



---

# Rapport de Projet STL

---

A.Fernandez & S.Ung

Encadré par  
V.Botbol & G.Ziat

Le 15 mai 2017

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Sujet . . . . .	2
1.2	Problématique . . . . .	3
<b>2</b>	<b>État de l'art</b>	<b>4</b>
2.1	Introduction . . . . .	4
2.2	Programmation mobile <i>Android</i> . . . . .	4
2.3	Modèle client-serveur . . . . .	4
2.4	Drone Parrot Bebop 2 . . . . .	4
<b>3</b>	<b>Conception</b>	<b>5</b>
3.1	Introduction . . . . .	5
3.2	Jeu mobile . . . . .	5
3.3	Communication Client - Serveur . . . . .	5
3.4	Communication Serveur - Drone . . . . .	5
<b>4</b>	<b>Implantation</b>	<b>6</b>
4.1	Jeu mobile . . . . .	6
4.2	Client mobile et serveur . . . . .	6
4.3	Communications serveur-drone . . . . .	8
<b>5</b>	<b>Conclusion</b>	<b>10</b>

# 1 Introduction

## 1.1 Sujet

Il va s'agir dans ce projet STL d'étudier la programmation sur les drones dans le cadre des activités robotiques du M2 STL dont les réalisations reposent sur des robots terrestres. Ce projet est donc un moyen d'investigation sur les possibilités offertes par un robot aérien pour un éventuel changement de support robotique.

Pour cela, le problème a été posé d'une manière ludique en demandant la réalisation d'un jeu mobile multi-joueur. Selon les données envoyées par les différents joueurs, qui seront centralisées, le drone se mettra en mouvement vers une certaine direction. L'idée de départ est la conception d'un jeu de rythme de type *Guitar Hero*<sup>1</sup>.

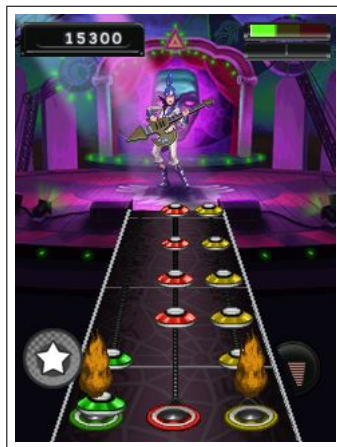


FIGURE 1 – *Guitar Hero 5 Mobile*

Un environnement client-serveur va être mis en place : les joueurs, à partir de leur application mobile, représentent les clients et se connecteront à un serveur hébergé sur un ordinateur dont le rôle sera de rassembler et traiter les scores des joueurs. A partir de ces données, le serveur communiquera avec le drone pour le piloter vers l'un des joueurs.

Plusieurs objectifs clés ont ainsi été définis durant le projet :

- Mettre en place un client/serveur
- Concevoir un jeu mobile
- Communiquer avec le drone
- Commander le drone

---

1. Série de jeux vidéo de rythme éditée par *Activision*

## 1.2 Problématique

L'enjeu majeur de ce projet est le contrôle à distance avec succès du drone. En effet, tout s'articule autour de cet objectif qui demande une programmation sûre sur un robot dont le comportement ne peut pas toujours être prévisible et contrôlé et dont la documentation disponible est très pauvre. Autrement dit, le programme de contrôle du drone se devra d'être fiable d'un point de vue sécurité lors de son utilisation.

## 2 État de l'art

### 2.1 Introduction

Du fait du nombre important de modules nécessaires pour le projet, différentes technologies, dont certaines très récentes, ont été utilisées pendant le développement.

### 2.2 Programmation mobile *Android*

De plus en plus d'applications mobiles ont été développées au cours de ces dernières années, notamment depuis l'arrivée sur le marché des smartphones qui ont contribué à l'explosion du marché des applications mobiles. Le choix du système d'exploitation sur lequel nous allons développer notre jeu mobile s'est porté sur *Android*<sup>2</sup>. Cette décision était pour nous la plus pertinente au vu de la domination de l'OS sur le marché.

Nous nous sommes donc intéressés à la programmation mobile sous *Android* et à sa SDK<sup>3</sup>. Le code associé est en Java.

De plus, nous nous sommes servis du framework *libGDX*<sup>4</sup> qui est une interface de programmation Java multi-plateformes pour développer notre jeu de rythme. Cela étant décidé, il nous fallait trouver un moyen de lier cette partie au reste.

### 2.3 Modèle client-serveur

Dans notre cas, le modèle client-serveur, pour se faire communiquer nos différents programmes au sein d'un réseau local, était parfaitement adapté à la situation. Ce type d'environnement tout à fait classique dans le domaine de la programmation réseau a été étroitement étudié au cours de cette année universitaire, en enseignement de *Programmation répartie* ou encore de *Programmation concurrente, réactive et répartie* par exemple.

### 2.4 Drone Parrot Bebop 2

Le modèle du drone utilisé dans le cadre de notre projet STL est un drone *Parrot BEBOP 2*<sup>5</sup>. Parmi ses spécifications techniques, on peut relever :

- une caméra
- un gyroscope
- un GPS
- une antenne Wi-Fi émettant un signal jusqu'à 300m

En ce qui concerne le développement du programme du drone, nous nous sommes servis de la version 3 de la SDK mise à disposition<sup>6</sup> avec pour documentation quelques programmes exemple et un forum de développeurs.

---

2. Système d'exploitation mobile basé sur le noyau Linux développé par *Google*

3. Kit de développement logiciel

4. <https://github.com/libgdx/libgdx>

5. <https://www.parrot.com/fr/Drones/Parrot-Bebop-2>

6. [https://github.com/Parrot-Developers/arsdk\\_manifests.git](https://github.com/Parrot-Developers/arsdk_manifests.git)

## 3 Conception

### 3.1 Introduction

### 3.2 Jeu mobile

Le jeu mobile consiste en un jeu de rythme avec un écran divisé entre 4 zones verticales. Dans ces zones, des objectifs défilent du haut vers le bas et doivent être touchés par le joueur lorsqu'elles arrivent sur une ligne horizontale en bas de l'écran pour marquer des points.

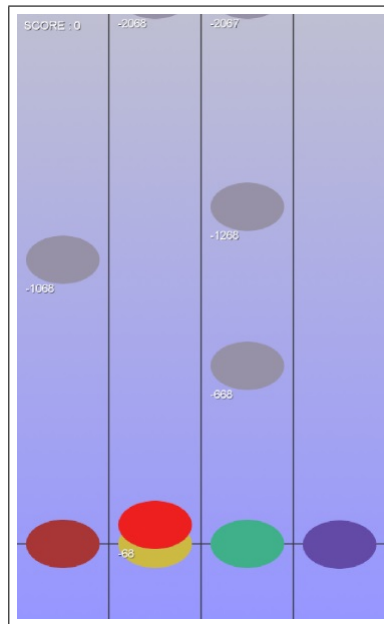


FIGURE 2 – Jeu mobile

Un niveau se résume à une liste de zones (0,1,2,3) qui est associée à une liste d'instants.

Une fois le niveau lancé, le jeu se chargera d'afficher les objectifs aux bons endroits en tenant compte du temps écoulé depuis le début du niveau.

### 3.3 Communication Client - Serveur

La première chose qu'il fallait faire était de mettre en place une architecture client-serveur le plus rapidement possible et établir un protocole de communication. Pour des raisons de simplicité nous avons opté pour un protocole textuel, dont la définition est la suivante :

### 3.4 Communication Serveur - Drone

## 4 Implantation

### 4.1 Jeu mobile

Pour le jeu mobile, nous avons eu besoin de trouver une bibliothèque permettant de faire des jeux sous Android.

Nous avons choisi *libGDX* car elle nous a permis d'obtenir assez rapidement un résultat testable pour le jeu mobile. Cependant, cette bibliothèque est une bibliothèque multi-plateformes qui fonctionne avec un coeur compatible avec toutes les plateformes et un programme lançant le coeur pour chaque plateforme où l'on souhaite porter le jeu.

Le jeu fonctionne grâce à une méthode `loop` qui prend en argument le temps qui s'est écoulé depuis le dernier appel à cette fonction. Le jeu étant assez simple, cette fonction s'occupe des entrées utilisateurs, de la mise à jour de l'état du jeu et des affichages.

Cette architecture s'adapte particulièrement bien à un jeu pour un joueur disponible sur plusieurs plateformes. En effet, le lanceur *Android* s'occupe dans notre cas de toutes les communications, mais pour assurer des communications entre le lanceur et le jeu, il a fallu définir deux interfaces de communications :

- Une pour permettre au jeu de notifier les changements de score au lanceur
- Une pour permettre au lanceur de notifier au jeu une fin de partie et le nom du vainqueur

Pour équilibrer le jeu nous avons dû construire un intervalle de temps autour du moment où un objectif doit être appuyé. Une réaction précoce de la part du joueur a été privilégié par rapport à une réaction tardive.

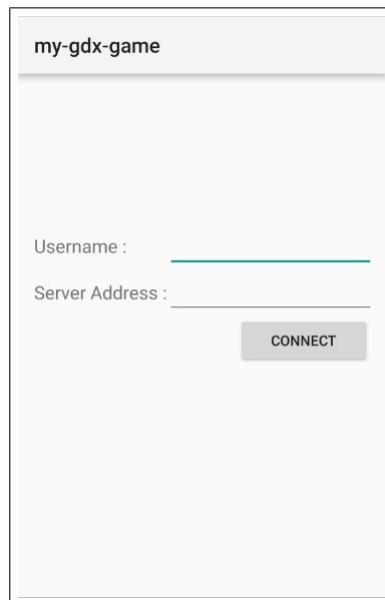
### 4.2 Client mobile et serveur

Le client mobile s'occupe de la connexion au serveur et de lancer le jeu. Une première activité lui permet de saisir l'adresse du serveur et son nom d'utilisateur, elle permet aussi d'afficher les messages d'erreur de connexion. Une seconde activité affiche une liste d'utilisateurs connectés, autrement dit le *lobby* ou *salon* de la partie, ainsi que deux boutons **ready** et **leave**.

Le serveur s'occupe d'une partie : il commence par attendre que les joueurs se connectent sur le lobby de la partie. Lorsque tout les joueurs ont signalé qu'ils sont prêts, il lance la partie et notifie à tout les clients de lancer le jeu.

Un seul niveau prédéfini de longueur fixe est disponible pour le moment. Lorsque les joueurs atteignent un objectif, en le touchant au bon moment, leur client envoie un message indiquant le score qu'ils ont obtenu. Le serveur met donc à jour le score des différents joueurs et, lorsque le temps imparti est écoulé, notifie les clients que la partie est terminée et leur indique le gagnant.

Pour implémenter ces communications nous avons utilisé le protocole UDP. Une difficulté rencontrée sur cette partie fût que le message permettant au serveur de notifier le client que sa connexion est acceptée et de lui octroyer son numéro de communication (permettant d'identifier le client à l'origine du message reçu), n'était pas reçu dans 100 % des cas. Plusieurs séances de tests ne posèrent aucuns problèmes tandis que, d'autres fois, la connexion échouait côté client, laissant un « client fantôme » sur le serveur.



The screenshot shows a mobile application interface titled "my-gdx-game". Below the title bar, there are two input fields: "Username :" and "Server Address :". The "Username :" field has a green underline. Below these fields is a grey button labeled "CONNECT".

FIGURE 3 – Connexion



The screenshot shows a mobile application interface titled "my-gdx-game". Below the title bar, the word "Lobby" is displayed. Underneath, the name "Sylvain" is shown. At the bottom of the screen is a grey button labeled "READY".

FIGURE 4 – Lobby



L'origine de ce problème n'a pas pu être identifié à ce jour et n'est remarqué que sur des communications entre des appareils android et un ordinateur. Il peut être corrigé en passant par des communications TCP qui sont protégées contre la perte de paquets et les erreurs de transmission, permettant ainsi de s'abstraire de l'identifiant des clients, car les communications côté serveur dans ce cas seraient distribuées sur plusieurs *socket*.

### 4.3 Communications serveur-drone

Pour établir les communications entre le serveur et le drone, nous nous sommes portés sur les API permettant de piloter notre drone à distance. Le drone étant un *Parrot BEBOP 2* nous nous sommes penchés sur la SDK ARSDK fourni par le constructeur. Cette SDK étant en cours de développement, nous avons eu des difficultés à l'utiliser. Au premier abord, il semblait y avoir une version compatible avec Java, le langage de programmation utilisé aussi bien pour le serveur que pour le client.

Cependant cette version de la SDK n'était destinée qu'à développer des applications mobiles android, nous avons du considéré les natives écrites en C à la place.

Pour comprendre comment utiliser ces natives, nous avons étudié l'exemple fournit : la documentation étant trop pauvre pour suffire. Cet exemple permet de piloter un drone à distance à l'aide d'un ordinateur connecté en Wi-Fi et d'avoir un retour vidéo du drone. Comme la communication entre l'appareil manipulant le drone et celui-ci est en Wi-Fi, le serveur de jeu, et, par extension, les clients android devront être connecté au réseau sans fil du drone pour pouvoir communiquer entre eux.

Nous avons commencé par faire de courts programmes C permettant de nous connecter au drone et de le faire décoller puis avancer en ligne droite, reculer et répéter ces opérations plusieurs fois avant de se poser pour terminer. La principale difficulté pour cette partie de tests fut de trouver des moments de libre à la fac où le drone était disponible, cela nous a considérablement ralenti dans notre développement.

Nous avons ensuite décidé d'appeler à la place des routines C depuis Java en créant un programme supplémentaire chargé de piloter le drone et le connecter au serveur de jeu par le biais d'une *socket* TCP. Par mesure de sécurité, nous avons décidé de mettre en place un mode de pilotage manuel sur lequel on peut basculer à tout moment.

Il nous reste à implémenter le déplacement du drone en fonction du score des joueurs et de l'avancement du niveau. L'idée en théorie serait, par exemple pour deux joueurs, de positionner le drone au centre d'un segment passant par les deux joueurs, et définir la distance par rapport au centre du segment avec l'avancement du niveau dans la direction du joueur gagnant. Une autre possibilité serait de décider d'une fréquence pour déplacer le drone dans la direction du joueur qui mène la partie.

La dernière difficulté réside en les mouvements du drone qui sont imprécis risquant de le faire dévier à cause des facteurs extérieurs ou même des moteurs. Un *callback* permet de vérifier l'orientation et l'inclinaison du drone, on pourrait

FIGURE 5 – Schéma du déroulement d'une partie

donc après chaque commande de mouvement, corriger la direction en fonction des données recueillies par ce *callback*.

On pourrait aussi borner la position du drone, grâce à ses coordonnées GPS mais celles-ci sont trop imprécises pour obtenir une gestion aussi fine qu'on voudrait avoir. Combiner des données tels que les retours moteurs, la puissance du signal Wi-Fi entre les appareils et le drone et les coordonnées GPS pourrait s'avérer utile pour prévenir des risques d'une imprécision sur la position et les mouvements du drone.

## 5 Conclusion