FROM Config")) {
            qDebug() << "Erreur SELECT Config :" << query.lastError().text();
            QThread::sleep(1);
            continue;
        } // if select
        if (!query.next()) {
            qDebug() << "La table Config est vide ou aucune ligne trouvée.";
            QThread::sleep(1);
            continue;
        } // if next
        cfg.options = query.value("Options").toString();
        cfg.ptsRecolte = query.value("PtsRecolte").toString();
        cfg.nbrPAV = query.value("NbrPAV").toString();
        cfg.status = query.value("Status").toString();
        cfg.luminosite = query.value("Luminosite").toString();
    }
}

```

```

        // synchro WEB --> Serveur TCP
        emit sig_configUpdated(cfg);
        QThread::sleep(1); // Pause entre chaque lecture
    } // wh
    // en cas de sortie du thread
    threadDb.close();
    QSqlDatabase::removeDatabase("CDatabaseConnection");
}

bool CDatabase::insertDB(const QString &table, const QVariantList valuesBP)
{
    if (valuesBP.isEmpty()) {
        qDebug() << "Erreur : aucune valeur fournie pour l'insertion.";
        return false;
    } // if empty

    int etat;
    etat = valuesBP.at(1).toString().toInt();
    QVariantList values = valuesBP;
    if ( (table=="PAV") && (etat==0) )
        values.removeAt(1); // enleve l'état pour PAV trame BONJOUR

    // Construire la liste des placeholders "?" pour la requête préparée
    QStringList placeholders;
    for (int i = 0; i < values.size(); ++i) {
        placeholders.append("?");
    } // for

    QStringList columns;
    QStringList arg;
    columns.clear();

    if (table == "BOM") {
        switch(etat) {
            case 0:
                arg = {"IPAddr", "Status", "Couleur"};
                break; // BONJOUR
            case 1:
                arg = {"IPAddr", "Status"};
                break; // VIDAGE
            case 2: // FIN VIDAGE
            case 3: // ANN VIDAGE
                arg = {"IPAddr", "Status", "Remplissage"};
                break;
            case 4:
                arg = {"IPAddr", "Status", "NbrCollision"};
                break; // CHOC
        } // sw
    } else { // "PAV"

```

```

        switch(etat) {
        case 0: // BONJOUR
            //values.removeAt(1); // enlève le status
            arg = {"IPAddr", /*"Status", */"Couleur"};
        break;
        case 20: // VIDE
        case 21: // PLEIN
        case 22: // VIDAGE
            arg = {"IPAddr", "Status"};
        break;
        } // sw
    } // else
    columns.append(arg);

    if (columns.size() != values.size()) {
        qDebug() << "Erreur : Le nombre de valeurs ne correspond pas aux
colonnes. Colom : " << columns.size() << "valeur:" << values.size();
        return false;
    } // if size

    // Construire la requête "ON DUPLICATE KEY UPDATE"
    //permet de mettre à jour une ligne existante lorsqu'une contrainte
d'unicité est violée lors d'un INSERT
    QStringList updateClauses;
    for (const QString &col : columns) {
        if (col != "IPAddr") { // Ne pas mettre à jour l'IPAddr car c'est la
clé
            updateClauses.append(QString("%1 = VALUES(%1)").arg(col));
            qDebug() << col;
        } // if
    } // for col

    QString queryString = QString(
        "INSERT INTO %1 (%2) VALUES (%3) "
        "ON DUPLICATE KEY UPDATE %4"
    ).arg(table)
        .arg(columns.join(", "))
        .arg(placeholders.join(", "))
        .arg(updateClauses.join(", "));

    QString ip;
    ip = values[0].toString();

    qDebug() << "Commande suite ip :" << ip << " " << queryString;

    QSqlQuery query(threadDb);
    query.prepare(queryString);
    for (const QVariant &value : values) {
        query.addBindValue(value);
    }

```

```

        qDebug() << value;
    } // for
    if (!query.exec()) {
        qDebug() << "Erreur lors de l'insertion :" <<
query.lastError().text();
        return false;
    } // if
    return true;
}

bool CDatabase::checkPAV(const QString &ip)
{
    QSqlQuery query(threadDb);

    if (!query.exec("SELECT IPAddr FROM PAV WHERE IPAddr = '"+ip+"'")) {
        qDebug() << "Erreur checkPAV : " << query.lastError().text();
        return false;
    } // if exec
    if (query.next()) {
        return true;
    } // if next
    return false;
}

QStringList CDatabase::getAllIPs()
{
    QStringList ipList;
    QSqlQuery query(threadDb);

    if (!query.exec("SELECT IPAddr FROM BOM UNION SELECT IPAddr FROM PAV")) {
        qDebug() << "Erreur lors de la récupération des IPs:" <<
query.lastError().text();
        return ipList;
    } // if exec

    while (query.next()) {
        ipList.append(query.value(0).toString());
    } // wh

    return ipList;
}

T_SEND CDatabase::getDatasToSend(QString ip, int ordre, T_SEND toSend)
{
    T_SEND toSendLocal;
    toSendLocal.pb = toSend.pb;
    QSqlQuery query(threadDb);
    QString req;

```

```

// récupération de la luminosité dans tous les cas
req = "SELECT Luminosite, Status FROM Config LIMIT 1";
if (!query.exec(req)) {
    qDebug() << "Erreur lors de la récupération des IPs:" <<
query.lastError().text();
    return toSendLocal;
} // if exec
query.next();
toSendLocal.luminosite = query.value(0).toString();
toSendLocal.etatJ = query.value(1).toString();

// répondre à BONJOUR du PAV, ordre 0
if ( (toSendLocal.pb=="P") && (ordre==0) ) {
    req = "SELECT Status FROM PAV WHERE IPAddr='"+ip+"'";
    if (!query.exec(req)) {
        qDebug() << "Erreur lors de la récupération des IPs:" <<
query.lastError().text();
        return toSendLocal;
    } // if exec
    query.next();
    toSendLocal.etatP = query.value(0).toString();
} // if répondre INIT

// répondre à BONJOUR du BOM, ordre 0
if (toSendLocal.pb == "B" && ordre == 0) {
    req = "SELECT NbrCollision, Remplissage FROM BOM WHERE IPAddr='"+ip+"'";
    if (!query.exec(req)) {
        qDebug() << "Erreur lors de la récupération des IPs:" <<
query.lastError().text();
        return toSendLocal;
    } // if exec
    query.next();
    toSendLocal.collisions = query.value(0).toString();
    toSendLocal.leds = query.value(1).toString();
} // if P 0
return toSendLocal;
}

```

Annexe 2: Serveur TCP - CConnectChecker.cpp

```

#include "CConnectChecker.h"

CConnectChecker::CConnectChecker(QObject *parent)
    : QThread(parent), shouldStop(false) {}

CConnectChecker::~CConnectChecker() {
    stop();
    wait();
}

```

```

void CConnectChecker::stop() {
    shouldStop = true;
}

QSqlDatabase CConnectChecker::getDatabaseConnection()
{
    QString connectionName =
QSqlDatabase("checkerConnection_1").arg(reinterpret_cast<quintptr>(QThread::currentThread()));

    if (QSqlDatabase::contains(connectionName)) {
        return QSqlDatabase::database(connectionName);
    } // if contains

    QSqlDatabase db = QSqlDatabase::addDatabase("QMYSQL", connectionName);
    db.setHostName(MARIADB_HOST);
    db.setPort(MARIADB_PORT);
    db.setDatabaseName(MARIADB_DBNAME);
    db.setUserName(MARIADB_NAME);
    db.setPassword(MARIADB_PASSWORD);
    if (!db.open()) {
        qDebug() << "Erreur : Impossible d'ouvrir la base de données.";
        qDebug() << db.lastError().text();
    } // if open
    return db;
}

void CConnectChecker::run()
{
    while (!shouldStop) {
        QStringList ipList = getAllIPs(); // Met à jour la liste à chaque
cycle
        for (const QString &ip : ipList) {
            if (shouldStop) break; // Vérifie l'arrêt en cours d'exécution
            bool isConnected = pingIP(ip);
            updateConnectStatus(ip, isConnected ? 1 : 0);
        } // for ipList
        for (int i = 0; i < 50 && !shouldStop; ++i) {
            QThread::msleep(100); // Pause plus réactive (5s au total)
        } // for
    } // wh
    // Nettoyage de la connexion après utilisation
    QSqlDatabase::removeDatabase(QSqlDatabase::database().connectionName());
}

bool CConnectChecker::pingIP(const QString &ip) {
    QProcess process;
    process.start("ping", QStringList() << "-c" << "1" << "-W" << "1" << ip);
}

```

```

        process.waitForFinished(1000); // Timeout général de 2s
        return process.exitCode() == 0; // Retourne true si l'IP répond au ping
    }

int CConnectChecker::getConnectStatus(const QString &ip)
{
    QSqlDatabase db = getDatabaseConnection();
    QSqlQuery query(db);
    query.prepare("SELECT Connected FROM BOM WHERE IPAddr = :ip UNION SELECT
Connected FROM PAV WHERE IPAddr = :ip");
    query.bindValue(":ip", ip);
    if (!query.exec()) {
        qDebug() << "Erreur lors de la récupération du statut de connexion
pour" << ip << ":" << query.lastError().text();
        return -1;
    } // if exec
    if (query.next()) {
        return query.value(0).toInt();
    } // if next

    return -1;
}

QStringList CConnectChecker::getAllIPs()
{
    QStringList ipList;
    QSqlDatabase db = getDatabaseConnection();
    QSqlQuery query(db);
    if (!query.exec("SELECT IPAddr FROM BOM UNION SELECT IPAddr FROM PAV")) {
        qDebug() << "Erreur lors de la récupération des IPs:" <<
query.lastError().text();
        return ipList;
    } // if exec
    while (query.next()) {
        ipList.append(query.value(0).toString());
    } // wh next
    //qDebug() << "Liste des IPs : " << ipList;
    return ipList;
}

void CConnectChecker::updateConnectStatus(const QString &ip, int status)
{
    QSqlDatabase db = getDatabaseConnection();
    QSqlQuery query(db);

    query.prepare("UPDATE BOM SET Connected = :status WHERE IPAddr = :ip");
    query.bindValue(":status", status);
    query.bindValue(":ip", ip);
    if (!query.exec()) {

```

```

        qDebug() << "Erreur lors de la mise à jour de BOM:" <<
query.lastError().text();
    } // if exec
    query.prepare("UPDATE PAV SET Connected = :status WHERE IPAddr = :ip");
    query.bindValue(":status", status);
    query.bindValue(":ip", ip);
    if (!query.exec()) {
        qDebug() << "Erreur lors de la mise à jour de PAV:" <<
query.lastError().text();
    } // if exec
}

```

Annexe 3: Serveur TCP - CTcpSender.cpp

```

// CTcpSender.cpp
#include "CTcpSender.h"

CTcpSender::CTcpSender(CTcpServer *tcpServer, QObject *parent) :
QObject(parent), _tcpServer(tcpServer) {
    // nada !
}

void CTcpSender::on_sendTcpMessage(const QString &ip, int ordre, T_SEND
toSend)
{
    qDebug() << "<====|TCP-SENDER-" << ip <<"|====>";
    QTcpSocket *socket = _tcpServer->getSocketForIP(ip);

    if (!socket || socket->state() != QAbstractSocket::ConnectedState) {
        qWarning() << "Socket non disponible ou déconnecté pour l'IP:" << ip;
        return;
    } // if socket

    QJsonObject jsonObj;
    jsonObj["ordre"] = QString::number(ordre);
    switch(ordre) {
    case 0: // INIT suite à BONJOUR
        qDebug() << "Ordre0: INIT;";
        if (toSend.pb == "B") {
            jsonObj["etatJeu"] = toSend.etatJ;
            jsonObj["collisions"] = toSend.collisions;
            jsonObj["leds"] = toSend.leds;
        } else {
            jsonObj["etatPAV"] = toSend.etatP;
            jsonObj["etatJeu"] = toSend.etatJ;
        } // else
        break;
    case 1:
        qDebug() << "Ordre1: START;";

```

```

        jsonObj["luminosite"] = toSend.luminosite;
        break;
    case 2: qDebug() << "Ordre2: STOP;"; break;
    case 11: qDebug() << "Ordre11: EMPTYING;"; break;
    case 12: qDebug() << "Ordre12: EMPTY;"; break;
    case 13: qDebug() << "Ordre13: CANCEL;"; break;
} // sw

QJsonDocument doc(jsonObj);
QByteArray message = doc.toJson(QJsonDocument::Compact) + '\n';

socket->write(message);
socket->flush();
qDebug() << "Message TCP envoyé à" << ip;
qDebug() << "Données:" << message;
}

```

Annexe 4: Serveur TCP - CTcpServer.cpp

```

// CTcpServer.cpp
#include "CTcpServer.h"

CTcpServer::CTcpServer(QObject *parent) : QTcpServer(parent) {}

void CTcpServer::startServer(quint16 port) {
    if (!this->listen(QHostAddress::Any, port)) {
        qDebug() << "Server TCP n'a pas démarré!";
    } else {
        qDebug() << "Server TCP démarré sur le port" << port;
    } // else
}

QTcpSocket *CTcpServer::getSocketForIP(const QString &ip) const
{
    return socketMap.value(ip, nullptr);
}

void CTcpServer::incomingConnection(qintptr socketDescriptor) {
    QMetaObject::invokeMethod(this, [=]() {
        QTcpSocket *clientSocket = new QTcpSocket(); // Pas de parent ici !

        if (!clientSocket->setSocketDescriptor(socketDescriptor)) {
            qDebug() << "Erreur de socket TCP !";
            clientSocket->deleteLater();
            return;
        } // if client

        // Adresse IP du client connecté
        QString senderIP = clientSocket->peerAddress().toString();
    });
}

```

```

    if (senderIP.startsWith("::ffff:")) {
        senderIP = senderIP.right(senderIP.length() - 7);
    } // if sender
    qDebug() << "Nouvelle connexion TCP de:" << senderIP;

    // Stockage temporaire uniquement si socket est en ConnectedState
    if (clientSocket->state() == QAbstractSocket::ConnectedState) {
        socketMap[senderIP] = clientSocket;
    } else {
        connect(clientSocket, &QTcpSocket::connected, this, [=]() {
            qDebug() << "Connexion confirmée pour:" << senderIP;
            socketMap[senderIP] = clientSocket;
        });
    } // else

    QByteArray *buffer = new QByteArray;
    // Message reçu d'un périphérique (PAV, BOM)
    connect(clientSocket, &QTcpSocket::readyRead, this, [=]() {
        buffer->append(clientSocket->readAll());
        if (!buffer->contains('\n')) return;
        QList<QByteArray> messages = buffer->split('\n');
        for (int i = 0; i < messages.size() - 1; ++i) {
            QByteArray message = messages[i].trimmed();
            if (message.isEmpty()) continue;
            qDebug() << "<====|TCP-REQUEST-" << senderIP << "|====>";
            qDebug() << "Message TCP reçu:" << message;
            QJsonParseError parseError;
            QJsonDocument doc = QJsonDocument::fromJson(message,
&parseError);
            if (parseError.error != QJsonParseError::NoError ||
!doc.isObject()) {
                qDebug() << "Erreur JSON TCP:" <<
parseError.errorString();
                continue;
            } // if
            QJsonObject jsonObj = doc.object();
            T_INFOS infos;
            infos.status = jsonObj.value("status").toString();

            switch(infos.status.toInt()) {
            case 0: // BOM/PAV/BUS/ABB annonce BONJOUR
                infos.type = jsonObj.value("type").toString();
                infos.couleur = jsonObj.value("couleur").toString();
                if (infos.type == "BUS") infos.couleur = "40";
                if (infos.type == "ABB") infos.couleur = "40";
                break;
            case 1: // BOM/BUS annonce DEB TRANSFER
                infos.type = "BOM";

```

```

        infos.ipPAV =
QString(PREFIXE_IP)+jsonObj.value("ipPAV").toString();
        break;
        case 2: // BOM/Bus annonce FIN TRANSFER
        case 3: // BOM/Bus annonce ANN TRANSFER
            infos.type = "BOM";
            infos.ipPAV =
QString(PREFIXE_IP)+jsonObj.value("ipPAV").toString();
            infos.leds = jsonObj.value("leds").toString();
            break;
        case 4: // BOM/Bus annonce CHOCS
            infos.type = "BOM";
            infos.collisions = jsonObj.value("collisions").toString();
            break;
        case 20: // PAV/ABB VIDE
            infos.type = "PAV";
            infos.ipPAV =
QString(PREFIXE_IP)+jsonObj.value("ipPAV").toString();
            break;
        case 21: // PAV/ABB PLEIN
            infos.type = "PAV";
            infos.ipPAV =
QString(PREFIXE_IP)+jsonObj.value("ipPAV").toString();
            break;
        case 22: // PAV/ABB VIDAGE
            infos.type = "PAV";
            infos.ipPAV =
QString(PREFIXE_IP)+jsonObj.value("ipPAV").toString();
            break;
        case 99: // message debug de BOM
            qDebug() << jsonObj.value("texte").toString();
            continue;
            break;
    } // sw

    // adresse IP du périphérique qui a envoyé la trame
    QString senderIP = clientSocket->peerAddress().toString();
    if (senderIP.startsWith("::ffff:")) {
        senderIP = senderIP.right(senderIP.length() - 7);
    } // if senderIp

    // Vers la BDD via CApp
    emit sig_infoUpdated(infos, senderIP);
} // for

*buffer = messages.last();
});
connect(clientSocket, &QTcpSocket::disconnected, clientSocket,
&QTcpSocket::deleteLater);

```

```
    }, Qt::QueuedConnection);
}
```

Annexe 5: Serveur TCP - CApp.cpp

```
#include "CApp.h"

CApp::CApp(QObject *parent)
    : QObject(parent)
{
    init();
}

CApp::~CApp()
{
    if (_dbReader) {
        _dbReader->requestInterruption();
        _dbReader->wait();
        delete _dbReader;
    }
    if (_tcpThread) {
        _tcpThread->quit();
        _tcpThread->wait();
        delete _tcpThread;
    }
    delete _tcpSender;
    if (_connectThread) {
        _connectThread->quit();
        _connectThread->wait();
        delete _connectThread;
    }
    delete _connectChecker;
}

void CApp::on_configUpdated(T_CONFIG cfg)
{
    static int _etatPrecedent = -1; //-1 pour garantir un premier envoi
    _currentEtat = cfg.status.toInt();
    QStringList ipList = _dbReader->getAllIPs();
    T_SEND toSend;

    if (_currentEtat != _etatPrecedent) {
        toSend.etatP = QString::number(_currentEtat);
        for (const QString &ip : ipList) {
            switch (_currentEtat) {
                case 0: // attente départ partie
                    if (_dbReader->checkPAV(ip)==true)
                        toSend.pb = "P";
                    else
```

```

        toSend.pb = "B";
        sendMsgTCP(ip, 0, toSend); // Attention, INIT différents selon
PAV et BOM
        qDebug() << "Ordre 0, initialisation....";
        break;
    case 1: // départ partie
        sendMsgTCP(ip, 1, toSend);
        qDebug() << "Ordre 1, début de partie....";
        break;
    case 2: // fin de la partie (podium)
        sendMsgTCP(ip, 2, toSend);
        qDebug() << "Ordre 2, fin de la partie....";
        break;
    case 3: // compte à rebour début partie
        // Pas d'effet sur PAB, BOM
        // sendMsgTCP(ip, 0, type);
        // qDebug() << "Ordre 0....?";
        break;
    } //sw
} // for
_etatPrecedent = _currentEtat;
} // if
} // method

// Appelé lors de la réception d'un état d'un périphérique
// réponse lorsque c'est nécessaire
void CApp::on_infoUpdated(T_INFOS infos, QString ip)
{
    qDebug() << "<====|MAJ-INFO-" << ip << "|====>";

    T_SEND toSend;
    if ( (infos.type=="BOM") || (infos.type=="BUS") ){
        toSend.pb = "B";
        switch (infos.status.toInt()) {
        case 0: //BONJOUR
            qDebug() << "BONJOUR de " << infos.type << " (" << ip << ")";
            _dbReader->insertDB("BOM", QVariantList{ip, infos.status,
infos.couleur});
            sendMsgTCP(ip, 0, toSend); // Message INIT
            break;
        case 1: //EMPTYING
            if (_currentEtat == 1) {
                _dbReader->insertDB("BOM", QVariantList{ip, infos.status});
                qDebug() << "Remplissage en cours de BOM (" << ip << ") qui
vide PAV (" << infos.ipPAV << ")";
                sendMsgTCP(infos.ipPAV, 11, toSend);
            } // if
            break;
        case 2: //EMPTY

```

```

        if (_currentEtat == 1) {
            _dbReader->insertDB("BOM", QVariantList{ip, infos.status,
infos.leds});
            sendMsgTCP(infos.ipPAV, 12, toSend);
        } // if 1
        break;
    case 3: //CANCEL
        if (_currentEtat == 1) {
            _dbReader->insertDB("BOM", QVariantList{ ip, infos.status,
infos.leds});
            sendMsgTCP(infos.ipPAV, 13, toSend);
        } // if 1
        break;
    case 4: //CHOC
        if (_currentEtat == 1) {
            _dbReader->insertDB("BOM", QVariantList{ip, infos.status,
infos.collisions});
        } // if 1
        break;
    default:
        qDebug() << "Status BOM inconnu :" << infos.status;
        break;
    } // sw
} // if BOM

if ( (infos.type=="PAV") || (infos.type=="ABB") ) {
    toSend.pb = "P";
    switch (infos.status.toInt()) {
    case 0: // BONJOUR
        qDebug() << "BONJOUR de PAV (" << ip << ")";
        _dbReader->insertDB("PAV", QVariantList{ip, infos.status,
infos.couleur});
        sendMsgTCP(ip, 0, toSend); // envoi de INIT
        break;
    case 20: // PAV VIDE
        _dbReader->insertDB("PAV", QVariantList{ip, infos.status});
        break;
    case 21: // PAV PLEIN
        _dbReader->insertDB("PAV", QVariantList{ip, infos.status});
        break;
    case 22: // PAV VIDAGE
        _dbReader->insertDB("PAV", QVariantList{ip, infos.status});
        break;
    default:
        qDebug() << "Status BOM inconnu :" << infos.status;
    } // sw
} // if PAV
}

```

```

void CApp::sendMsgTCP(const QString &ip, int ordre, T_SEND toSend)
{
    toSend = _dbReader->getDatasToSend(ip, ordre, toSend);
    emit sig_sendTcpMessage(ip, ordre, toSend);
}

void CApp::init() {
    // Initialisation de la base de données
    _dbReader = new CDatabase(this);

    // Démarrage du serveur TCP dans un thread
    _tcpServer = new CTcpServer;
    _tcpThread = new QThread();
    _tcpServer->moveToThread(_tcpThread);

    // Connexion du signal de démarrage
    connect(_tcpThread, &QThread::started, [=]() {
        _tcpServer->startServer(TCP_PORT);
    });

    connect(_tcpServer, &CTcpServer::sig_infoUpdated, this,
    &CApp::on_infoUpdated);
    // Nettoyage du thread quand il se termine
    connect(_tcpThread, &QThread::finished, _tcpServer,
    &QObject::deleteLater);
    _tcpThread->start();
    // Création du sender TCP
    _tcpSender = new CTcpSender(_tcpServer);
    _tcpSender->moveToThread(_tcpThread);
    // connecte
    connect(this, &CApp::sig_sendTcpMessage, _tcpSender,
    &CTcpSender::on_sendTcpMessage);
    // Connexion pour les mises à jour de configuration
    connect(_dbReader, &CDatabase::sig_configUpdated, this,
    &CApp::on_configUpdated);
    _dbReader->start();

    _connectChecker = new CConnectChecker(_dbReader);
    _connectChecker->start();
}

```

Annexe 6: Serveur Web - index.php

```

<?php
require 'debut.inc.php';
require 'head.inc.php';
require 'bodyHeader.inc.php';
?>

```

```

    <main>
        <div class="image-container">
            <div class="manetteImg">
                <a href="./gestion-de-la-partie.php">
                    
                </a>
            </div>
            <div class="engrenagesImg">
                <a href="./parametre-du-jeu.php">
                    
                </a>
            </div>
        </div>

    </main>
<?php
require 'fin.inc.php';
?>

```

Annexe 7: Serveur Web - gestion-de-la-partie.php

```

<?php
require 'debut.inc.php';
require 'head.inc.php';
require 'bodyHeader.inc.php';
?>

<main>
    <section>
        <h2>Gestion de la partie</h2>
        <div id="timer" class="timer">Vous pouvez lancer la partie !</div>

        <div id="TableauGestion" class="GTableauGestion">
            <!-- Bouton pour ouvrir la fenêtre de saisie des joueurs -->
            <button id="openModalButton">Nom des joueurs</button>

            <!-- Fenêtre modale Joueur -->
            <div id="playerModal" class="modal">
                <div class="modal-content">
                    <span class="close-btn">&times;</span>
                    <h2>Nom de vos joueurs</h2>
                    <form id="playerForm">
                        <div id="playerInputs">
                            <?php
                                require './private-estom/db_connect.php';

                                // Récupération du nombre de BOM configurées

```

```

        $sql = "SELECT NbrBOM_V, NbrBOM_J, NbrBOM_N,
NbrBOM_B FROM Config WHERE Id = 1";
        $result = $conn->query($sql);
        $nbrBOMConfig = 0;

        if ($result && $result->num_rows > 0) {
            $row = $result->fetch_assoc();
            $nbrBOMConfig = $row['NbrBOM_V'] +
$row['NbrBOM_J'] + $row['NbrBOM_N'] + $row['NbrBOM_B'];
        }

        // Récupération du nombre de BOM enregistrées
seulement, sans le bus

        $sql = "SELECT COUNT(*) AS totalBOM FROM BOM
WHERE Couleur<4";

        $result = $conn->query($sql);
        $nbrBOMEnregistrees = 0;

        if ($result && $result->num_rows > 0) {
            $row = $result->fetch_assoc();
            $nbrBOMEnregistrees = $row['totalBOM'];
        }
        $colorMap = [
            0=> "Jaune",
            1=> "Vert",
            2=> "Bleu",
            3=> "Noir",
            40=> "Blanc"
        ];

        // Récupération des BOM avec leur ID, couleur
et joueur

        $sql = "SELECT ID, Couleur, Joueur FROM BOM
ORDER BY Couleur, ID";

        $result = $conn->query($sql);

        // Conversion explicite en entiers
        $nbrBOMConfig2 = (int) $nbrBOMConfig;
        $nbrBOMEnregistrees2 = (int)
$nbrBOMEnregistrees;

        if ($result && $result->num_rows > 0) {
            while ($row = $result->fetch_assoc()) {
                $bomID = $row['ID'];
                $couleur = $colorMap[$row['Couleur']];
                $joueur = $row['Joueur'] ?? "";

                echo "<div class='player-input'>

```

```

<span>". ($couleur=="Blanc"? "BUS": "BOM") . "#$bomID - $couleur</span>
        <input type='text'
name='joueurs[$bomID]' value='$joueur' maxlength='20' placeholder='Pseudo
joueur' required>

        </div>";
    }
}
// Affichage du message d'alerte si le nombre
de BOM est incorrect

    if ($nbrBOMConfig2 != $nbrBOMEnregistrees2) {
        echo "<p style='color: red; font-weight:
bold; margin-top: 10px;'>
                ⚠ Attention : La configuration
prévoit <strong>$nbrBOMConfig2 BOM</strong>,
                mais <strong>$nbrBOMEnregistrees2
BOM</strong> sont enregistrées !
                </p>";
    }
    ?>
</div>
<button type="submit">Valider</button>
</form>
</div>
</div>

<?php
require './private-estom/db_connect.php';

// Récupérer la durée du timer (en secondes) depuis la base de
données

$timerDuration = 180; // Valeur par défaut : 3 minutes (180
secondes)

$sql = "SELECT duree FROM Config WHERE id = 1"; // Exemple de
table de config
$result = $conn->query($sql);

if ($result && $result->num_rows > 0) {
    $row = $result->fetch_assoc();
    $timerDuration = (int)$row['duree'];
}

// Injecter la durée du timer dans le script JS
echo "<script>const timerDuration = $timerDuration;</script>";

// Récupération des paramètres existants
$sql = "SELECT Options, PtsRecolte, NbrPAV, MalusCollision,
NbrBOM_V, NbrBOM_J, NbrBOM_N, NbrBOM_B, duree FROM Config WHERE Id = 1";
$result = $conn->query($sql);

```

```

if ($result && $result->num_rows > 0) {
    $row = $result->fetch_assoc();
    $options = $row['Options'];
    $ptsRecolte = $row['PtsRecolte'];
    $nbrPAV = $row['NbrPAV'];
    $malusCollision = $row['MalusCollision'];
    $nbrBOM_V = $row['NbrBOM_V'];
    $nbrBOM_J = $row['NbrBOM_J'];
    $nbrBOM_N = $row['NbrBOM_N'];
    $nbrBOM_B = $row['NbrBOM_B'];
    $tempsPartie = $row['duree'];
} // if
echo "<script>
        const malusCollision = $malusCollision;
        const ptsRecolte = $ptsRecolte;
        const optionMode = $options;
        const nbrPAV = $nbrPAV;
    </script>";

$colorMap = [
    "Jaune" => 0,
    "Vert" => 1,
    "Bleu" => 2,
    "Noir" => 3
    //"Blanc" => 40
];
?>

<button id="startGameButton">Lancer la partie</button>
<button id="resetGameButton"
style="display:none;">Réinitialiser la partie</button>
</div> <!-- fin div "TableauGestion" -->

<!-- Modal "podiumModal" -->
<div id="podiumModal" class="modal" style="display:none;">
    <div class="modal-content">
        <div class="img">
            
        </div>
        <div id="classementContent">
            <!-- Contenu du podium injecté ici par JavaScript -->
        </div>
    </div>
</div>

<div id="tableauBars">
    <?php

```

```

//      CRGB(255,205,0); // jaune RAL1018
//      CRGB(0,124,89); // vert RAL6032
//      CRGB(0,83,135); // Bleu RAL5015
//      CRGB(85,93,80); // Gris RAL7011

$couleurs = [
    "Jaune" => $nbrBOM_J,
    "Vert" => $nbrBOM_V,
    "Bleu" => $nbrBOM_B,
    "Noir" => $nbrBOM_N
];

$rgbaColors = [ //rgb correspondant aux couleurs BOM & PAV
/*      "Vert" => "rgba(14, 202, 26, 0.6)",
    "Jaune" => "rgba(250, 206, 0, 0.6)",
    "Noir" => "rgba(59, 59, 59, 0.6)",
    "Bleu" => "rgba(23, 140, 221, 0.6)" */
    "Jaune" => "rgba(255, 205, 0, 0.6)",
    "Vert" => "rgba(0, 124, 89, 0.6)",
    "Bleu" => "rgba(0, 83, 135, 0.6)",
    "Noir" => "rgba(85, 93, 80, 0.6)",
    "Blanc" => "rgba(255, 255, 255, 0.6)"
];

foreach ($couleurs as $couleur => $nombreBOM) {
    // Récupérer les IP des BOM de cette couleur
    $sql = "SELECT IPAddr FROM BOM WHERE Couleur = ?";
    $stmt = $conn->prepare($sql);
    $stmt->bind_param("i", $colorMap[$couleur]);
    $stmt->execute();
    $result = $stmt->get_result();

    $ipList = [];
    while ($row = $result->fetch_assoc()) {
        $ipList[] = $row['IPAddr'];
    }
    $stmt->close();

    for ($i = 0; $i < $nombreBOM; $i++) { // normalement 1
seul BOM !
        $ip = $ipList[$i] ?? "NULL"; // Assigne une IP si
disponible, sinon NULL

        echo "<div class='route-container'
data-couleur='$couleur' data-ip='$ip'>";

        // Scoreboard à gauche du parcours
        echo "<div class='scoreboard-container'>
<img src='./img/CadreScore-$couleur.png'
class='scoreboard' data-couleur='$couleur' data-ip='$ip'>
<span class='score-text' id='score-$ip'>0</span>";
    }
}

```

```

</div>";

echo "<img src='./img/route.png' alt='Route'
class='imageRoute'>";

// BOM avec IP ou sans IP
echo "<img src='./img/BOM-$couleur.png' alt='Camion
$couleur'

class='imageCamion' data-couleur='$couleur'

data-ip='$ip'

style='position: absolute; left: 0%; top: 50%;
width: 80px; opacity: 0.9; z-index: 3;'>";
echo "<span class='pseudo' data-ip='$ip'
style='
display:none;
background-color: {$rgbaColors[$couleur]};
'></span>";

echo "<img src='./img/LigneArrive.gif' alt='Ligne
d'arrivée' class='imageArrive'>";

//
// PAVs ABBs
//
for ($j = 0; $j < $nbrPAV; $j++) {
    $espacement = 95/$nbrPAV; // et non 100 pour
laisser un espace avant la fin
    $positionLeft = ($j + 1) * $espacement;

    echo "<img src='./img/PAV-$couleur.png' alt='PAV
$couleur'

class='imagePAV' data-couleur='$couleur'

data-ip='$ip' data-index='$j'

style='
position: absolute;
top: 50%;
left: {$positionLeft}%;
transform: translate(-50%, -50%);
width: 50px;
opacity: 1;
z-index: 4;

'>";
} // for j
echo "</div>";
} // for i
} // foreach couleur

//
// Affichage du BUS et ABB

```

```

//
$couleur = 'Blanc';
$sql = "SELECT IPAddr FROM BOM WHERE Couleur = 40 LIMIT 1";
// 1 BUS

$stmt = $conn->prepare($sql);
$stmt->execute();
$result = $stmt->get_result();

if ($result->num_rows === 1) { // Si un bus détecté
    $row = $result->fetch_assoc();
    $ip = $row['IPAddr'];
    $stmt->close();
    echo "<div class='route-container' data-couleur='Blanc'
data-ip='$ip'>";

    // Scoreboard à gauche du parcours
    echo "<div class='scoreboard-container'>
    <img src='../img/CadreScore-$couleur.png'
class='scoreboard' data-couleur='$couleur' data-ip='$ip'>
    <span class='score-text' id='score-$ip'>0</span>
    </div>";

    echo "<img src='../img/route.png' alt='Route'
class='imageRoute'>";

    // BUS avec IP ou sans IP
    echo "<img src='../img/bus.png' alt='Bus'
        class='imageCamion' data-couleur='Blanc' data-ip='$ip'
        style='position: absolute; left: 0%; top: 50%; width:
120px; opacity: 0.9; z-index: 3;'>";
    echo "<span class='pseudo' data-ip='$ip'
        style='
            display:none;
            background-color: {$rgbaColors['Blanc']};
        '></span>";

    echo "<img src='../img/LigneArrive.gif' alt='Ligne
d'arrivée' class='imageArrive'>";

    for ($j = 0; $j < $nbrPAV; $j++) {
        $espacement = 95/$nbrPAV;
        $positionLeft = ($j + 1) * $espacement;

        echo "<img src='../img/abribus.png' alt='PAV Blanc'
        class='imagePAV' data-couleur='Blanc'
data-ip='$ip' data-index='$j'
        style='
            position: absolute;
            top: 50%;

```

```

        left: {$positionLeft}%;
        transform: translate(-50%, -50%);
        width: 70px;
        opacity: 1;
        z-index: 4;
    '>";
    } // for j
} // if 1 bus
echo "</div>";
//
// FIN BUS et ABB
//
?>
</div>
</section>
</main>
<script>
    document.addEventListener("DOMContentLoaded", function () {
        const modal = document.getElementById("playerModal");
        const openModalButton =
document.getElementById("openModalButton");
        const closeModalButton = document.querySelector(".close-btn");
        const playerForm = document.getElementById("playerForm");

        openModalButton.addEventListener("click", function () {
            modal.style.display = "flex";
        });

        closeModalButton.addEventListener("click", function () {
            modal.style.display = "none";
        });

        playerForm.addEventListener("submit", function (event) {
            event.preventDefault();

            const formData = new FormData(playerForm);
            fetch('update_joueurs.php', {
                method: 'POST',
                body: formData
            })
                .then(response => response.text())
                .then(() => {
                    modal.style.display = "none";
                    location.reload();
                })
                .catch(error => console.error("Erreur :", error));
        });
    });
</script>

```

```

<script>
    function showPodium() {
        loadPodium(); // exécute la fonction loadPodium du script
loadPodium.js
        document.getElementById("podiumModal").style.display = "flex";
    }
    function hidePodium() {
        document.getElementById("podiumModal").style.display = "none";
    }
    // Fermeture quand on clique en dehors de la zone modale (hors
.modal-content)
    document.getElementById("podiumModal").addEventListener("click",
function () {
    hidePodium();
});

    // Fermeture avec la touche Échap
    document.addEventListener("keydown", function (event) {
        if (event.key === "Escape") {
            hidePodium();
        }
    });
</script>
<script src="js/gestion-de-la-partie/loadPodium.js"></script>
<script src="js/gestion-de-la-partie/updateCamion.js"></script>
<script src="js/gestion-de-la-partie/updateTimer.js"></script>
<?php
require 'fin.inc.php';
?>

```

Annexe 7: Serveur Web - parametre-du-jeu.php

```

<?php
require 'debut.inc.php';
require 'head.inc.php';
require 'bodyHeader.inc.php';
?>

<main>
    <section>
        <h2>Paramétrage du Jeu</h2>

        <?php
            // Inclusion du fichier de connexion
// include '/var/www/html/private-estom/db_connect.php';
            require 'private-estom/db_connect.php';

            // Initialisation des paramètres
            $tempsPartie = 180;
            $options = 1;

```

```

$ptsRecolte = 10;
$nbrPAV = 10;
$malusCollision = 2;
$nbrBOM_V = 1;
$nbrBOM_J = 1;
$nbrBOM_N = 1;
$nbrBOM_B = 1;
// Récupération des paramètres existants
$sql = "SELECT Options, PtsRecolte, NbrPAV, MalusCollision,
NbrBOM_V, NbrBOM_J, NbrBOM_N, NbrBOM_B, duree FROM Config WHERE Id = 1";
$result = $conn->query($sql);

if ($result && $result->num_rows > 0) {
    $row = $result->fetch_assoc();
    $options = $row['Options'];
    $ptsRecolte = $row['PtsRecolte'];
    $nbrPAV = $row['NbrPAV'];
    $malusCollision = $row['MalusCollision'];
    $nbrBOM_V = $row['NbrBOM_V'];
    $nbrBOM_J = $row['NbrBOM_J'];
    $nbrBOM_N = $row['NbrBOM_N'];
    $nbrBOM_B = $row['NbrBOM_B'];
    $tempsPartie = $row['duree'];
} // if

// Si le formulaire est soumis
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    // Mise à jour du temps de la partie
    // Validation des entrées utilisateur avec double vérification
des valeurs!
    $nouveauTempsPartie = filter_var($_POST['temps_partie'],
FILTER_VALIDATE_INT, [
        'options' => ['min_range' => 30, 'max_range' => 600]
    ]) ?? $tempsPartie;

    $nouveauNbrPAV = filter_var($_POST['nombre_PAV'],
FILTER_VALIDATE_INT, [
        'options' => ['min_range' => 1, 'max_range' => 10]
    ]) ?? $nbrPAV;
    $nouveauMalusCollision = filter_var($_POST['malus_collision'],
FILTER_VALIDATE_FLOAT);
    if ($nouveauMalusCollision === false || $nouveauMalusCollision
< 0.2 || $nouveauMalusCollision > 5) {
        $nouveauMalusCollision = $malusCollision;
    } // if

    $nouveauNbrBOM_V = filter_var($_POST['nombre_bom_v'],
FILTER_VALIDATE_INT, [

```

```

        'options' => ['min_range' => 1, 'max_range' => 2]
    ]) ?? $nbrBOM_V;
    $nouveauNbrBOM_J = filter_var($_POST['nombre_bom_j'],
FILTER_VALIDATE_INT, [
        'options' => ['min_range' => 1, 'max_range' => 2]
    ]) ?? $nbrBOM_J;
    $nouveauNbrBOM_N = filter_var($_POST['nombre_bom_n'],
FILTER_VALIDATE_INT, [
        'options' => ['min_range' => 1, 'max_range' => 2]
    ]) ?? $nbrBOM_N;
    $nouveauNbrBOM_B = filter_var($_POST['nombre_bom_b'],
FILTER_VALIDATE_INT, [
        'options' => ['min_range' => 1, 'max_range' => 2]
    ]) ?? $nbrBOM_B;

    $nouveauPtsRecolte = filter_var($_POST['pts_recolte'],
FILTER_VALIDATE_INT, [
        'options' => ['min_range' => 1, 'max_range' => 100]
    ]) ?? $ptsRecolte;

    //Option, si Option = 0 alors Option A, si Option = 1 Alors
option B !
    if (isset($_POST['regle']) && $_POST['regle'] === 'option_a')
    {
        $nouveauOptions = 0;
    } else {
        $nouveauOptions = 1;
    } // else

    // Mise à jour dans la base de données
    $sqlUpdate = "UPDATE Config SET Options = ?, PtsRecolte = ?,
NbrPAV = ?, MalusCollision = ?, NbrBOM_V = ?, NbrBOM_J = ?, NbrBOM_N = ?,
NbrBOM_B = ?, duree = ? WHERE Id = 1";
    $stmt = $conn->prepare($sqlUpdate);
    $stmt->bind_param("iiidiiii", $nouveauOptions,
    $nouveauPtsRecolte, $nouveauNbrPAV, $nouveauMalusCollision,
    $nouveauNbrBOM_V, $nouveauNbrBOM_J, $nouveauNbrBOM_N, $nouveauNbrBOM_B,
    $nouveauTempsPartie);
    $stmt->execute();

    header("Location: parametre-du-jeu.php?success=1");
    exit;
} // if
$conn->close();
?>

<div id="Principale">
    <form method="post" action="parametre-du-jeu.php"
id="BoiteRegles">

```

```

<fieldset id="fieldset">
    <legend id="titre">Règles du Jeu</legend>

    <div class="Regle-Container">

        <label for="temps_partie" id="temps_partie">Temps
de la partie (en secondes) :</label>
        <input type="number" id="temps_partie_input"
name="temps_partie" value="<?php echo htmlspecialchars($tempsPartie); ?>"
step="10" min="30" max="600" required>

        <label for="nombre_PAV" id="nombre_PAV">Nombre de
PAV par couleur :</label>
        <input type="number" id="nombre_PAV_input"
name="nombre_PAV" value="<?php echo htmlspecialchars($nbrPAV); ?>" min="1"
max="10" required>

        <label for="malus_collision"
id="malus_collision">Malus de collision :</label>
        <input type="number" id="malus_collision_input"
name="malus_collision" value="<?php echo htmlspecialchars($malusCollision);
?>" step="0.2" min="0.2" max="5" required>

        <label id="optionA_label" for="optionA">
            <input type="radio" name="regle" id="optionA"
value="option_a" <?php echo $options == 0 ? 'checked' : ''; ?>
            <span id="optionA_text">Option A : Points
identiques pour chaque récolte</span>
        </label>
        <div id="optionA_settings">
            <label for="pts_recolte">Points par PAV
récolté :</label>
            <input type="number" id="pts_recolte_input"
name="pts_recolte" value="<?php echo htmlspecialchars($ptsRecolte); ?>"
min="1" max="100">
        </div>

        <label id="optionB_label" for="optionB">
            <input type="radio" name="regle" id="optionB"
value="option_b" <?php echo $options == 1 ? 'checked' : ''; ?>
            <span id="optionB_text">Option B : Points
progressifs selon l'avancement de la récolte</span>
        </label>

        <div id="tableau_Nombres">
            <label for="nombre_bom_j"
id="nombre_jaune">BOM Jaune:</label>

```

```

        <input type="number" id="nombre_jaune_input"
name="nombre_bom_j" value="<?php echo htmlspecialchars($nbrBOM_J); ?>" min="0"
max="2" required>

        <label for="nombre_bom_v" id="nombre_vert">BOM
Verte:</label>

        <input type="number" id="nombre_vert_input"
name="nombre_bom_v" value="<?php echo htmlspecialchars($nbrBOM_V); ?>" min="0"
max="2" required>

        <label for="nombre_bom_b" id="nombre_bleu">BOM
Bleu:</label>

        <input type="number" id="nombre_bleu_input"
name="nombre_bom_b" value="<?php echo htmlspecialchars($nbrBOM_B); ?>" min="0"
max="2" required>

        <label for="nombre_bom_n" id="nombre_noir">BOM
Noir:</label>

        <input type="number" id="nombre_noir_input"
name="nombre_bom_n" value="<?php echo htmlspecialchars($nbrBOM_N); ?>" min="0"
max="2" required>
    </div>
</div>
</fieldset>

    <button id="savebt" type="submit">Sauvegarder les
paramètres</button>
</form>

<div id="IconesParametre">
    <?php
    $couleurs = [
        "Jaune" => $nbrBOM_J,
        "Vert" => $nbrBOM_V,
        "Bleu" => $nbrBOM_B,
        "Noir" => $nbrBOM_N,
        "Blanc" => 1
    ];
    $maxBOM = max($nbrBOM_V, $nbrBOM_J, $nbrBOM_N, $nbrBOM_B);
    $position = 1; // Position de chaque colonne dans la
grille

    foreach ($couleurs as $couleur => $nombre) {
        echo '<div id="ColonneBOM" . $couleur . "'
style="grid-column: ' . $position . ' ; grid-row: 1;">';

        // Afficher 1 ou 2 BOM par couleur
        for ($i = 0; $i < $maxBOM; $i++) {
            if ($couleur == "Blanc") { // si Bus

```

```

        echo '<div id="BOM-' . $couleur . '-' . $i .
'" class="bom-container">
        
        <span id="BOM-IP-' . $couleur . '-' .
$i . '" class="ip-text"></span>
        </div>';
    } else {
        echo '<div id="BOM-' . $couleur . '-' . $i .
'" class="bom-container">
        
        <span id="BOM-IP-' . $couleur . '-' .
$i . '" class="ip-text"></span>
        </div>';
    } // else BOM
} // for
echo '</div>';
echo '<div id="ColonnePAV' . $couleur . '"
style="grid-column: ' . $position . '; grid-row: 2;">';
for ($i = 0; $i < 10 ; $i++) {
    if ($couleur == "Blanc") { // si ABB
        echo '<div class="LignePAV">
        
        <span id="PAV-IP-' . $couleur . '-' .
$i . '" class="ip-text"></span>
        </div>';
    } else {
        echo '<div class="LignePAV">
        
        <span id="PAV-IP-' . $couleur . '-' .
$i . '" class="ip-text"></span>
        </div>';
    } // else
} // for
echo '</div>';
$position++; // Passe à la colonne suivante
} // for
?>
</div>

</div>
</section>
</main>

```

```

<script src="js/parametre-du-jeu/updateConnexion.js"></script>
<script>
    // Fonction pour gérer la visibilité du champ "Pts par PAV"
    function togglePointsField() {
        const optionAInput =
document.querySelector('input[name="regle"][value="option_a"]');
        const pointsField = document.getElementById("optionA_settings");

        // Afficher ou masquer le champ selon l'option sélectionnée
        if (optionAInput.checked) {
            pointsField.style.display = "block";
        } else {
            pointsField.style.display = "none";
        }
    }

    // Ajout d'événement
    document.addEventListener("DOMContentLoaded", () => {
        // Exécuter une première fois pour régler l'affichage initial
        togglePointsField();

        // Ajouter des "écouteurs" sur les options de règle
        document.querySelectorAll('input[name="regle"]').forEach(input =>
{
            input.addEventListener("change", togglePointsField);
        });
    });
</script>
<?php
require 'fin.inc.php';
?>

```