AÉRONAUTIQUE
& SYSTÈMES EMBARQUÉS

Hunting Town

SYSTEME EMBARQUE DE COMMUNICATION

Lavergne Clément | Escape Game| 2018-1019

Sommaire

Présentation du projet

CADRE DU PROJET

Dans le cadre d'un projet qui se déroule entre le 20 novembre 2018 date de choix des projets et le 22 mai 2019 où il faut rendre les rapports nous avons eu à mettre à l'épreuve nos compétences techniques. Parmi les 5 projets proposés par l'école mon choix s'est porté sur le projet « Hunting Town ». Ce projet est en lien avec l'entreprise du même nom. Trois jours par semaine (lundi, mardi, mercredi) sont mis à notre disposition pour travailler sur le projet en plus d'un budget conséquent bien que non défini. Pour ce projet le client ne paye rien à l'école, nous n'avons donc pas une obligation de résultat vis-à-vis de ce dernier.

L'hypothèse du lancement d'une start-up à été émise lors de la rencontre avec le client. En effet si le projet est mené à terme le produit fini sera composé d'au moins une carte électronique conçue par nos soins. L'objectif de la start-up pourrait être la production desdites cartes et le téléchargement des logiciels (code et image d'OS) les cartes. Ce projet reste assez loin dans le temps et dépend de nombreux facteurs, il s'agit tout de même de source importante de motivation.

Membres du groupe : Antoine Venier, Axel Devaux, Nicolas Gibaud, Jean-Gabriel Massicot et moi même

Notre Groupe à l'origine composé de cinq membres avec Jean-Gabriel Massicot dans le rôle de « Scum Master » et d'intermédiaire avec le client. Notre groupe s'est retrouver réduit à quatre dès janvier 2019 lorsque Jean-Gabriel est parti en stage, j'ai donc repris la fonction d'intermédiaire avec le client. Avec ce départ nous perdons la volonté de fonctionner en agile, mouvement qui été porté par Jean-Gabriel mais que personne de pense pouvoir reprendre à cause du temps à consacrer à l'obtention des connaissances nécessaires pour ce poste.

L'ENTREPRISE

L'entreprise Hunting Town propose deux types d'activités sur leur site internet :

- Une chasse au trésor en ville (la rochelle et bordeaux) ouvert à tous dont l'objectif est de suivre une série d'indice pour être le premier à trouver un code à envoyer à un numéro qu'il faut également déduire. Ces défis sont éphémères, ils se terminent dès que quelqu'un a réussi.

- La seconde activité sur laquelle nous allons nous pencher plus en détail est défini comme un « Escape Game » en plein air, dans la ville. Les participants, doivent successivement suivre des indices les amenant visiter de nombreux point d'intérêt du centre-ville de Bordeaux.

DETAILS CONCERNANT LE JEU

L'objectif est réussi à résoudre les énigmes en moins d'une heure et le plus rapidement possible pour être classé par rapport aux autres participants. Les joueurs (en équipe de 2 à 6 personne) se rendent sur un lieu prévu à l'avance, équipés d'une carte de la ville. Nous nommerons chaque lieu un checkpoint. La partie est constituée d'une série de checkpoint qui doivent être résolus successivement.

- Résolution d'un checkpoint : Lorsque les joueurs se trouvent à l'emplacement du checkpoint une vidéo leur est joué. Cette vidéo est une énigme qui sera résolut en observant les alentours pour déduire un mot de passe. Les joueurs communiquent le mot de passe qu'ils ont trouvé au maître du jeu, si leur réponse est correcte, une nouvelle vidéo est jouée indiquant l'emplacement du prochain checkpoint.

L'ambiance lors de la séance est très importante il faut réussir à maintenir un rythme soutenu et assurer la continuité de l'histoire. Dans ce cadre tout problème technique doit être absolument évité.

Le Système actuel

LE MATERIEL

Pour effectuer les actions présentées plus tôt l'entreprise a mis en place un système assez complexe nécessitant de nombreux appareils :

- Un joueur porte sur lui un harnais accueillant le caméra type GoPro, sur lequel est également fixé un téléphone relié à des écouteurs servant de micro.
- Un second joueur est équipé d'un téléphone qu'il porte au bras qui affichera les vidéos
- Un troisième joueur doit utiliser son téléphone pour transmettre les mots de passes déduits.
- Un dernier joueur porte une montre bracelet indiquant le temps restant avant la fin de la partie.

DESCRIPTION

Le téléphone accroché sur le harnais de la caméra permet d'écouter le son autour du joueur qui le porte à l'aide des écouteurs qui y sont branchés et accroché au harnais. L'objectif est d'entendre les communications entre les joueurs pour les aiguiller s'ils partent sur une mauvaise piste. La communication est obtenue en passant un appel téléphonique en continu.

Le téléphone accroché au bras du second joueur est contrôlé à distance à l'aide de l'application « Team Viewer », en prenant le contrôle de la sorte le maître du jeu peu depuis son PC lancer les vidéos à sa guise. Ce téléphone possède également une seconde utilité : l'application « Messenger » tourne en tâche de fond, elle est utilisée pour la géolocalisation intégrée à l'application pour savoir où sont les joueurs et ainsi lancer les vidéos au bon moment.

LIMITES

Le problème majeur du système actuel est dû à l'application « Team Viewer », en effet le réseau 4G est logiquement assez variable lorsque les joueurs se déplacent dans la ville. Bien que la couverture réseau soit très bonne il est possible qu'une déconnection rapide intervienne, lorsque cela se produit l'application ferme la connexion et il n'est plus possible de contrôler les vidéos et l'expérience de jeu s'en voit détériorée.

Le second problème important est le nombre de téléphone portable important nécessaire. En effet, il est prévu d'augmenter le nombre d'équipe utilisant le système (aujourd'hui une seule équipe peut jouer à la fois) mais ceci nécessiterait l'achat de plusieurs téléphones portables et forfaits qui vont avec. Les coûts serait assez important. De plus, plus le nombre d'appareil par kit augment, plus la chance qu'une panne survienne augmente également.

Notre solution

FONCTIONNALITES

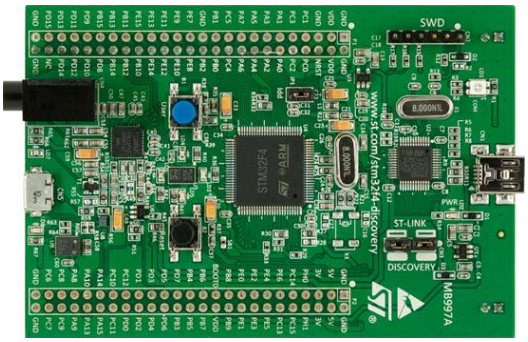
Afin de savoir comment construire notre système il faut dans un premier temps lister les fonctionnalités dont le client a besoin :

- Communication 4G
- Traçage GPS
- Affichage des vidéos (Vidéo et audio)
- Transmission des voix des joueurs

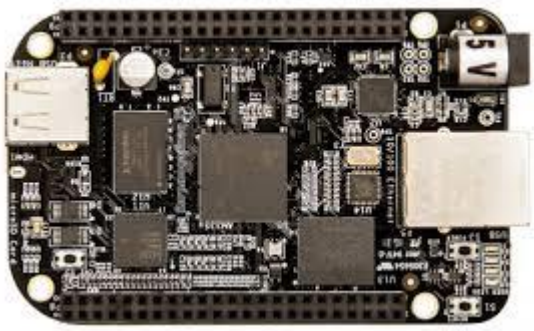
C'est à partir de ses besoins simplifiés que nous allons mettre en place un prototype fonctionnel.

MATERIEL UTILISE

Pour la réalisation de notre prototype nous avons choisis d'utiliser un microcontrôleur STM32, nous possédons à l'école plusieurs cartes de prototypage sur lesquelles nous pouvons développer bon nombre d'applications. Elle est équipée d'un microcontrôleur puissant sur lequel nous avons l'habitude travailler.



Néanmoins la puissance d'un microcontrôleur n'est pas suffisante pour l'affichage d'une vidéo ou pour la gestion d'un écran avec une qualité importante, c'est pour cela que nous souhaitons que la partie affichage vidéo soit gérée par un BeableBoneBlack qui est équiper d'un micro-processeur qui saura supporter notre utilisation.



Il aurait été préférable de concevoir un système en un bloc qui se serait fix é sur le bras du joueur et c'est en théorie possible : un téléphone portable possède les fonctionnalités que nous recherchons mais nous n'avons pas les compétences nécessaires pour fabriquer des composants aussi compacts que ceux d'un téléphone.

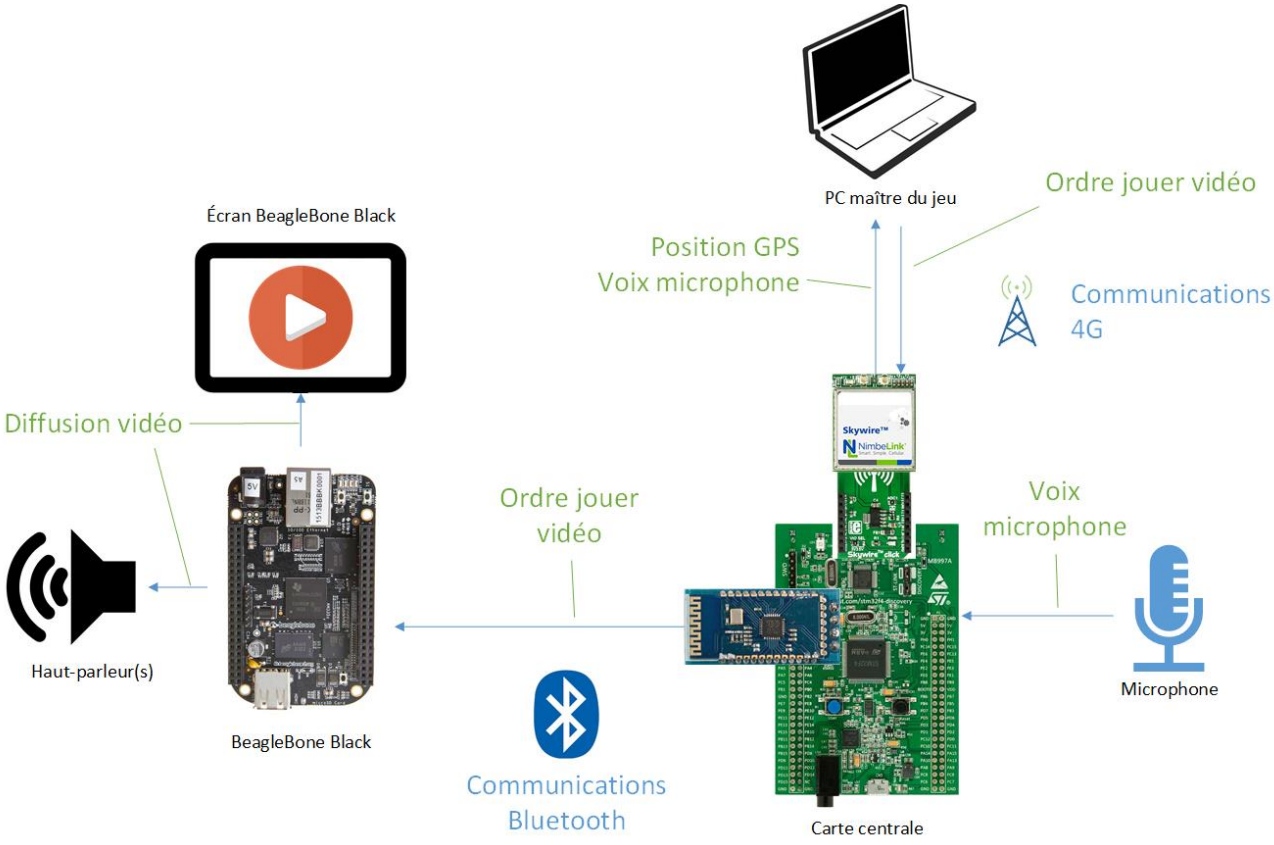
Nous avons tout de même souhaité garder l'affichage de la vidéo au niveau du poignet du joueur car à l'usage, nous savons que cette configuration est pratique. C'est ainsi que nous avons décidé que notre système serait en deux parties distinctes :

1. La première sera fixée sur le bras du joueur et sera composée d'un BeagleBoneBlack et d'un écran.
2. La seconde sera fixée sur le joueur, les possibilités de fixation sont importantes car les deux briques communiqueront sans fils. Elle gèrera les autres besoins.

La partie matérielle nous rajoute un besoin à mettre en place :

- Communication entre Carte STM32 et le BeagleBoneBlack

COMMUNICATIONS PREVUES



Cette représentation de notre système permet de voir toutes les communications entre les différentes parties. Nous avons choisi une communication Bluetooth entre deux cartes car les derniers BeagleBoneBlack possède un module Bluetooth intégré, de plus, ce type de communication est particulièrement adapté à de courte distance comme dans notre cas.

Un premier objectif

Quelques semaines après le début du projet, lors de l'un des entretiens téléphoniques qui ont lieu toutes les deux semaines. Le client nous a demandé s'il était possible de fournir une partie du projet en priorité. Parce qu'il souhaite pouvoir faire participer plusieurs équipes en même temps il doit renouveler sa collection d'appareil. Il souhaite donc que l'on développe un module de tracking GPS.

Concevoir un produit sous forme de plusieurs briques fonctionnelles est une excellente idée car cela permet de de suivre un cap proche dans le temps, cap plus facile à atteindre qu'un projet global fini. Il s'agit donc de notre objectif intermédiaire.

Développement

GESTION COMMANDE AT

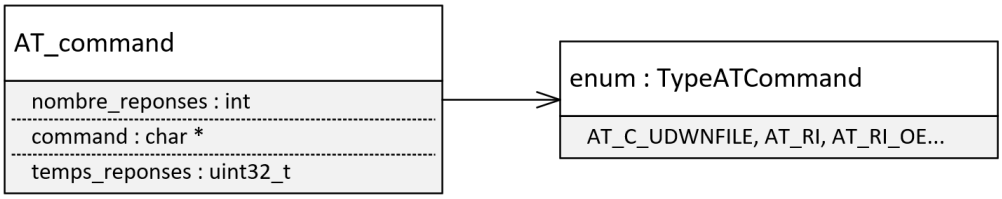
Nous avons pris la décision d'utiliser un module LARA-R211 pour connecter notre carte à internet par 4G. La communication avec ses modules s'effectue par UART à l'aide d'un protocole de commande AT. Il s'agit de commande que l'on va retrouver dans une documentation commune à beaucoup de module. Je n'ai pas personnellement beaucoup travaillé sur le choix des commandes à envoyer pour faire fonctionner notre projet. J'ai cependant eu la charge d'implémenter la gestion de ces commandes sur notre microcontrôleur.

Figure 1 Module LARA-R211

Dans un premier temps nous avons définie les informations utiles pour la gestion d'une commande AT :

- Le texte de la commande en elle-même qui est de taille variable, deux commandes de se ressemblant jamais dans la forme
- Le nombre de réponses : Après l'envoi de chaque commande le module va nous répondre, les réponses en question et leur nombre sont très variable, dans tous les cas le module va envoyer un écho de la commande envoyer. La suite dépend des commandes, parfois la transmission s'arrête ici, d'autre fois il faudra attendre des réponses et les traiter.
- Temps_reponse : certaine commande (souvent liées à l'http) lance des processus assez long. En temps normal lorsque l'on de reçoit pas de réponse d'une commande ceci est traité comme une erreur. Pour les commandes où l'on indiquera un temps de réponses important nous attendrons davantage avant de prendre une décision.
- Type de commande : Lorsque l'on s'intéresse aux commandes à effectuer et à la manière de traiter leur réponse certain pattern semble de dégager : par exemple bon nombre de commande suivent de schéma : écho, information à ne pas traiter, OK ou ERROR. Nous avons donc mis en place une énumération permettant d'étiqueter les commandes à leur création pour savoir comment des traiter.

Une fois les informations utiles définie nous les avons regroupées au sein d'une structure de donnée.



TypeATCommand

Ce type que nous avons fabriquer tient une place capitale dans le développent de notre solution, bien que parfois certaines commandes soit gérer de manière individuelle car elles sont trop spécifiques, d'autre en revanche sont regroupées car gérée de la même manière. Nous avons mis en place un nommage parlant pour nous y retrouver :

- Un « C » signifie qu'il s'agit d'une commande particulière dont le nom est indiqué à la suite
- Un « RI » signifie qu'une réponse est présente mais qu'il n'y a pas d'action à effectuer dessus.
- Enfin, un « OE » signifie que le module nous renvoie si la commande à réussie ou non, deux réponses sont donc possibles : OK ou ERROR. Cette réponse est lue pour savoir si l'on a réussi l'action effectuée.

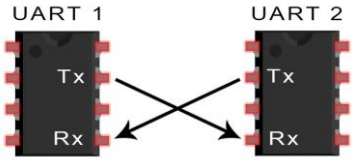
Exemple de nom de type de commande :

- AT_C_UDWNFILE : Commande particulière nommée UDWNFILE
- AT_RI : Une seule réponse à ignorer (en plus de l'écho)
- AT_RI_OE : Première réponses à ignorer, puis une réponse de type OK ou ERROR à traiter

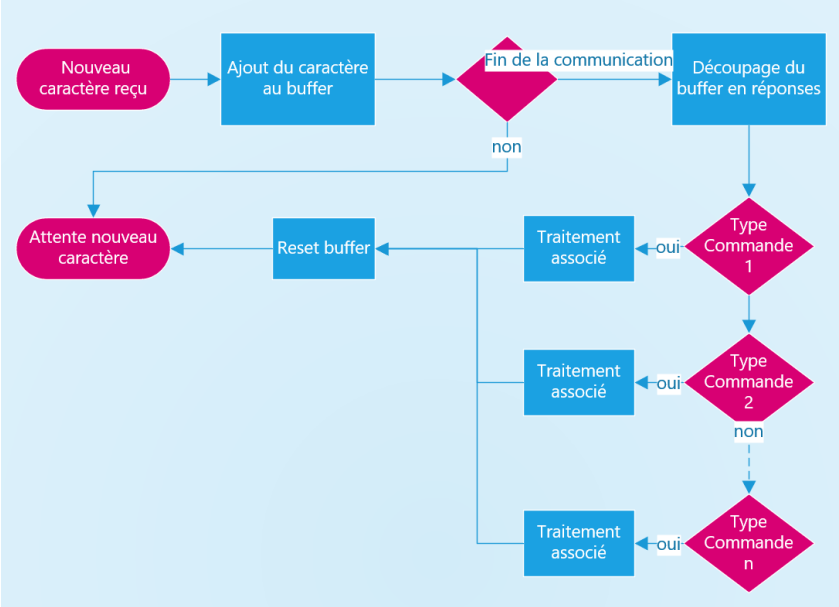
Pour chaque commande reçue si la réponse indique une erreur la commande est renvoyée, cette opération est effectuée cinq fois les commandes classiques, ce nombre tombe à deux pour les commandes avec un long temps d'attente de la réponse afin de ne pas perdre trop de temps.

RECEPTION UART

La réception en UART des commandes AT en provenance du module 4G s'est avérer légèrement plus complexe que prévu. En effet, pour la réception par interruption que nous avons voulue mettre en place il faut connaître à l'avance la taille de ce que l'on va recevoir. C'est impossible pour nous car certaines réponses ne sont absolument pas prévisibles du point de vue de leur taille.



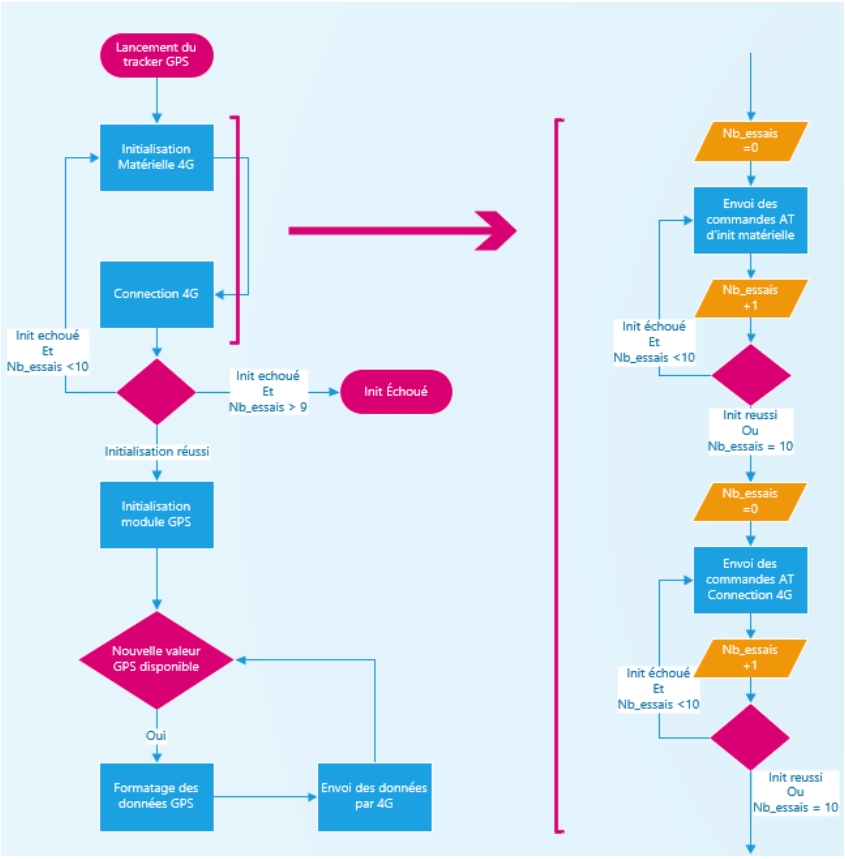
La solution trouvée consiste à récupérer les caractères un par un et les stockées dans un grand buffer, on continu cette opération (peu couteuses en temps) jusqu'à ce que l'on détecte la fin des réponses des modules, cette détection de fin de communication dépend des commandes mais n'est pas trop complexe. Une fois toute la communication enregistrée dans le buffer on peut passer au traitement.



Pour le traitement nous utilisons une variable globale que nous avons nommée statusAT, elle peut prendre trois valeurs : EN_COURS, OK ou FAILED. La variable est mise à l'état EN_COURS lors du lancement de la routine d'envoi de commande AT, le traitement associer à chaque commande a pour tâche principale de passer la variable statusAT à la valeur OK ou FAILED en fonction de ce que l'on reçoit.

INITIALISATION

L'initialisation du module 4G et de la connexion à l'API est une partie cruciale, en effet, si cette dernière ne s'effectue pas correctement tout le reste du code sera inutile. Dans cette optique il nous a fallut concevoir un code robuste capable de détecter si l'initialisation s'est bien passée ou non, et le cas échéant de recommencer les étapes qui ont même s'il faut reprendre tout l'initialisation si la source du problème persiste.



Un nombre important d'initialisations infructueuses peut mener à un temps de démarrage important, mais au vu des tests effectués nous arrivons à notre objectif souvent de premier coup, parfois au bout de trois, dans ces conditions, le temps de démarrage du module 4G n'excède pas trois min. Pour simplifier la compréhension nous présentons la limite de 'Nb_essais' à 10, mais en pratique ce nombre est à ajuster en fonction des étapes les plus susceptible d'échouer.

Afin de tester la réussite ou non des commandes que l'on lance à l'initialisation nous avons déclaré une variable initStatus qui va prendre pour valeur l'état retourner par la dernière commande (EN_COURS, OK ou FAILED vu précédent) du bloc à tester (init matérielle ou http). De ce fait nous avons assez logiquement placer en dernière commande de chaque bloc celle qui ne fonctionnent qu'en cas de franc succès.

Nous avons donc de nombreux niveaux de test et de répétition des étapes échouée :

a) Niveau commande, chaque commande, tant qu'elle échoue se répète (5 fois max)
b) Niveaux 'bloc', si l'initialisation du bloc (matérielle ou http) à échoué elle est réeffectuée (respectivement 20 fois et trois fois)
c) Enfin, niveau init, une fois les deux niveaux précédents résolus au vérifie que la dernière commande à bien fonctionnée, dans le cas contraire l'initialisation est effectuée à nouveau (DOIT fonctionner, répété un très grand nombre de fois)

ROUTINE DE FONCTIONNENMENT

Le fonctionnement en lui-même est assez simple, ma partie commence lorsqu'une nouvelle valeur en provenance du GPS est détectée. A ce moment les valeurs des variables latitude et longitude sont mises à jour. Et nous allons les inscrire au format JSON dans un fichier sur le

module 4G. La commande pour la création du fichier est la plus complexe que nous avons eu à implémenter : Il faut d'abord écrire la commande suivante :
AT+UDWNFILE="nomFichier",<taille>\r"

Ensuite il faut attendre que le module nous autorise à écrire, il faut donc attendre qu'il nous envoie un charactère « > », une fois ce caractère reçu il nous faut écrire le contenue du fichier que l'on veut créer sur la liaison série.

Le contenu du fichier est créé par concaténation de multiples chaines, notamment les coordonnées mais aussi le formatage JSON. Quand notre chaine ainsi formée à été reçu par le module celui-ci envoi une réponse OK ou ERROR à tester comme pour les autres réponses de ce type.

Il y a ensuite une fonction AT (AT+UHTTPC) à envoyer pour transférer le contenue du fichier crée sur l'api d'Hologram, enfin, on supprime le fichier précédemment crée.

TESTS ET AMELIORATIONS

Les tests en grandeurs réelles ont démontré des problèmes de fiabilité en autre au sein de la routine d'initialisation. Le principe même décrit plus tôt n'est pas remis en cause, en effet, quelques erreurs de logique avait été commises est ont pu être mis en lumière par les tests. La partie de la routine de fonctionnement que j'ai décrit (hors GPS donc) est assez lente, il faut entre cinq et sept secondes pour poster les valeurs. Pour l'instant le temps de rafraîchissement actuel est convenable mais visiblement améliorable.

Nous avons prévu de changer de module, ce changement de devrait pas (ou peut) impacter mon cade car le nouveau module utilise également des commandes AT dont bon nombre en commun. De plus l'architecture du code permet de facilement envoyer d'autre commande. La seule partie à ajouter sera la partie traitement de nouvelle commande.

La partie du code qui vérifie le type de commande en cour est améliorable, on peut y gagner légèrement en lisibilité mais cela prendrait un temps que nous avons jusqu'ici invertie différemment.

Le nouveau module 3G devra gérer la partie GPS, peut-être de nouveau défis que je n'est pas envisagés sont à prévoir…

Gestion des données

Une partie importante de notre projet réside dans la gestion de la 4G, cette gestion doit dans tous les cas passer par un opérateur ce qui va engendrer des coûts supplémentaires. Comme point de départ il faut noter qu'actuellement le client paye un forfait à 10€ par mois pour chaque téléphone avec beaucoup de donnée (bien plus que nécessaire).

Carte SIM

Le choix de la carte SIM est assez complexe, en effet il existe de très nombreuses offres sur le marché mobile. Le choix de notre client semble être le moins coûteux parmi les offres grand public mais 10€/mois reste assez cher.

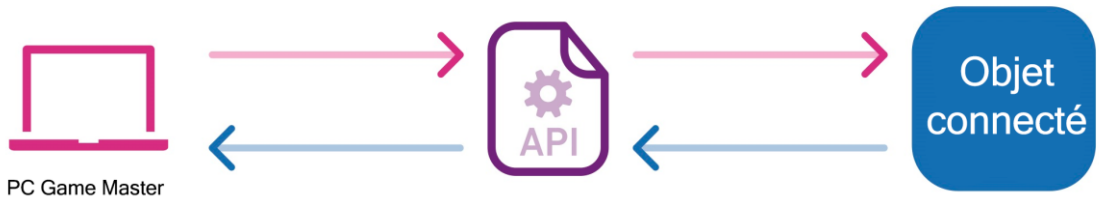
Lorsque l'on se penche sur les offres adaptées aux professionnels et plus spécialement aux objets connectés on en trouve qui semble plus adaptées à nos besoins, les mensualités sont bien plus faibles, de l'ordre de 1 à 3 € parfois il n'y en a même pas. A ce prix va s'ajouter un coût en fonction de la consommation de la carte SIM. Ses forfaits sont parfaitement adaptés à notre premier système de training GPS car un système comme celui-ci va consommer peu de donnée.

Il est pour l'instant difficile de prévoir la consommation de notre tracker GPS car il va dépendre du format de nos données et de l'encapsulation qui peut nous être imposée. Enfin la communication du son des joueurs en direct va constituer la part la plus importante de consommation de donnée si elle doit être comptabilisé. Nous n'avons pas creusé dans cette direction car il ne s'agit pas de notre objectif actuel, un forfait SIM pour l'IoT fera très certainement l'affaire pour le tracker GPS.

Nous avons choisi d'utiliser les services de hologram.io, cette entreprise est spécialisée en tant que fournisseur de carte SIM et de forfait adaptés aux objets connectés. Les tarifs sont assez compétitifs avec un abonnement de $1.50 plus $0.40 par MB. De plus Hologram propose une carte SIM gratuite pour du prototypage. Enfin le site propose un cloud et une API intégrée pour interagir directement avec les cartes.

API web

Une Api Web est un service en ligne sur lequel des données vont pourvoir être traitée.



REST API

REST est l'acronyme de « Representational State Transfert » ces API sont basées sur les ressources, les données et non pas sur des actions. On se concentre d'exprimer sur ce que l'on veut récupérer ou gérer.

Toutes ses APIs utilisent des URI qui correspondent à des identifiants pour des ressources, ces identifiants sont habituellement uniques (sur d'autre API), propre à chaque ressource, mais dans notre cas un URI peut référer à différentes ressources, et une même ressource peut parfois être acquise en utilisant différent URI.

Les données dans les API REST sont en général retournées sont for JSON ou XML (même s'il n'y a pas de contraintes) de cette manière les ressources sont facilement modifiables ou même supprimables pour tous les clients de l'API avec les permissions adéquates.

Six principes fondateurs des API REST :

1. Interface uniforme : Ce type d'API utilise les 4 verbes classiques des requêtes http qui sont : GET, POST, PUT, DELETE appliqués à nos URIs on obtient notre réponse sous forme http (état, « body »)
2. Stateless (« sans état ») : Ceci signifie que le serveur sur lequel est hébergé l'API de détient aucune information sur les clients, il faut donc que chaque requête soit explicite. Si un état doit pris en compte il doit être présent dans la requête.
3. Client-serveur : Il s'agit de comprendre que l'architecture n'est **pas** prévue pour une communication sans discontinuité. Le Client doit clairement demander ce qu'il souhaite obtenir, puis le serveur lui fournit.
4. Mise en cache : Toute information reçu (ou ressource) du serveur doit pouvoir être mis en cache, de même pour ce qui est envoyer au serveur depuis les clients.
5. Système par couche : Il est important de noter que le serveur avec lequel nous échangeons peut avoir d'autres tâches à faire, la réponse à notre requête peut être délayée, mise en cache ou autre. Il y a un système plus ou moins opaque sur le serveur créant **un service discontinu**.
6. Code sur demande : Notre serveur peut donner des réponses sous forme de code source.

En pratique

Hologram propose des explications claires et complètes sur l'utilisation de leur API :

On remplace API_KEY par la clé unique correspondant à notre serveur.

Une requête GET prend cette forme :

```
Curl --verbose --request GET \
'https://dashboard.hologram.io/api/1/users/me?apikey=API_KEY'
```

Une requête POST sous forme JSON prend cette forme :

```
Curl --verbose --request POST \
--header "Content-Type: application/json" \
--data '{"deviceid": 56668, "body": "Hello device!"}' \
```

```
'https://dashboard.hologram.io/api/1/sms/incoming?apikey=API
_KEY'
```

Toutes les réponses sont au format JSON contienne trois paramètres de base :

1. Success : indique si la requête à fonctionnée
2. Data : la donnée (si la requête à fonctionnée)
3. Error : Information si la requête à échouée

Introduction au format JSON

JSON est l'acronyme de JavaScript Object Notation. Il s'agit d'un format lisible pour les humains et les machine qui permet un stockage léger de donnée. Quelques exemples de stockage JSON :

```
json
    element

value
    object
    array
    string
    number
    "true"
    "false"
    "null"

object
    '{' ws '}'
    '{' members '}'

members
    member
    member ',' members

member
    ws string ws ':' element

array
    '[' ws ']'
    '[' elements ']'

elements
    element
    element ',' elements

element
    ws value ws
```

```
string
    '"' characters '"'
characters
    ""
    character characters
character
    '0020' . '10ffff' - '"' - '\'
    '\' escape
escape
    '"'
    '\'
    '/'
    'b'
    'n'
    'r'
    't'
    'u' hex hex hex hex

hex
    digit
    'A' . 'F'
    'a' . 'f'

number
    int frac exp

int
    digit
    onenine digits
    '-' digit
    '-' onenine digits
```

Utilisation de l'API d'Hologram

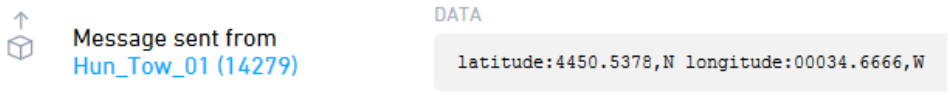
Dans un premier temps nous souhaitions envoyer les données de position sou format JSON, il se trouve que cette opération fut assez compliquée pour plusieurs raisons :

1. La gestion des caractères spéciaux ('{', ',' …) produits par le format JSON est assez chronophage et peut intuitif en langage C, nous forçant à ajouter de nombreux caractères '\' rendant la lecture difficile.
2. De base le fichier que l'on créer sur le module était déjà au format JSON, il nous fallait donc ajouter des données JSON au sein du fichier en lui-même augmentant encore la complexité.

Nous avons donc choisi un formatage de nos données plus simple, nous gardons le format JSON du fichier en lui-même car il est indispensable pour que l'API d'Hologram comprenne ce que l'on transmet, mais les données de position seront un long string sous cette forme :

Longitude:<donnée longitude> latitude:<donnée latitude>

On peut visualiser notre POST sur Hologram :



A l'utilisation nous avons remarqué que nous consomions plus de donnée que prévu, sur le produit fini le client devrait payer 1.5.€ par mois plus 0.40€ par MB suplémentaire. Il est donc imporant que la consommation soit raisonable. En l'état nous avons conssomé presque 3 MB en 1 semaine de tests. Nous n'avons pas pu pour l'instant trouver la source de cette consommation exessive, il faudrait analyser quel type d'action consomme beaucoup mais il y a fort à parier qu'il s'agit du POST qui est donc indispensable et souvent répété.

Conclusion

Le développement à bien avancé, mais par suite du test nous accusons des problèmes de fiabilité et de précision, l'architecture qui a été développée permet de changer notre solution sans avoir à tout refaire ce qui est un vrai avantage. Malgré tout il a y du travail à refaire, d'un coté pour améliorer l'initialisation pour le rendre encore plus fiable et de l'autre pour implémenter la nouvelle partie GPS en espérant atteindre la précision souhaitée.