



# PROJET HUNTING TOWN

Rapport Individuel

GIBAUD Nicolas



## Table des matières

Table des matières .....	1
Table des illustrations.....	2
Introduction.....	3
1.Présentation du projet .....	3
2.Phase de recherche .....	4
A.Choix initiaux.....	4
B.Choix spécifiques.....	5
a)Choix de la carte SIM.....	5
b)Interface de communication.....	6
3.Localisation GPS .....	7
A.Répartition des tâches .....	7
B.Communication 4G.....	8
a.M-Center .....	9
b.Utilisation de l'API REST d'Hologram .....	10
c.Configuration du module 4G .....	12
d.Utilisation du FTP .....	14
e.Implémentation sur la carte .....	17
C.Tests effectués .....	22
D.Mesures suites aux tests.....	22
4.Lancement des vidéos .....	22
A.Description .....	22
B.Réalisation de l'interface graphique .....	23
C.Configuration du module .....	25
D.Implémentation .....	26
E.Tests.....	28
5.Récupération du son en direct .....	28
A.Objectif.....	28
B.Solution envisagée .....	28
C.Configuration du module .....	29
D.Problèmes rencontrés.....	31
Conclusion .....	32

## Table des illustrations

<a href="#">Figure 1 - Organisation globale du système</a>	5
<a href="#">Figure 2 - Logo Hologram</a>	6
<a href="#">Figure 3 - description utilisation API</a>	7
<a href="#">Figure 4 - Fonctionnement API</a>	7
<a href="#">Figure 5 - Module 4G LTE Click</a>	8
<a href="#">Figure 6 - Envoi de commandes AT via m-center</a>	9
<a href="#">Figure 7 - Page Network m-center</a>	10
<a href="#">Figure 8 - Topic de gestion des messages</a>	11
<a href="#">Figure 9 - Fichier JSON de réponse de l'API</a>	11
<a href="#">Figure 10 - Liste des messages via l'interface d'Hologram</a>	12
<a href="#">Figure 11 - serveur wing utilisé pour les tests et démonstrations</a>	15
<a href="#">Figure 12 - Configuration FTP via commandes AT</a>	15
<a href="#">Figure 13 - Documentation commande UFTPC</a>	16
<a href="#">Figure 14 - Interaction avec le serveur via commandes AT</a>	16
<a href="#">Figure 15 - Fonction sendAT</a>	17
<a href="#">Figure 16 - Type AT command</a>	17
<a href="#">Figure 17 - Les différents types de commandes possibles</a>	18
<a href="#">Figure 18 - Exemple de test de fin de réponse</a>	18
<a href="#">Figure 19 - Exemple de test du résultat de l'exécution de la commande AT</a>	19
<a href="#">Figure 20 - Fonction d'initialisation du module</a>	19
<a href="#">Figure 21 - Fonction d'initialisation de la connexion Internet</a>	20
<a href="#">Figure 22 - Fonction de configuration de la connexion FTP</a>	20
<a href="#">Figure 23 - Fonction d'envoi des coordonnées GPS via FTP</a>	20
<a href="#">Figure 24 - Test de fin de réception de la réponse à une commande de type "AT C UFTPC"</a>	21
<a href="#">Figure 25 - Test de succès de la commande</a>	21
<a href="#">Figure 26 - Logo TKinter</a>	23
<a href="#">Figure 27 - Logo python</a>	23
<a href="#">Figure 28 - Interface de choix des vidéos</a>	24
<a href="#">Figure 29 - Fonction d'envoi de l'ordre vidéo via FTP</a>	24
<a href="#">Figure 30 - Fonction de connexion au serveur FTP</a>	24
<a href="#">Figure 31 - Fonction de récupération des coordonnées GPS</a>	25
<a href="#">Figure 32 - Commande AT permettant de récupérer un fichier sur un serveur FTP</a>	25
<a href="#">Figure 33 - Configuration Timer</a>	26
<a href="#">Figure 34 - Callback de déclenchement du timer</a>	26
<a href="#">Figure 35 - appel à la fonction getVideo ftp</a>	27
<a href="#">Figure 36 - Vérification de l'exécution de la commande +URDFILE</a>	27
<a href="#">Figure 37 - Test de la valeur de la variable valueBluetooth</a>	28
<a href="#">Figure 38 - Logo VoLTE</a>	29
<a href="#">Figure 39 - Logo VOIP</a>	29
<a href="#">Figure 40 - Interface audio des modules u-blox</a>	30
<a href="#">Figure 41 - Datasheet sur la configuration du mode de l'I2S</a>	31

## Introduction

Dans ce rapport individuel, je vais vous présenter de manière concise en quoi consiste le projet Hunting Town. Je détaillerai les différents choix que l'on a fait ainsi que le travail que j'ai effectué dans le groupe. Je vous parlerai aussi des différents problèmes que l'on a rencontrés et les solutions que l'on a apportées en réponse. Cela permet de revenir plus en détail sur certains points de travail que l'on ne pouvait pas aborder dans le rapport de groupe.

### 1. Présentation du projet

Ce projet est réalisé dans le cadre du Master 1 « Chef de Projet systèmes Embarqués » au sein de l'école *Campus YNOV Bordeaux*. La mise en contact de l'entreprise *Hunting Town* et de son directeur M. Thibaut Giuseppi avec l'école a été faite par un élève, Jean-Gabriel Massicot. Lui-même a été mis en relation avec la société grâce sa compagne y ayant travaillée.

À la suite d'explications sur le fonctionnement technique du jeu, sa constatation a été que le système mis en place est complexe, présente des difficultés techniques impactant le Gameplay, et que la conception d'un objet connecté serait à la fois intéressante pour l'entreprise, passionnant à mettre en place et entrant dans le cadre des compétences de l'Ecole, de la formation et du projet.

C'est après avoir demandé confirmation à M. Pierre Aubry, responsable de la formation, et après avoir effectué un premier rendez-vous téléphonique où étaient présents Jean-Gabriel Massicot, M. Pierre Aubry, et M. Thibaut Giuseppi, que la décision d'organiser ce projet a été prise.

Le projet Hunting Town est un projet réalisé dans le but d'améliorer la solution existante de l'entreprise Hunting Town. En effet, cette entreprise organise des escape games dans la ville de Bordeaux. Dans cette situation où les clients sont en extérieur, l'entreprise a besoin de communiquer avec eux afin de leur transmettre des indices et pour récupérer le son émis par les clients. Pour cela, l'entreprise a une solution de base utilisant différents portables qu'ils confient aux clients mais cette solution n'est pas du tout optimale et il arrive quelque fois à l'entreprise de perdre la connexion avec les joueurs. Notre objectif est donc d'améliorer grandement cette solution.

Actuellement, l'entreprise Hunting Town possède l'équipement suivant :

- Une GoPro filmant le périple des joueurs
- Une montre
- Un premier téléphone portable connecté à celui du maître du jeu permettant de récupérer le son ambiant des joueurs
- Un deuxième portable accroché au poignet d'un des joueurs permettant de diffuser les vidéos associés

- Nécessite aussi l'utilisation du portable de l'un des participants afin de communiquer les réponses au maître du jeu

Pour améliorer cette solution actuelle nécessitant l'utilisation de trois portables différents, nous allons faire un objet connecté pouvant communiquer avec l'entreprise pendant le jeu. Les principaux objectifs sont :

- L'objet doit être équipé d'un écran assez large pour pouvoir voir les indices vidéo
- Pouvoir accéder à la position GPS de l'objet à tout moment
- Communiquer avec l'objet connecté sans coupure
- Jouer des vidéos ainsi que le son associé à un volume suffisamment fort pour que tous les joueurs puissent l'entendre
- Dimension de l'objet : Ne doit pas être trop encombrant ni trop lourd
- Récupérer le son ambiant des joueurs et l'envoyer au maître du jeu
- Alimentation : deux heures au minimum

## 2. Phase de recherche

### A. Choix initiaux

La première partie du projet était donc une phase de recherche durant laquelle nous avons essayé de trouver les solutions les plus adaptés pour les contraintes exposées (voir rapport de groupe pour les tableaux résumés des recherches).

Après cette période de recherche, nous avons fait les choix suivants :

- Protocole de communication entre le Game Master et les joueurs : **4G LTE Click**
- Carte écran : **BBONE-BLACK**
- Carte communication : **STM32 équipé de cartes d'extension click pour les différentes fonctionnalités**
  - Module 4G : **4G LTE Click Board équipe du module LARA-R211**
  - Module GPS : **Nano GPS Click**
- Protocole de communication entre la carte écran et la carte communication : **Bluetooth**

Nous avons aussi fait un choix d'organisation des objets connectés sur le joueur. C'est-à-dire qu'en fonction des contraintes que l'on avait (écran assez large, carte communication pouvant prendre pas mal de place, besoin d'avoir un micro connecté à cette carte communication) nous avons décidé de l'architecture suivante :

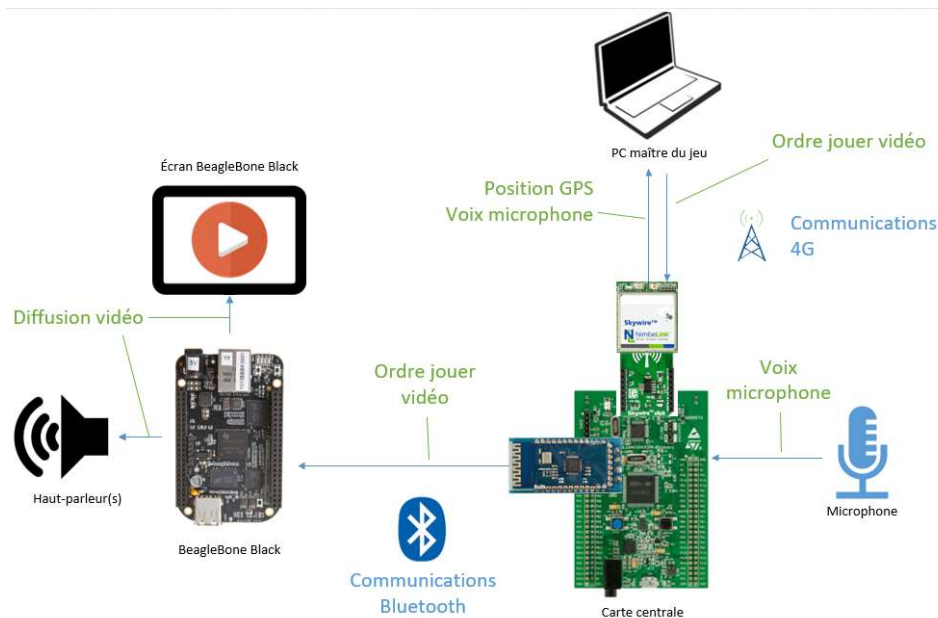


Figure 1 - Organisation globale du système

On voit donc que la partie communication est gérée par une carte qui se trouvera au niveau du torse de l'un des joueurs. Cette carte sera composée de modules click qui seront pluggés à la carte STM32. La carte communication aura donc pour rôle la localisation GPS, l'envoi du son en continu récupéré par le micro et de la position GPS des joueurs, la réception des ordres des indices à afficher et l'envoi de ces ordres à la carte écran qui elle, aura pour rôle de réceptionner les ordres des indices à afficher et de jouer les vidéos associées sur un écran.

## B. Choix spécifiques

On va maintenant aborder les choix spécifiques qui ont été fait durant ce projet. Ces choix n'ont pas forcément été effectués durant la phase de recherche lors du début du projet mais plus durant l'évolution du projet.

Je vais seulement parler des choix qui m'ont concerné directement afin de voir l'impact qu'ils ont eu sur mon travail.

### a) Choix de la carte SIM

#### - Hologram

Nous avons besoin d'une carte SIM s'adaptant parfaitement à nos besoins. Nous nous sommes donc d'abord renseignés sur les solutions IOT existantes et nous avons découvert Hologram qui semblait être la solution la plus optimale pour notre projet. En effet, Hologram à l'avantage d'avoir des prix avantageux, d'être facile à mettre en place et de fournir à ses clients une API REST facile à utiliser, ce qui était un point très important car comme on le verra plus tard, cela était nécessaire pour nous à ce moment du projet.



Figure 2 - Logo Hologram

Mais on s'est rendu compte plus tard dans le projet que le choix d'Hologram était en fait une erreur de par divers aspects. Notamment sur le fait qu'Hologram ne met pas à disposition de numéro de téléphone. Ce qui est un problème non négligeable pour la partie de la récupération du son en direct. Un autre défaut d'Hologram était le fait que le forfait soit configuré de manière à être utilisé pour échanger peu de données. Encore une fois, cela n'était pas adéquat pour la partie de la récupération du son en direct car cela nécessitait d'utiliser un montant de données non négligeable.

### - Forfait classique

Après réflexion, on s'est rendu compte qu'un forfait classique téléphonique était plus optimale. En effet, il palliait les différents problèmes liés au choix d'Hologram en nous permettant d'envoyer un montant de données bien plus important ainsi qu'en nous fournissant un numéro de téléphone nous permettant de rendre possible la récupération du son en direct. Néanmoins, ce choix nous oblige à trouver une solution alternative à l'API REST qui était fourni par Hologram.

#### b) Interface de communication

Ce choix est très important, il permet de décider de quelle façon l'interface graphique côté maître du jeu va communiquer avec le module 4G.

### - API REST d'Hologram

L'API REST fournie par Hologram semble être un excellent choix. Une API, de manière générale est une interface entre un utilisateur et un service. Nous l'utiliserions, par exemple de la manière suivante dans le cas d'envoi des coordonnées GPS :

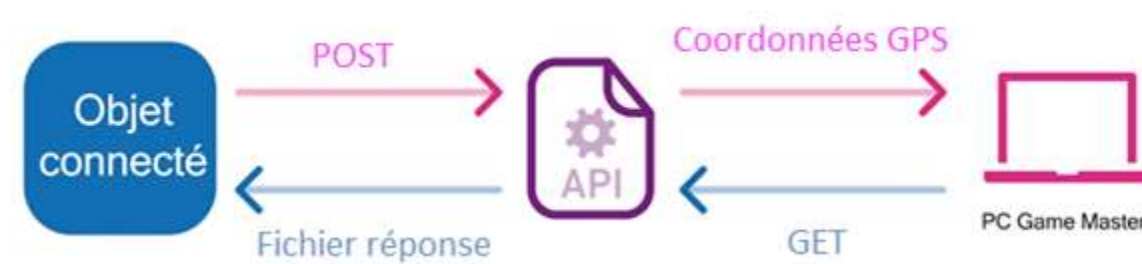


Figure 3 - description utilisation API

On communiquera donc avec l'API via requêtes HTTP. On pourra voir avec plus de détails par la suite, la façon dont on communiquait avec l'API.

Mais suite à notre choix de changer de solution concernant la carte SIM et de ne plus utiliser Hologram. Nous avons dû oublier tout le travail effectué sur l'API et trouver une nouvelle solution utilisable.

### - Serveur FTP

Après différents essais sur plusieurs autres solutions, nous avons fait le choix d'utiliser le File Transfer Protocol. Le File Transfer Protocol est un protocole de communication dont le but est le transfert de fichier. Il fonctionne via TCP.

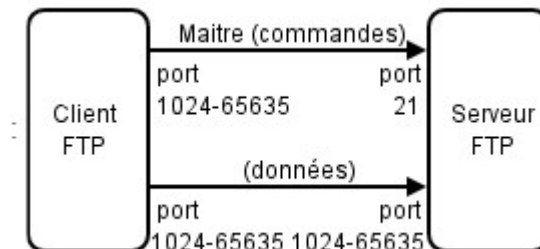


Figure 4 - Fonctionnement FTP

Nous aurons donc le module 4G ainsi que l'ordinateur hébergeant l'interface graphique qui seront connectés en tant que client à un serveur FTP. Pour les tests et démonstrations actuels, nous utilisons un serveur public car nous manquons de temps mais par la suite, il sera très facile de mettre en place un serveur FTP uniquement destiné à l'utilisation d'Hunting Town.

Cette solution nous semblait la plus optimale et nous avons donc fait le choix de l'adopter de la mettre avant la fin du projet.

## 3. Localisation GPS

### A. Répartition des tâches



Après les choix généraux d'effectués, notre client, le patron de l'entreprise Hunting Town nous a demandé de travailler en priorité sur la localisation GPS, point très important pour l'objet connecté. Nous avons donc décidé de prioriser les tâches suivantes :

Tableau 1 - Répartition des tâches GPS

Capture GPS	Communication	Réception & IHM	Alimentation
Choix module	Choix module	Affichage d'une carte	Etude Besoins
Communication Carte-module	Communication Carte-module	Récupérations données	Choix batterie
Récupération données	Choix SIM	Afficher Données carte	Carte Alimentation
	Connection internet	Gestion multi-Groupe -> Adressage	
	Communication API		

Pour la réalisation de ces tâches, nous nous sommes répartis initialement de la manière suivante :

#### Communication 4G :

- Clément
- Nicolas

#### Localisation GPS :

- Antoine
- Axel

## B. Communication 4G

Mon travail s'est donc porté sur la partie communication 4G. Comme le montre le tableau, il y avait différentes tâches à réaliser dans cette partie. La première consistait à choisir le module 4G selon différents critères. Notre choix s'est porté sur la carte click **MIKROE-2527** qui sera directement connectée à la carte STM32 pour les tests. Ces deux cartes communiqueront via UART.



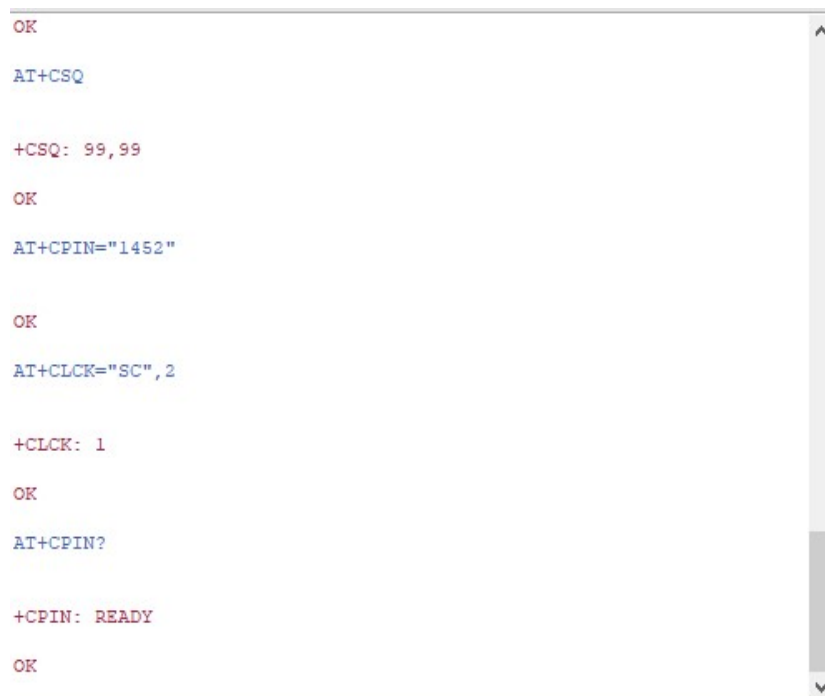
Figure 5 - Module 4G LTE Click

Nous avons choisi la carte **4G LTE Click** équipé du module LARA-R211 qui sera directement connecté à la carte STM32 par rapport à l'organisation choisie de base.

En effet, nous avons choisi que toute la partie communication (voix des clients, position GPS, indices) se fera sur la carte STM32 grâce aux modules Click qui se situera dans un boîtier fixé sur la poitrine de l'un des clients. Ensuite, une BeagleBone Black situé sur l'avant-bras de l'un des clients aura pour rôle de stocker les indices, et de les afficher en temps voulu via un écran connecté à la carte. La carte STM32 enverra les ordres d'indice à afficher reçus par 4G à la BeagleBone Black afin qu'elle puisse lancer la vidéo associée. Ces deux cartes communiqueront ensemble via Bluetooth.

#### a. M-Center

La carte d'extension **MIKROE-2527** utilise comme module le **lara-r211**. En attendant la réception de la commande, j'ai donc effectué des recherches sur la meilleure façon de mettre en place une communication 4G avec ce module. J'ai pu observer que le module marche sur un principe de commande AT. Ce sont des commandes que l'on va envoyer au module et il va effectuer une action en retour. C'est donc de cette façon que l'on va communiquer avec le module. J'ai aussi découvert que **ublox** nous met à disposition le logiciel **m-center**, qui permet d'interagir avec le module par une interface graphique. Le logiciel va tout de même communiquer avec le module grâce à des commandes AT en fonction de ce que l'on fait dessus. Nous pouvons donc observer les commandes AT utilisées en fonction de ce que l'on veut faire comme on peut le voir ci-dessous :



The screenshot shows a text-based interface with a scrollable area. It displays a sequence of AT commands sent to a module (in blue) and the corresponding responses received (in red). The commands include setting the baud rate, setting the PIN, and checking the PIN status. The responses indicate successful configuration and readiness.

```
OK
AT+CSQ

+CSQ: 99,99
OK
AT+CPIN="1452"

OK
AT+CLCK="SC",2

+CLCK: 1
OK
AT+CPIN?

+CPIN: READY
OK
```

Figure 6 - Envoi de commandes AT via m-center

Les commandes en bleu sont celles qui sont envoyées au module et les réponses sont en rouge. Ici, nous pouvons observer la phase de connexion à la carte SIM grâce au code PIN. On peut donc voir

que pour cela, nous utilisons la commande **AT+CPIN=<code pin>** et on peut ensuite observer le résultat de l'opération via la commande de read **AT+CPIN?** qui nous renvoie l'état de la carte SIM.

Lors de la réception du module, nous avons pu commencer à tester le logiciel **m-center**. La communication entre la carte et le PC se fait via UART par câble USB. Nous avons pu rapidement communiquer avec le module et récupérer certaines informations comme le nom du module, la version du firmware, ou même l'heure actuelle.

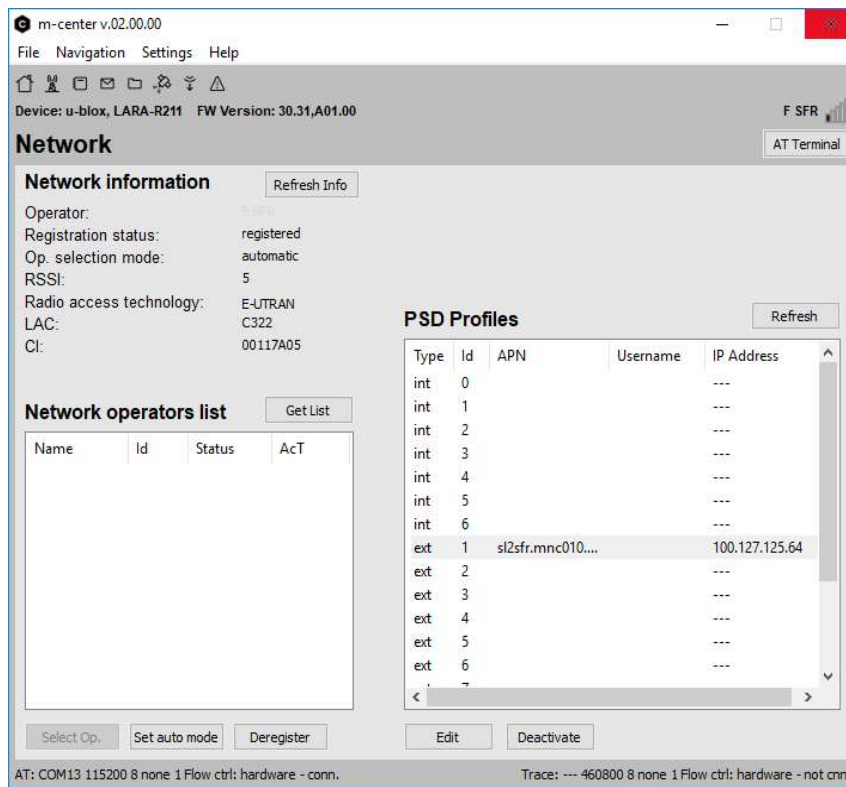


Figure 7 - Page Network m-center

Le logiciel m-center va donc nous permettre de gagner beaucoup de temps sur les différents tests concernant les commandes AT à envoyer au module 4G afin d'obtenir la configuration voulue.

#### b. Utilisation de l'API REST d'Hologram

Comme énoncé plus tôt, nous avons besoin de transmettre les coordonnées GPS depuis le module 4G jusqu'au PC qui contiendra l'interface graphique. Pour cela, nous avons besoin d'un intermédiaire entre ces deux parties. Notre première idée est donc d'utiliser l'API d'Hologram en tant qu'intermédiaire entre le module 4G et l'interface graphique.

Après une rapide étude technique, on peut observer que la gestion des messages se fait sur le topic suivant :

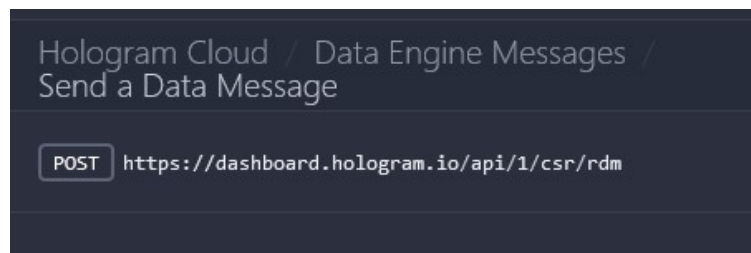


Figure 8 - Topic de gestion des messages

Il suffit donc d'effectuer des POST et des GET à cette adresse afin de respectivement envoyer et recevoir les données GPS. L'API gère la réception des messages sous le format JSON. C'est-à-dire qu'il faudra absolument envoyer les coordonnées GPS sous forme d'un fichier JSON afin que l'API puisse les recevoir et les stocker. Lors du GET, nous récupérerons donc un fichier JSON contenant de nombreuses données dont les coordonnées GPS.



Figure 9 - Fichier JSON de réponse de l'API

Comme on peut le voir, le GET nous renvoie la liste de tous les messages qui ont été envoyés, le dernier étant le n°0. Les données intéressantes se trouvent dans data et sont encodées en base 64. Tous les messages seront donc envoyés à partir de l'objet connecté afin qu'ils soient récupérés par l'interface graphique associé, on peut voir ici les messages envoyés pour les différents tests effectués :

Message sent from	DATA	TOPICS	Timestamp
Hun_Tow_01 (14279)	Latitude:4450.5378,N Longitude:00034.6666,W	#_RESTAPL_ Hun_Tow_01 (14279)	Feb 26 2019, 16:15:01
Hun_Tow_01 (14279)	Latitude:4450.5694,N Longitude:00034.6778,W	#_RESTAPL_ Hun_Tow_01 (14279)	Feb 26 2019, 16:14:52
Hun_Tow_01 (14279)	Latitude:4450.5799,N Longitude:00034.6931,W	#_RESTAPL_ Hun_Tow_01 (14279)	Feb 26 2019, 16:14:32
Hun_Tow_01 (14279)	Latitude:4450.5992,N Longitude:00034.7031,W	#_RESTAPL_ Hun_Tow_01 (14279)	Feb 26 2019, 16:13:57
Hun_Tow_01 (14279)	Latitude:4450.5994,N Longitude:00034.6948,W	#_RESTAPL_ Hun_Tow_01 (14279)	Feb 26 2019, 16:13:47
Hun_Tow_01 (14279)	Latitude:4450.5148,N Longitude:00034.6892,W	#_RESTAPL_ Hun_Tow_01 (14279)	Feb 26 2019, 16:13:38
Hun_Tow_01 (14279)	Latitude:4450.5269,N Longitude:00034.7026,W	#_RESTAPL_ Hun_Tow_01 (14279)	Feb 26 2019, 16:13:19
Hun_Tow_01 (14279)	Latitude:4450.4784,N Longitude:00034.6930,W	#_RESTAPL_ Hun_Tow_01 (14279)	Feb 26 2019, 16:13:09
Hun_Tow_01 (14279)	Latitude:4450.5201,N Longitude:00034.6992,W	#_RESTAPL_ Hun_Tow_01 (14279)	Feb 26 2019, 16:13:04

Figure 10 - Liste des messages via l'interface d'Hologram

### c. Configuration du module 4G

Une des principales tâches de la mise en place de la communication 4G est l'étude des commandes AT à utiliser afin de configurer le module 4G de la manière la plus optimale possible selon notre besoin principal qui est :

- De communiquer à l'API d'Hologram les données GPS récupérées préalablement qui peut se décomposer en :
  - Connexion à Internet
  - Mise en place de la communication HTTP avec l'API d'Hologram
  - Envoi de requêtes de type POST contenant les coordonnées GPS

On va donc voir les commandes AT utilisées pour ces différentes étapes :

- **Connexion à Internet**

- **+COPS** : Sélection de l'opérateur réseau à utiliser
  - **AT+COPS=0** → Connection automatique au meilleur opérateur disponible.
- **+UPSD** : Gestion de toute la configuration réseau. Pour communiquer via Internet, il faut créer un profil PSD. Ce profil contiendra les informations de l'APN. Il faut donc renseigner les informations nécessaires à l'utilisation de l'APN d'Hologram.

- **AT+UPSD=0,1,"Hologram"** → Ajout de l'utilisation de l'APN d'Hologram pour le profil réseau 0.
  - **AT+UPSD=0,100,1** → Définition du profil PSD à utiliser. On utilise donc celui que l'on vient de créer.
- **+UPSDA** : Permet d'effectuer des actions sur le profil PSD voulu.
  - **AT+UPSDA=0,1** → Sauvegarde des informations du profil PSD 0 afin d'avoir à éviter de le recréer à chaque fois.
  - **AT+UPSDA=0,3** → Activation du contexte PDP sur le profil PSD courant. C'est après cette étape que nous sommes connectés à Internet et que nous sommes en mesure d'effectuer des PING sur [www.google.com](http://www.google.com) par exemple
- **Mise en place de la communication HTTP avec l'API d'Hologram**
  - **+UHTTP** : Gestion de toute la configuration HTTP. La configuration se fait en précisant les paramètres voulus tels que le nom de domaine du serveur ou le port de communication à utiliser.
    - **AT+UHTTP=0** → Reset du profil HTTP. On efface toutes les traces de l'ancienne configuration.
    - **AT+UHTTP=0,1,"dashboard.hologram.io"** → On précise l'adresse du serveur avec qui on veut communiquer. Dans notre cas, c'est bien sur l'API d'Hologram.
    - **AT+UHTTP=0,5,80** → On renseigne le fait que le module va communiquer avec le port lors de cette communication http.
    - **AT+UHTTP=0,6,1,2** → Activation de la communication sécurisée. Cela permet d'accéder aux adresses **HTTPS** comme l'est celle de l'API d'Hologram.
    - **AT+UDNSRN=0,"dashboard.hologram.io"** → Résolution DNS de l'adresse du serveur afin d'obtenir son adresse IP.
- **Envoi de requêtes POST contenant les coordonnées GPS**
  - **+UDWNFILE** : Création d'un fichier qui contiendra les données de latitude et de longitude. Ce fichier sera au format JSON et sera envoyé à l'API d'Hologram.
    - **AT+UDWNFILE="LatitudeLongitude",77** → Création du fichier *LatitudeLongitude.json* de taille 77 octets. Après avoir tapé cette commande, il faut entrer la chaîne de caractère que contiendra le fichier.
  - **+UHTTPC** : Permet de gérer l'envoi des requêtes HTTP de tous types. On va donc l'utiliser dans notre cas afin d'envoyer des requêtes POST.

- **AT+UHTTPC=0,4,"/api/1/csr/rdm?apikey=2bPkIUk5bQwezsMckFc7lZkWQcxLTg","responseFilename","fichierJSON",4** → Ceci est donc la commande qui nous permet d'envoyer le fichier JSON contenant la latitude et la longitude à l'API d'Hologram. Le premier 4 correspond au type de requête donc dans notre cas, un POST d'un fichier. Il y a ensuite l'adresse à laquelle faire la requête afin que le message soit bien pris en compte par l'API. Le paramètre suivant est le fichier de réponse de la requête contenant son état (failed ou success) et vient ensuite le fichier JSON de données créé précédemment. Le dernier quatre indique que le fichier de données est au format JSON.

Voici donc les commandes AT utilisées afin de mettre en place la configuration de base du module afin qu'il puisse transmettre les données GPS. Les recherches associées à l'utilisation de ces commandes ont été grandement facilitées par la documentation très claire de **ublox** ainsi que par l'utilisation du logiciel **m-center** qui nous a permis de tester et de comprendre le fonctionnement de ces différentes commandes.

#### d. Utilisation du FTP

Lors de la suite du projet, nous nous sommes rendu compte que le choix d'utiliser une carte SIM Hologram n'était pas le meilleur choix possible pour différentes raisons qui seront détaillés par la suite. Nous avons donc décidé d'abandonner cette solution et tout le travail effectué dessus, notamment l'utilisation de leur API, afin d'avoir à la fin une solution permettant d'atteindre tous les objectifs possibles.

Il a donc fallu trouver et mettre en place une solution alternative. Notre choix s'est porté sur l'utilisation d'une carte SIM classique couplé à l'utilisation d'un serveur FTP qui servira d'intermédiaire entre le maître du jeu et les joueurs en remplacement du HTTP qui permettait de communiquer avec l'API d'Hologram.

Nous aurons donc le module 4G ainsi que l'ordinateur hébergeant l'interface graphique qui seront connectés en tant que client à un serveur FTP. Pour les tests et démonstrations actuels, nous utilisons un serveur public car nous manquons de temps mais par la suite, il sera très facile de mettre en place un serveur FTP uniquement destiné à l'utilisation d'Hunting Town.

### Mise en place

Concernant la mise en place du FTP dans notre solution, il y a eu différentes étapes. La première étant bien sur la recherche d'un serveur FTP utilisable pour les différents tests. En effet, après quelques tentatives, nous nous sommes rendu compte que l'on n'avait pas le temps de mettre en place un serveur FTP pour les échéances qui arrivaient. On a donc fait le choix d'utiliser un serveur FTP public disponible gratuitement. Notre choix s'est porté sur **wftpservers**. Nous possédons les droits d'écriture et de lecture dans le dossier uploads et n'avons pas de contraintes au niveau des requêtes, ce qui est important car on vérifiera le fichier des coordonnées environ toutes les 5 secondes afin d'actualiser la position des joueurs sur la carte.



Figure 11 - serveur wing utilisé pour les tests et démonstrations

Une des raisons principales qui a fait que nous avons choisi d'utiliser le FTP est le fait qu'il est très bien supporté par le module. En effet, des commandes AT existent spécifiquement pour l'utilisation du FTP et il semblait simple à mettre en place sur le module 4G.

Afin d'utiliser le FTP via le module 4G, deux étapes sont nécessaires :

- **La phase de renseignement**

Ici, on va renseigner toutes informations nécessaires à la connexion à un serveur FTP, c'est-à-dire :

- L'adresse du serveur
- Le pseudo client
- Le mot de passe client
- Le port de communication du serveur
- Si la communication est sécurisée

Ces informations permettent bien sûr de configurer la connexion au serveur FTP. Cela se fait par le biais de la commande **+UFTP** qui permet de renseigner ces différentes informations via les paramètres de cette commande :

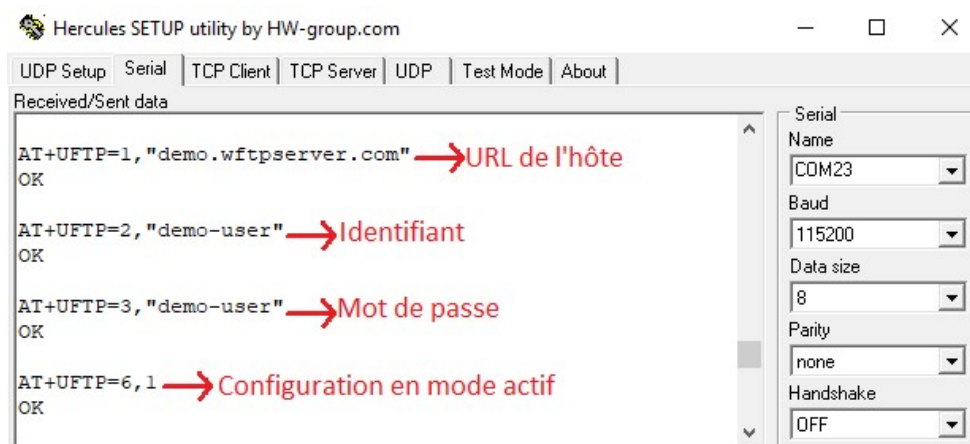


Figure 12 - Configuration FTP via commandes AT



Une fois cela effectué, il ne nous reste plus qu'à nous connecter au serveur.

## - La phase de connexion et d'action

Maintenant nous allons nous connecter et effectuer les différentes actions que l'on souhaite. Cela se fait via la commande **+UFTPC**. Cette commande s'occupe de tout ce qui est inhérent à la communication avec un serveur FTP.

## 27.2 FTP command +UFTPC

+UFTPC						
Modules	TOBY-L200-02S TOBY-L200-03A TOBY-L200-03S TOBY-L201 TOBY-L210-02S TOBY-L210-03A TOBY-L210-03S TOBY-L210-62S TOBY-L220 TOBY-L280 MPC1-L200-02S MPC1-L200-03S MPC1-L201 MPC1-L210-02S MPC1-L210-03S MPC1-L220 MPC1-L280 LARA-R2 TOBY-R2 SARA-U2 LISA-U2 LISA-U1 SARA-G4 SARA-G340 SARA-G350 LEON-G1					
Attributes	Syntax	PIN required	Settings saved	Can be aborted	Response time	Error reference
	partial	No	No	No	-	<a href="#">Appendix A.9.1</a>

### 27.2.1 Description

Triggers the FTP actions corresponding to the <op\_code> parameter. The final result code indicates if sending the command request to the FTP process was successful or not. The +UUFTPCR (FTP command result) URC returns to the user the final result of the FTP command previously sent with +UFTPC. As well, the +UUFTPCD

Figure 13 - Documentation commande UFTPC

Comme on peut le voir sur la datasheet, il y a différentes utilisations de cette commande, nous allons donc, dans le cas d'utilisation du GPS, envoyer de manière régulière un fichier contenant l'identifiant du module et les coordonnées du joueur. Ce fichier sera ensuite récupéré et analysé par l'interface graphique afin d'actualiser la position du point sur la carte.

Donc pour effectuer cette action, il suffit d'utiliser les commandes suivantes :

```
AT+UFTPC=1
OK
+UUFTPCR: 1,1

***Connexion serveur FTP : OK***

On va dans uploads
AT+UFTPC=8,"upload"
OK
+UUFTPCR: 8,1
```

Figure 14 - Interaction avec le serveur via commandes AT

- **AT+UFTPC=1** → Connexion au serveur en utilisant les valeurs renseignées précédemment par la commande **+UFTP**

- **AT+UFTPC=8, "uploads"** → Déplacement dans le répertoire uploads du serveur où l'on a les droits de créer un fichier.
- **AT+UFTPC=5, "gps\_position", "gps\_position"** → Upload du fichier contenant les coordonnées actuelles du joueur

Ces commandes AT seront donc utilisées à la place du bloc de commande *http* présentés ci-dessus permettant de communiquer avec l'API d'Hologram.

Maintenant après avoir testé avec succès l'utilisation du FTP avec **m-center**, il reste à l'implémenter sur le programme de la STM32.

#### e. Implémentation sur la carte

Concernant l'implémentation des commandes AT sur le programme de la carte. Nous avons, avec Clément, défini une organisation plutôt simple.

De base, les commandes AT sont envoyés par UART au module 4G. Nous avons donc juste besoin d'écrire sur l'UART la commande que l'on souhaite envoyer. Pour cela, nous avons mis en place la fonction **sendAT** qui permet d'envoyer la commande AT voulue au module.

```
void sendAT(UART_HandleTypeDef* huart, AT_command at_command){
    char buff[100];
    int time_out = 5;
    int count_time_out = 0;
    statusAT = EN_COURS;

    if (at_command.temps_reponse > 999)
        time_out = 2;

    //vidage buffer
    HAL_UART_Receive(huart, (uint8_t*)buff,100,10);
    HAL_Delay(50);
    //-----

    HAL_UART_Receive_IT(huart, (uint8_t *)rxBuffer, 1);
    while (statusAT != OK && count_time_out < time_out){
        HAL_UART_Transmit(huart, (uint8_t*)at_command.command, sizeTabChar(at_command.command), 10);
        HAL_Delay(at_command.temps_reponse);
        count_time_out++;
        if(currentAT.type == AT_C_UDWNFILE)
            break;
    }
}
```

Figure 15 - Fonction sendAT

Afin de spécifier la commande AT à envoyer, nous avons créés le type **AT\_command** qui contient des informations sur la commande AT à envoyer dont on a besoin par la suite afin de vérifier la réussite de la commande.

```
typedef struct {
    int nombre_reponses;
    char * command;
    TypeATCommand type;
    uint32_t temps_reponse; // en ms
}
```

Figure 16 - Type AT\_command

On peut donc voir que l'on renseigne le nombre de réponses attendues, la commande en elle-même, le temps de réponse, et le type de celle-ci. Le type de commande permet de savoir de quelle façon vont être organisés les réponses à cette commande. Par exemple, certaines commandes attendent juste un OK, mais d'autres vont d'abord envoyer une information suivie d'un OK. Il y a même des commandes qui vont envoyer une réponse spécifique à cette commande. Il faudra donc traiter les réponses de ces différentes commandes de manière différente en fonction de leur type renseigné.

```
/*
Type :
RI -> Réponse à ignorer qui sera juste affichée
OE -> Réponse doit afficher OK, à tester pour mettre à jour le statusAT
C -> Commande particulière suivit du nom de la commande
*/
typedef enum {
    AT_C_UDWFILE,
    AT_RI,
    AT_RI_OE,
    AT_OE,
    AT_OE_RI,
    AT_C_CPIN,
    AT_C_UHTTPC,
    RI_AT_C_UHTTPC,
    AT_C_PING,
    AT_C_COPS,
    AT_C_UFTPC,
    AT_C_URDFILE
}TypeATCommand;
```

Figure 17 - Les différents types de commandes possibles

Concernant l'analyse des réponses, nous avons mis en place une interruption se déclenchant à chaque caractère reçu sur l'UART. Dans cette interruption, nous allons ensuite tester si c'est la fin de la réponse et si tel est le cas, il faudra appliquer les vérifications inhérentes au type de la commande permettant de savoir si la commande s'est bien exécutée ou si une erreur a été rencontrée.

```
else if((staking[index] == 'K' && staking[index-1] == 'O') || (staking[index] == 'R' && staking[index-1] == 'O'))
    flag_end_response = 1;
```

Figure 18 - Exemple de test de fin de réponse

Cette organisation peut paraître complexe mais elle nous permet de tester la réponse de chaque commande de manière précise, ce qui est indispensable pour le bon déroulement du programme.

Ensuite, nous avons aussi la variable globale **currentAT** qui représente la commande AT actuelle qui est en train d'être exécutée. Cela permet d'avoir accès aux informations de cette commande AT partout dans le programme. Ce qui est fortement utile, surtout dans l'interruption liée à la réception sur l'UART.

Sinon une fois tout cela mis en place, l'ajout de commande AT est plutôt simple, il suffit juste de la créer et de l'initialiser en renseignant les informations détaillées ci-dessus et on peut l'envoyer facilement via la fonction **sendAT**. On peut donc observer que de par notre organisation mise en place, l'envoi de commande AT s'en voit fortement simplifié.

```
else if (currentAT.type == AT_RI_OE) {
    if (tabsEquals(reponses[2], "OK\0"))
        statusAT = OK;
    else
        statusAT = FAILED;
}
```

Figure 19 - Exemple de test du résultat de l'exécution de la commande AT

Concernant donc, les différentes actions que l'on doit effectuer via commandes AT, on a juste créé différentes fonctions regroupant les différentes commandes permettant de faire une grosse action (initialisation module, connexion à Internet, envoi de coordonnées GPS).

Nous avons donc juste à regrouper les commandes AT présentés ci-dessus et à les envoyer depuis différentes fonctions.

```
void initLARA(UART_HandleTypeDef *huart) {
    int nbCommand = 3;
    AT_command initsCommands[nbCommand];
    int num_commande;
    int timeout = 20;
    int nb_init = 0;

    // Code Pin
    initsCommands[1] = init_AT_command(2, "AT+CPIN=\0264\0\r", AT_OE, 250);

    initsCommands[2] = init_AT_command(3, "AT+CPIN?\r", AT_C_CPIN, 250);

    // Mode full fonctionnality

    initsCommands[0] = init_AT_command(2, "AT+CFUN=1\r", AT_OE, 250);

    do {
        for (num_commande = 0; num_commande < nbCommand; num_commande++) {
            HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_12);
            currentAT = initsCommands[num_commande];
            sendAT(huart, currentAT);
        }
        HAL_Delay(100);
        nb_init++;
        if (statusAT == OK)
            HAL_UART_Transmit(&huart2, (uint8_t*) "****Init LARA : OK****\n\n", 23, 10);
        else
            HAL_UART_Transmit(&huart2, (uint8_t*) "****Init LARA : RETRY****\n", 25, 10);
    } while (statusAT != OK && nb_init < timeout);
}
```

Figure 20 - Fonction d'initialisation du module

```

StatusAT initConnection(UART_HandleTypeDef *huart){
    int nbCommand = 5;
    AT_command initsCommands[nbCommand];
    int timeout_HTTP = 3;
    int nb_init = 0;

    /* Config réseau */

    // Automatic network registration
    initsCommands[0] = init_AT_command(2, "AT+COPS=0\r", AT_C_COPS, 250);

    /* AT+UPSD=0,1,"hologram" */
    initsCommands[1] = init_AT_command(2, "AT+UPSD=0,1,\"s12sfr\"\r", AT_OE, 250);

    /* AT+UPSDA=0,1 */
    initsCommands[2] = init_AT_command(2, "AT+UPSDA=0,1\r", AT_OE, 250);

    /* AT+UPSD=0,100,1 */
    initsCommands[3] = init_AT_command(2, "AT+UPSD=0,100,1\r", AT_OE, 250);

    // On active le contexte PDP --> connection à l'internet
    initsCommands[4] = init_AT_command(3, "AT+UPSDA=0,3\r", AT_OE_RI, 1000);
}

```

Figure 21 - Fonction d'initialisation de la connexion Internet

```

void connexion_ftp(UART_HandleTypeDef* huart)
{
    uint8_t cpt_timeout = 0;
    int nbCommand = 6;
    AT_command initsCommands[nbCommand];

    /* Renseignement de l'adresse de l'hôte */
    initsCommands[0] = init_AT_command(2, "AT+UFTP=1,\"demo.wftpserver.com\"\r", AT_OE, 100);

    /* Renseignement du pseudo */
    initsCommands[1] = init_AT_command(2, "AT+UFTP=2,\"demo-user\"\r", AT_OE, 100);

    /* Renseignement du mot de passe */
    initsCommands[2] = init_AT_command(2, "AT+UFTP=3,\"demo-user\"\r", AT_OE, 100);

    /* Mode de connexion passif */
    initsCommands[3] = init_AT_command(2, "AT+UFTP=6,1\r", AT_OE, 100);

    /* Résolution DNS de l'hôte renseigné */
    initsCommands[4] = init_AT_command(3, "AT+UDNSRN=0,\"demo.wftpserver.com\"\r", AT_RI_OE, 2500);

    /* Connexion au serveur FTP */
    initsCommands[5] = init_AT_command(3, "AT+UFTPC=1\r", AT_C_UFTPC, 10000);
}

```

Figure 22 - Fonction de configuration de la connexion FTP

Par exemple, pour l'envoi des coordonnées GPS, on utilise les lignes de code suivantes qui permettent d'envoyer la commande nécessaire au module.

```

void postGPS_ftp(UART_HandleTypeDef* huart)
{
    /* Envoi du fichier des coordonnées sur le serveur FTP */
    currentAT = init_AT_command(3, "AT+UFTPC=5,\"gps_positions\", \"gps_positions\"\r", AT_C_UFTPC, 5000);
}

```

Figure 23 - Fonction d'envoi des coordonnées GPS via FTP



L'avantage de cette organisation est que si l'on a une erreur lors de l'exécution d'une commande, on a juste besoin de lancer la fonction associée. Par exemple, si lors de l'exécution d'une commande, on a une erreur de connexion, on aura juste besoin de lancer la fonction **initConnexion** qui devrait résoudre ce problème.

Voici donc un résumé de la manière dont on a mis en place l'implémentation des commandes AT sur le programme C.

Maintenant, je vais revenir avec plus précision sur les parties du code pertinentes. Tout d'abord, au niveau de la vérification que l'exécution de la commande AT se soit bien réalisée. En effet, en fonction du type de la commande concernée, cela peut s'avérer plus ou moins complexe. Pour le type de commande **+UFTPC** par exemple, on reçoit après avoir envoyé la commande, un *OK* ainsi qu'une réponse d'un type spécifique devant être analysé.

La commande de réponse se décompose de la manière suivante :

#### **+UUFTPCR:5,1**

- 4 : type de la commande uftpc, ici un c'est un *GET*
- 1 : booléen indiquant si l'exécution de la commande s'est bien déroulée. 1 pour une réussite et 0 pour un échec

On va donc d'abord vérifier si on a reçu les derniers caractères de réponse indiquant que l'on a reçu une réponse complète afin de la tester ensuite.

```
else if(currentAT.type == AT_C_UFTPC){
    if(staking[index-1] == ',' && (staking[index] == '1' || staking[index] == '0'))
        flag_end_responce = 1;
}
```

Figure 24 - Test de fin de réception de la réponse à une commande de type "AT\_C\_UFTPC"

C'est donc le dernier booléen de la réponse qui va nous intéresser, on va donc l'isoler dans notre test et actualiser l'état de l'exécution de la commande en fonction.

```
else if(currentAT.type == AT_C_UFTPC){
    if (reponses[2][12] == '1')
        statusAT = OK;
    else
        statusAT = FAILED;
}
```

Figure 25 - Test de succès de la commande

La démarche est la même pour toutes les commandes, on définit d'abord avec précision quelle réponse on attend dans le cas d'une réussite ou d'un échec. Ensuite, on implémente la vérification dans le callback et on actualise l'état de l'exécution de la commande actuelle en fonction.

### C. Tests effectués

Concernant la mise en place de la localisation GPS, notre principale échéance était le mardi 26 février où l'on avait une démonstration à effectuer devant l'entreprise Hunting Town dans leur environnement de déroulement du l'escape Game. Pour préparer cette démonstration, on a effectué différents tests afin de s'assurer que tout fonctionnait bien. Les tests consistaient à ce qu'une personne équipée du prototype de notre objet embarqué parcourait les rues environnantes de Bordeaux pendant que le reste de l'équipe observaient son parcours à travers l'interface graphique.

Après quelques problèmes rencontrés lors des premiers tests, les derniers tests furent couronnés de succès. Cependant lors de la démonstration avec le client, nous avons rencontrés un gros problème concernant la précision du GPS, en effet, nous nous trouvions dans des rues étroites entourées de hauts murs, cela a désorienté le module GPS au point qu'il envoie des valeurs de localisation incohérentes. Le point affiché sur la carte effectuait donc des mouvements improbables.

### D. Mesures suites aux tests

Suite aux imprécisions de localisation remarquées lors des tests, il a été décidé de changer la manière de récupérer la position GPS de l'objet, nous avons donc fait le choix d'abandonner le module GPS que nous utilisions afin de le remplacer par un module 3G comportant la localisation GPS. Cela semble un bon choix sur le long terme car le produit final sera donc plus optimisé car il ne contiendra qu'un module qui s'occupera de la communication 3G ainsi que de la localisation GPS.

Concernant la charge de travail supplémentaire que cela apporte, le module 3G choisi est du même constructeur que le module 4G que l'on utilisait précédemment. Cela implique que l'on communique avec le module exactement de la même façon via les mêmes commandes AT donc il y a juste la partie localisation à implémenter, la partie communication via réseaux cellulaires fonctionne toujours avec le même code.

Malheureusement, n'ayant pas le temps de concevoir la carte qui contiendrait le module, nous utilisons dans ce projet des cartes de développement permettant de tester directement les modules. Seulement, dans le cas du module 3G, toutes les fonctionnalités du module ne sont pas implémentées sur la carte de développement, il manque notamment le *GPS Receiver* qui est indispensable afin de mettre en place une localisation GPS précise.

Nous allons donc continuer avec le module 4G durant le projet mais l'idée d'utiliser le module 3G reste très intéressante et devrait être mis en place par la suite.

## 4. Lancement des vidéos

### A. Description

Cette partie est la deuxième plus importante et nécessite une sérieuse organisation afin de la mettre en place avec succès. L'objectif est de pouvoir jouer une vidéo au choix sur l'écran de la BeagleBone à partir d'une interface graphique se trouvant sur le PC du maître du jeu. Il faut savoir

que la BeagleBone possède la contrainte qu'elle peut se trouver n'importe où dans Bordeaux. La vidéo doit aussi se lancer dans un intervalle de temps assez court (moins de 5 secondes) à partir du moment où le maître du jeu appuie sur *Jouer vidéo*.

Comme pour les coordonnées GPS, nous utiliserons un serveur FTP comme interface de communication entre le module et l'interface graphique. La carte STM32 récupérera un ordre vidéo toutes les 3 secondes et après avoir vérifié qu'il est différent de l'ordre précédent, il sera envoyé via *Bluetooth Low Energy* à la BeagleBone qui réceptionnera ce message et jouera la vidéo associée.

Afin de mettre en place cette organisation, il y avait différentes parties dont il fallait s'occuper. De mon côté, je me suis chargé de la réalisation de l'interface graphique de lancement des vidéos ainsi que l'implémentation du code permettant la communication avec l'interface graphique (via FTP notamment). Il faut aussi savoir que le choix d'utiliser le FTP s'est fait au moment de commencer cette partie. Je me suis donc aussi occupé de toutes les études de mise en place du FTP dans notre projet actuel et de l'élaboration de toutes les solutions tournant autour.

## B. Réalisation de l'interface graphique

L'objectif de cette interface graphique est de permettre au maître du jeu, de choisir de jouer la vidéo voulue sur l'écran connecté au BeagleBone des joueurs. L'interface doit être simplifiée au maximum afin que ce le choix de la vidéo soit le plus efficace possible. Après une rapide réflexion sur les façons de mettre en place cette User Interface, nous avons choisi d'utiliser le *Python* et son module *Tkinter*.



Figure 27 - Logo python

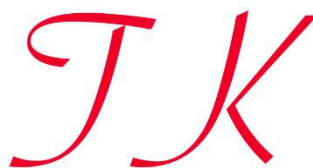


Figure 26 - Logo TKinter

Ce choix semble être le plus efficace pour nous de par la syntaxe du python permettant de simplifier des tâches qui peuvent sembler complexe de premier aspect. L'utilisation du Python permet aussi de faciliter l'utilisation du FTP à travers l'interface graphique car cela sera nécessaire afin de communiquer avec le module l'ordre de la vidéo.



On a donc mis en place l'interface suivante qui a l'avantage d'être concise et d'afficher clairement les informations essentielles à l'utilisateur :

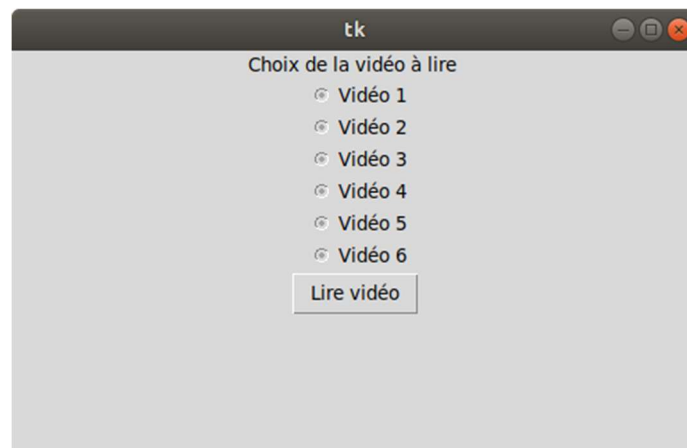


Figure 28 - Interface de choix des vidéos

Le code de l'interface graphique est assez simple et a été rapide à mettre en place. Notre programme contient la classe principale *Interface* qui hérite de *Frame*, un package du module Tkinter et qui représente la fenêtre. Les différents sont les éléments qui compose l'interface, c'est-à-dire plusieurs boutons radios, un bouton d'envoi ainsi qu'un Label contenant le titre de l'interface. Ensuite, cette classe contient seulement deux méthodes, l'initialisation et l'envoi des vidéos qui se fait comme cela :

```
def envoi_video(self):
    tkinter.messagebox.showinfo("Info", "Vidéo lancée !")

    with open("envoi_videos", 'w', encoding="utf-8") as fic_ordre:
        fic_ordre.write("{}".format(self.choix.get()))

    with open("envoi_videos", 'rb') as fic_ordre:
        ftp.storbinary("STOR ordre_video", fic_ordre)

    print("Envoi de la vidéo {}".format(self.choix.get()))
```

Figure 29 - Fonction d'envoi de l'ordre vidéo via FTP

Dans cette méthode, on crée tout d'abord un fichier *envoi\_videos* et on écrit dans ce fichier l'identifiant de la vidéo cochée. Ensuite, on envoie ce fichier via FTP sur le serveur auquel on s'est connecté précédemment. Cette fonction est appelée à chaque appui de la part de l'utilisateur sur le bouton *Lire*.

Concernant la connexion au serveur FTP, elle se fait de la manière suivante :

```
# FTP
ftp = FTP("demo.wftpserver.com")
ftp.login(user="demo-user", passwd="demo-user")
ftp.cwd("upload")
```

Figure 30 - Fonction de connexion au serveur FTP

On crée d'abord une instance *FTP* en renseignant l'hôte FTP. Ensuite, on se connecte au serveur en précisant le pseudo et le mot de passe associé. Après cela, nous sommes connectés et nous nous déplaçons au répertoire *upload* qui nous permet d'avoir les droits d'écriture et de lecture sur les fichiers.

Cette interface graphique s'occupe aussi de récupérer le fichier de coordonnées GPS posté par le module sur le même serveur. On utilise donc la même instance FTP que dans le reste du programme et l'on va récupérer le fichier *coor\_gps* sur le serveur souhaité et on va le sauvegarder sur le PC dans le dossier actuel afin de le mettre à disposition pour le programme affichant la position sur une carte. Cette fonction se déclenche toutes les sept secondes grâce à la fonction *after* inclus dans la classe *Frame* qui permet de déclencher un callback après un intervalle de temps précisé à l'avance.

```
def get_coor_gps():  
    ftp.retrbinary("RETR coor_gps", open("coor_gps", "wb").write)  
    fenetre_principale.after(7000, get_coor_gps)
```

Figure 31 - Fonction de récupération des coordonnées GPS

### C. Configuration du module

Concernant la configuration du module, presque tout le travail est déjà effectué après la partie *Localisation GPS*. En effet, la fonction de connexion au serveur FTP est déjà créée et l'étude sur les commandes AT concernant le FTP est déjà faite. Ici l'objectif est de récupérer à intervalle régulier (environ toute les 3 trois secondes) le dernier ordre envoyé sur le serveur FTP et de comparer cet ordre avec le dernier que l'on a reçu. Et en cas de différence, on doit envoyer cet ordre via Bluetooth Low Energy à la BeagleBone.

Afin de faire un *get* sur un serveur FTP, on utilise la commande **AT+UFTPC** car comme je l'ai dit précédemment, cette commande s'occupe de toutes les actions FTP possibles avec le module. Ici, on met un 4 en premier argument permettant d'effectuer un *get* sur le serveur. Ensuite, il n'y a qu'à préciser le nom du fichier que l'on veut récupérer et le nom qu'on veut lui donner quand il sera téléchargé.

Dans notre cas, nous aurons juste besoin d'utiliser la commande suivante :

```
AT+UFTPC=4, "coor_gps", "coor_gps"  
AT+UFTPC=4, "coor_gps", "coor_gps"  
OK
```

Figure 32 - Commande AT permettant de récupérer un fichier sur un serveur FTP

Nous n'avons rien de plus à faire concernant la configuration du module. Il ne reste plus qu'à implémenter cette fonctionnalité sur le code C de la carte STM32.

## D. Implémentation

L'objectif est de déclencher la commande de *GET* environ toutes les trois secondes. Pour cela, il va falloir ajouter un *Timer* à notre programme actuel. Ce *Timer* sera donc configuré avec un *Prescaler* de 750 et un *autoreload* de 64000. Nous pouvons vérifier via la formule suivante que cela nous permet d'obtenir un intervalle de déclenchement de trois secondes :

$$T = T_{horloge} \times prescaler \times (autoreload + 1)$$

$$T = \frac{1}{16000000} \times 750 \times 64001$$

$$T = 3,00004 \text{ s}$$

Ce *Timer* fut donc configuré comme cela :

```
htim2.Instance = TIM2;
htim2.Init.Prescaler = 750;
htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
htim2.Init.Period = 64000;
htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
```

Figure 33 - Configuration Timer

Ensuite dans l'interruption déclenchée par ce *Timer*, on modifie juste la valeur d'un booléen indiquant que les trois secondes sont écoulées et que l'on peut récupérer le dernier ordre de vidéo présent sur le serveur FTP.

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    /* Prevent unused argument(s) compilation warning */
    UNUSED(htim);

    bool_getVideo = 1;
}
```

Figure 34 - Callback de déclenchement du timer

Et donc ensuite dans la boucle infinie du *main*, on va appeler la fonction *getVideo\_ftp* si la valeur du booléen a bien été modifié.

```
while (1)
{
    HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_14);

    if(bool_getVideo){
        getVideo_ftp(&shuart3);
        HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_15);
        bool_getVideo = 0;
    }
}
```

Figure 35 - appel à la fonction *getVideo\_ftp*

La fonction *getVideo\_ftp* contient l'envoi de la commande AT détaillée précédemment permettant de récupérer le fichier souhaité sur le serveur FTP. On donc procéder de la même manière que pour l'envoi des commandes précédentes. On va d'abord initialiser *currentAT* avec notre commande AT de *GET*. Et ensuite on fait appel à *sendAT* qui gère toute la partie envoi de la commande. Concernant la gestion de la réception, elle se fait bien sûr au niveau du *callback* de réception UART où l'on va vérifier bien sûr que l'exécution de la commande s'est bien passée.

On peut observer qu'à la fin de la fonction, on envoie la commande AT **+URDFILE="ordre\_video"** au module. Cette commande permet de lire le contenu du fichier *ordre\_video*, fichier que l'on vient récupérer, et d'afficher son contenu. Ensuite dans la partie du code gérant la réception des réponses aux commandes, on va récupérer la partie du fichier qui nous intéresse qui est l'identifiant de la vidéo à lire et on va vérifier si cette valeur est différente de la précédente reçue afin de l'envoyer via Bluetooth Low Energy.

```
else if(currentAT.type == AT_C_URDFILE){
    if(strncmp(reponses[2], "OK")){
        statusAT = OK;
        static uint8_t old_value = 0;
        uint8_t value = reponses[1][28] - 48;
        if(value != old_value){
            valueBluetooth = value;
            old_value = value;
        }
    }
    else
        statusAT = FAILED;
}
```

Figure 36 - Vérification de l'exécution de la commande +URDFILE

Concernant l'envoi via Bluetooth de l'identifiant de la vidéo à lire, je ne me suis pas du tout occupé de la partie Bluetooth du code. Tout ce que j'ai à faire est de modifier la variable *valueBluetooth* qui contient les données à envoyer et ensuite dans la boucle infinie du main, il n'y a qu'à vérifier si la valeur de la variable *valueBluetooth* a changé et dans ce cas, envoyer la valeur via la fonction *nrf\_manage\_tx*.

```
if(valueBluetooth != NRF_DATA_DEFAULT){  
    nrf_manage_tx(valueBluetooth);  
    valueBluetooth = NRF_DATA_DEFAULT;  
    HAL_UART_Transmit(&huart2, (uint8_t*)"on vient d'envoyer\n", 19, 10);  
    HAL_TIM_Base_Start_IT(&htim2);  
}
```

Figure 37 - Test de la valeur de la variable *valueBluetooth*

## E. Tests

Cette partie a été testée plusieurs fois, elle a d'abord été confrontée à plusieurs bugs au niveau de la réception des réponses aux commandes AT de type **+UFTPC** mais cela fut par la suite corrigé ainsi que d'autres plus petits problèmes apparus au fil des tests. Les tests effectués dans l'enceinte de l'école sont maintenant entièrement favorables et l'on peut dire que cette partie du projet fonctionnelle. En effet, toute la chaîne de communication de l'interface graphique côté maître du jeu jusqu'à l'écran du BeagleBone est parfaitement établie et fonctionne sans faille.

On a pu tester cette partie avec Thibault et Nathalie le mardi 28 juin lors d'une démonstration dans l'entreprise Hunting Town. Cette démonstration s'est globalement bien passée. En effet, la chaîne de lancement des vidéos a parfaitement fonctionné. Le client était satisfait du fonctionnement global du système et des différentes contraintes présentes (temps d'initialisation, délai entre l'envoi de l'ordre et la vidéo qui se joue, robustesse).

# 5. Récupération du son en direct

## A. Objectif

Cette partie est moins importante que les deux précédentes qui viennent d'être présentées. Cela concerne un désir de notre client de pouvoir avoir accès au son ambiant des joueurs afin de connaître leur situation en temps réel afin de savoir s'ils ont besoin d'aide ou pas. L'objectif est donc de trouver un moyen de récupérer le son ambiant des joueurs et de l'envoyer sur un appareil accessible au maître du jeu. Cette partie, concernant essentiellement l'utilisation du module 4G, m'est entièrement attribuée.

## B. Solution envisagée

Après réflexion et, la solution envisagée au début qui semblait être la plus efficace serait un appel téléphonique effectué entre le module 4G et le téléphone du maître du jeu. Cet appel serait un appel *VOLTE* (Voice Over LTE). Cela se base sur le *Voice Over IP*, une technique qui permet de transmettre sa voix sur des réseaux IP, en l'occurrence Internet dans notre cas. Cette solution est supportée par le module 4G que nous utilisons et nous permet d'utiliser les antennes 4G afin de passer un appel. Le *Voice Over IP* a pour principaux défauts une éventuelle latence causée par le temps de parcours des paquets ainsi qu'une possible perte de paquets. Mais ces défauts ne sont pas un problème pour notre situation car la transmission du son des joueurs en direct sert juste à ce que le maître du jeu soit au courant de la situation des joueurs pour les aider si nécessaire. Ce n'est donc pas important s'il y a un petit délai ou qu'il manque quelques paquets.



Figure 39 - Logo VOIP



Figure 38 - Logo VoLTE

L'appel sera effectué entre le téléphone du maître du jeu qui sera sur haut-parleur et le module 4G. Cet appel ne servira qu'à la transmission du son en direct et ne permettra pas de communiquer entre le maître du jeu et les joueurs pour par exemple répondre aux énigmes ou pour poser des questions. Ce sera donc une communication unidirectionnelle depuis les joueurs vers le maître du jeu. Concernant la procédure de l'appel, ce sera sûrement le maître du jeu qui effectuera l'appel lors du départ des joueurs. Le module sera alors configuré pour répondre automatiquement à un appel. Le module devra donc aussi être équipé d'un microphone d'une assez bonne qualité qui récupérera le son des joueurs mais ne nécessite pas de haut-parleurs.

### C. Configuration du module

#### Gestion des appels

Afin d'effectuer un appel via VOLTE, il y a différentes commandes à appliquer au module. La première opération importante est de configurer l'*Ip Multimedia Subsystem* de manière appropriée. L'*IMS* est une architecture standardisée utilisée par les opérateurs téléphoniques. Cette architecture à l'avantage de permettre l'utilisation de la technologie *VoIP* ainsi que les appels téléphoniques traditionnels.

L'activation de l'*IMS* se fait via la commande **+UIMSCFG=0,1,50,1**.

N'ayant pas encore décidé qui était l'initiateur de l'appel, nous avons décidé d'implémenter les deux situations. Pour que le module passe un appel, une fois qu'il est connecté à Internet et que l'*IMS* est activé, il suffit d'utiliser la commande **ATD<num>** ;. Cette commande est très simple à utiliser et est

facilement implémentable. *num* représente le numéro à appeler suite à cette commande. Pour le format du numéro, il est réglable via la commande suivante :

- **+CSTA** → Type de numéro à écrire : - **145** : international  
- **129** : national

Pour gérer la situation inverse où c'est le maître du jeu qui appellerait le module, il faut mettre en place un mécanisme de réponse automatique. Cela se fait via la commande **ATS0=<value>** où *value* est le nombre de sonnerie attendu avant de répondre. Cela est encore une fois très simple, il suffit d'exécuter cette commande lors de l'initialisation du module afin de configurer la gestion du module des appels entrants.

Comme on a pu le voir, les deux solutions sont très simples à configurer et nous sommes maintenant prêt à utiliser les deux.

## Gestion du micro

Comme je l'explique par la suite, cette partie a posé quelques problèmes du fait que l'on ne pouvait pas tester réglages effectués. Mais cela n'empêche pas de faire une étude sur les commandes AT permettant de configurer le microphone de manière optimal.

L'interface audio générale des modules u-blox est représentée de la manière suivante :

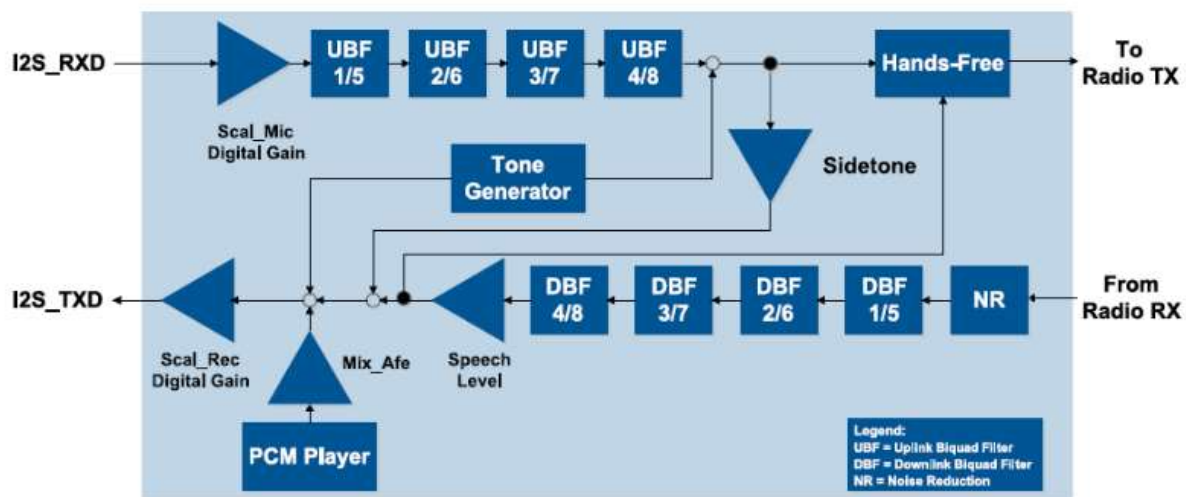


Figure 40 - Interface audio des modules u-blox

Mais sur le module sara U-201 qui possède une prise jack contrairement au module 4G, il ne possède pas d'entrée analogique, seulement des inputs numériques communiquant en *\*I2S\**.

- **AT+UPSM=1,2,0,0** → Permet de préciser l'*\*uplink\** et le *\*downlink\** qui sont numérique dans notre cas



- **AT+UI2S=1,1,0,3,1** → Configuration de l'interface I2S
  - **1** : I2S configuré en PCM (voir tableau qui suit pour plus de précision)

### 22.2.5 PCM modes (short synchronization signal)

Mode	CLK EDGE for TX	CLK EDGE for RX	WA pulse length	CLK frequency	WA frequency
0	RISING	FALLING	2 clks	18*<I2S_sample_rate>	<I2S_sample_rate>

Figure 41 - Datasheet sur la configuration du mode de l'I2S

- **1** : Identifiant des ports I2S physiques à utiliser sur le module.
- **0** : Horloge dynamique
- **3** : Taux d'échantillonnage fixé à 16 kHz
- **1** : Module configuré en Slave
- **AT+UMGC=1,,16000** → Configuration du gain appliqué au signal émanant du microphone
- **AT+UUBF= ?** → Commande permettant la gestion des filtres au travers du module sur le signal reçu. Le sujet étant compliqué et demandant beaucoup de temps afin de le mettre en place, nous avons décidé de ne pas nous attarder sur cet aspect car cela concernait une demande secondaire du client et que nous avons besoin de pouvoir tester nos différentes tentatives, chose impossible avec la carte d'évaluation du module.

## D. Problèmes rencontrés

Dans cette partie, nous avons rencontré plusieurs problèmes assez importants nous demandant une remise en question de nos solutions. En premier lieu, comme je l'ai expliqué précédemment, c'est ici que l'on s'est rendu compte que la carte SIM *Hologram* ne fournissait pas de numéro de téléphone et n'était donc pas compatible à l'utilisation d'un appel téléphonique.

Une autre déconvenue liée à cette partie est la vérification des commandes liées aux appels. En effet, ces commandes répondent toujours automatiquement par un *OK* mais cela ne nous intéresse pas. La réponse qui nous intéresse est une commande de réponse arrivant selon un délai aléatoire et contenant toutes les informations qui nous intéressent. Néanmoins, notre façon d'avoir géré les commandes AT dans notre programme n'est pas adapté à la réception d'un message selon un délai aléatoire et pouvant arriver à tout moment. Cela implique que la vérification que l'on fait suite à l'utilisation de ces commandes n'est pas optimale.

Nous nous sommes aussi rendu compte que la carte de test du module 4G n'avait pas d'entrée microphone, nous empêchant tester complètement les appels.

Ayant prévu de nombreux réglages du microphone afin d'appliquer des filtres permettant de supprimer le bruit environnant et de ne récupérer que les voix des joueurs, le fait de ne pas pouvoir tester les différentes solutions est un réel problème et de ce fait, nous avons donc décidé de ne pas s'attarder sur cet aspect, étant une demande optionnelle du client.

Nous avons donc décidé d'implémenter les commandes permettant de passer un appel et de répondre automatiquement ainsi qu'une configuration standard du micro censé fonctionner même si le fait de ne pas pouvoir la tester est un réel problème.



## Conclusion

A la fin de cette période de projet, le résultat obtenu est plutôt satisfaisant. Nous avons mis en place un système de localisation GPS associé à une interface graphique où l'on peut observer la position du joueur sur une carte de Bordeaux. Nous avons aussi un système permettant de lancer une vidéo sur l'écran des joueurs à partir d'une interface graphique. Cet aspect-là est parfaitement fonctionnel et a l'avantage de pouvoir fonctionner à n'importe quelle distance entre le maître du jeu et l'interface graphique.

Il y a aussi la partie de la récupération du son ambiant des joueurs qui est implémenté mais qui n'est pas encore fonctionnelle du fait que l'utilisation d'un microphone n'est pas possible sur les cartes d'évaluation du module 4G.

La réalisation de ce projet a connu plusieurs péripéties. En effet, n'ayant pas encore beaucoup d'expérience dans les différents domaines concernés par le projet, certaines décisions prises lors du début du projet se sont avérées ne pas être les meilleurs. De ce fait, nous avons dû réagir rapidement en trouvant des solutions alternatives. Cela a entraîné du travail effectué non utilisé dans la solution finale comme par exemple le travail fourni sur la communication avec l'API d'Hologram. Nous avons donc perdu un certain temps avec ces changements de technologies à utiliser mais il était important que notre solution finale puisse palier à tous les objectifs énoncés au début du projet.

Concernant mon travail plus spécifiquement lors de ce projet, en m'occupant entièrement de la partie communication avec le module, j'ai appris et maîtrisé le fonctionnement des commandes AT. J'ai aussi pu mettre en place une sorte d'API permettant d'envoyer les commandes AT au module 4G depuis la carte STM. J'ai aussi dû réfléchir rapidement à des solutions alternatives pouvant les problèmes de nos solutions initiales. Notamment, sur le remplacement de l'API d'Hologram par un serveur FTP.