Université Pierre et Marie Curie Faculté d'Ingénierie

Rapport de Projet STL

A.Fernandez & S.Ung

Encadré par
V.Botbol & G.Ziat

Le 21 mai 2017

Table des matières

1 Introduction

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* [1].

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

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. Autrement dit, le programme de contrôle du drone se devra d'être fiable d'un point de vue sécurité lors de son utilisation.

Tout d'abord, nous présenterons les différentes techniques mises en œuvre à la réalisation du projet. Ensuite, nous partagerons nos choix de conception sur le développement que nous comptons mener. Puis nous discuterons des différentes étapes d'implantation ou la mise en pratique des réflexions qui ont été faites en amont. Enfin, nous terminerons par un bilan du projet.

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

2 État de l'art

Nous allons présenter dans cette partie les différentes technologies nécessaires qui ont été utilisées pour mener à bien le projet. En effet, du fait du nombre important de modules nécessaires, différentes technologies, dont certaines très récentes, ont été utilisées pendant le développement.

2.1 Programmation mobile *Android*

Le jeu mobile a été développé sur *Android*, le système d'exploitation mobile proposé par *Google*. Nous nous sommes donc intéressés à la programmation mobile sous *Android* et à sa SDK [2] qui nous permet de développer des programmes en Java.

De plus, nous nous sommes servis du framework *libGDX* [3] 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.

2.2 Modèle client-serveur

Dans notre cas, le modèle client-serveur, pour 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.

2.3 Drone Parrot Bebop 2

Le modèle du drone utilisé dans le cadre de notre projet STL est un drone *Parrot BEBOP 2* [4]. Parmi ses spécifications techniques, on peut relever :

- un poids de 500g
- une autonomie de 25min
- 4 hélices flexibles motorisées
- une vitesse de pointe de 60km/h en horizontal et de 21km/h en vertical
- une caméra Full HD permettant de filmer et photographier
- un GPS pour le contrôle de vol et le dispositif de suivi/retour automatique
- une antenne Wi-Fi émettant un signal jusqu'à 300m

2. Kit de développement logiciel
3. `https://github.com/libgdx/libgdx`
4. `https://www.parrot.com/fr/Drones/Parrot-Bebop-2`

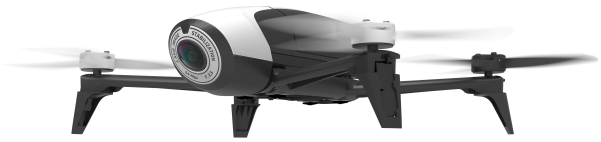

FIGURE 1 – Drone Parrot BEBOP 2

Une application fournie par le constructeur appelée *Free Flight Pro* permet de piloter l'appareil via un smartphone sous *Android* ou *iOS* connectée à travers son propre réseau Wi-Fi. De nombreuses fonctionnalités sont proposées par l'application en plus des commandes de contrôle du drone comme la possibilité de faire des figures prédéfinis ou de partager ses données de navigation avec d'autres pilotes.

En ce qui concerne le développement du programme du drone, nous nous sommes servis de la version 3 de la SDK mise à disposition [5] avec pour documentation quelques programmes exemple et un forum de développeurs.

5. `https://github.com/Parrot-Developers/arsdk_manifests.git`

3 Conception

Le choix du système d'exploitation mobile sur lequel nous avons développé le jeu mobile s'est porté sur *Android* du fait de la domination sur le marché des appareils mobiles qui en sont équipés. Ce choix nous paraissait donc pertinent et approprié pour toucher un plus large public mais aussi pour les tests car nous sommes nous même des possesseurs de mobiles *Android*.

3.1 Jeu mobile

Tout d'abord, nous avions besoin d'un jeu non coopératif dans lequel il est possible, à tout moment, de connaître le joueur en tête et de borner la durée d'un niveau. Ces critères peuvent s'adapter à de nombreux types de jeu, comme, par exemple, des jeux de courses, de combat ou bien, dans notre cas, de musique.

Une forme classique de ce type de jeu consiste en un jeu de rythme avec un écran formé de 4 zones verticales. Dans ces zones, des objectifs défilent en descendant et doivent être touchées par le joueur lorsqu'elles arrivent sur une ligne horizontale en bas de l'écran. Les objectifs représentent des notes de musiques qui doivent être jouées au bon moment afin de suivre la bande son du niveau.

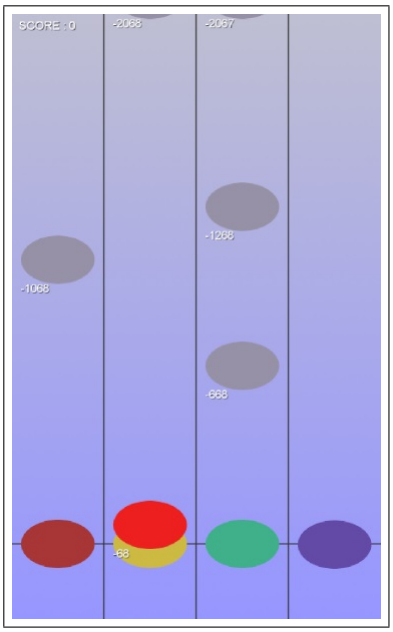

FIGURE 2 – Jeu mobile

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

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

Nous avions pensé qu'il pourrait être intéressant de se renseigner sur la possibilité de générer (peut-être partiellement) un niveau de ce jeu à l'aide d'une musique et d'outils d'analyse musicale. Cette idée a finalement été mise

à l'écart pour sa difficulté et ne trouvait pas sa place dans le problème posé par le projet.

3.2 Communication Client - Serveur

La première chose qu'il a fallu établir 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 :

1. Une première étape consiste en un échange de messages entre le client et le serveur pour inscrire correctement le joueur dans la partie.
2. L'étape suivante permet d'initier la partie entre tous les clients inscrits grâce à un message de confirmation qui est attendu de la part de tous les joueurs.
3. Durant la partie, les clients envoient périodiquement au serveur le score obtenu et permet ainsi à ce dernier de déterminer le joueur vers qui le drone doit se diriger.

3.3 Communication Serveur - Drone

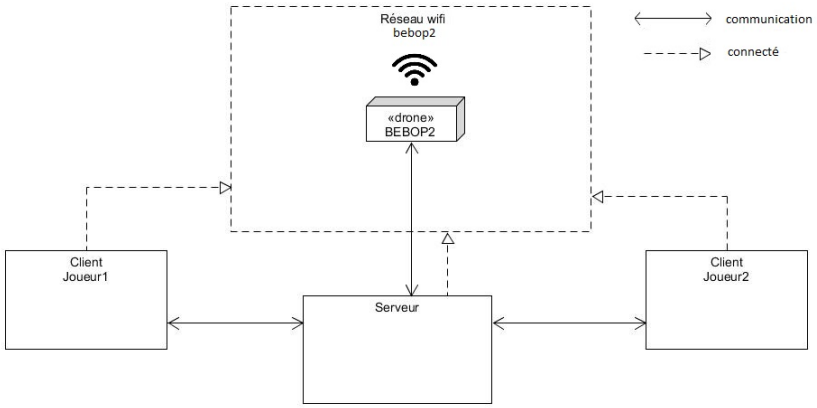

FIGURE 3 – Schéma de communication entre les différents appareils

La partie communication entre le serveur et le drone est le point le plus délicat du projet et se trouve au cœur de notre problématique. En effet, on rappelle que l'objectif principal est le pilotage du drone par un programme « fait maison ». Contrairement à la communication entre les clients mobiles et le serveur, aucun protocole n'a dû être spécifié manuellement puisque nous nous sommes servis des outils de développement mis à disposition par le constructeur pour les développeurs. En effet, comme expliqué plus loin dans la section 4.3 l'utilisation des primitives C ont permis d'établir la connexion avec le drone et de lui envoyer des commandes de déplacement avec un certain niveau d'approche bien qu'il ne soit pas très élevé sans pour autant être trop compliqué d'un point

de vue mécanique. Les déplacements sont gérés par un jeu d'instructions qui influencent l'inclinaison de l'appareil et la vitesse de rotation du lacet [6].

Plusieurs tests et observations ont été nécessaires pour connaître les effets des différentes instructions sur les mouvements du drone ce qui a rendu cette étape relativement difficile. À cela viennent s'ajouter des conditions de travail non optimales à savoir une disponibilité limitée du drone pour effectuer des tests ainsi qu'un environnement de test non idéal pour prévenir des dommages sur l'appareil qui sont difficilement évitables.

Par exemple, plusieurs hélices ont été endommagées au point qu'il nous a fallu en commander ou quelques blessures dues à une mauvaise manipulation ont été subies. L'autonomie relativement limitée constituait elle aussi un obstacle dans les tests qui se retrouvaient ainsi prématurément suspendus.

6. En aéronautique, désigne le mouvement de rotation horizontal du mobile autour d'un axe vertical

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 cœur 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 du 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*[7] affiche un formulaire qui permet à l'utilisateur de saisir l'adresse du serveur et son pseudonyme. Elle permet aussi d'afficher les messages d'erreur de connexion. Une seconde *activité* affiche une liste d'utilisateurs connectés, que l'on appellera 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 considéré l'utilisation du protocole `UDP`. Une difficulté rencontrée sur cette partie fut que le message permettant au serveur de notifier le client que sa connexion est acceptée pour lui attribuer un numéro de communication (permettant d'identifier le client à l'origine d'un message), n'était pas systématiquement reçu.

7. En développement *Android*, désigne un état de l'application mobile sur lequel une action utilisateur peut être attendue

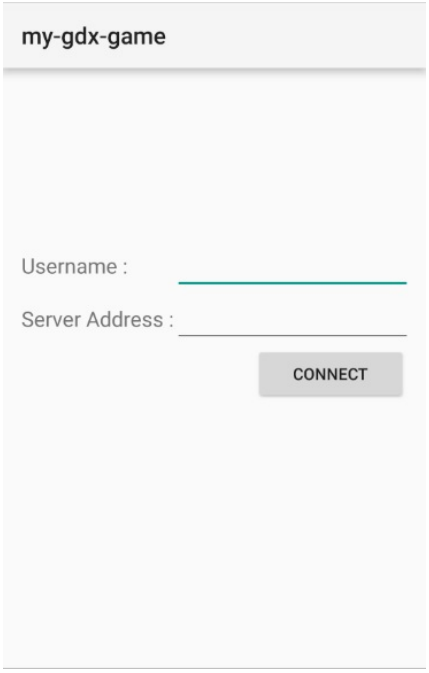

FIGURE 4 – Écran de connexion

FIGURE 5 – Salon de jeu

De ce fait, nous avons décidé qu'il valait mieux, de par la simplicité du jeu mobile et le fait qu'il était important de maintenir une connexion entre les joueurs et le serveur, adopter le protocole `TCP`, qui est protégé 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 dirigés vers les API permettant de piloter notre drone à distance. Le drone étant un *Parrot BEBOP 2* nous nous sommes intéressé à la SDK ARSDK fourni par le constructeur. Nous avons rencontré des difficultés pour mettre en place l'environnement de développement : la méthode de compilation n'était pas usuelle et la gestion des dépendances s'est avérée compliquée. À première vue, il semblait y avoir une version compatible avec Java, le langage que nous avons utilisé pour le serveur et pour le client via la SDK *Android*.

Cependant cette version de la SDK n'était destinée qu'à développer des applications mobiles *Android*, nous avons du considérer l'API C à la place.

Pour comprendre comment utiliser les primitives disponibles, nous avons étudié l'exemple fourni : 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 connectés au réseau sans fil du drone

pour la communication.

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 considérable 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.

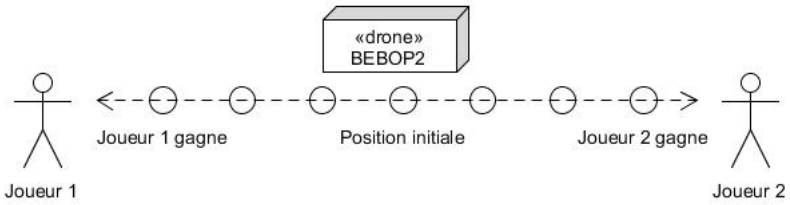

FIGURE 6 – Schéma du déroulement d'une partie

5 Conclusion

Au terme de ce projet STL, il apparaît que nous avons su correctement répondre aux besoins du sujet : la possibilité de créer une architecture client-serveur avec manipulation de drone.

Ce projet riche en terme de technologies utilisées, comme on a pu le montrer dans ce rapport, a exigé de nous une certaine polyvalence. Cela nous a valu un bon approfondissement de nos connaissances ou même apprentissage de nouvelles notions qui sont aujourd'hui très récentes : le marché des drones commerciales est, en effet, nouveau et en pleine expansion.

La programmation mobile *Android* était pour nous une première expérience ; la lecture et compréhension de code libre étaient aussi des éléments clés pour la partie pilotage de drone. On peut aussi relever des compétences annexes très utiles pour ce projet qui sont : l'utilisation d'un gestionnaire de version [8] et le langage de programmation LaTeXpour la rédaction de ce rapport.

Module	**Nombre de lignes de code**
Serveur Java	≈ 1000
Client Android	≈ 1400
Pilotage du drone	≈ 1200

En l'état actuel, comme la section 4.3 l'a souligné, le serveur de jeu n'est pas encore capable d'envoyer des instructions au drone en fonction du score des joueurs. En réalité, le programme de pilotage est séparé du serveur puisqu'on nous avons dû l'écrire en C. L'idée serait alors de mettre en place une communication supplémentaire qui soit locale à la machine entre le serveur et le « programme pilote ».

Une autre question se pose sur la manière dont le drone doit être piloté au cours d'une partie. Le problème étant que le drone pourrait avoir à se déplacer sur des distances imprévisibles. Et si les mènent la partie à tour de rôle, le drone ferait des allers-retours fréquents. Une solution serait de décider d'une fréquence, par exemple toutes les 10 secondes, pour déplacer le drone dans la direction du joueur en tête.

Enfin, le dernier problème est que les mouvements du drone sont imprécis et qu'il n'est pas stable. Celui-ci risque de dévier à cause de facteurs extérieurs ou moteurs. Un *callback* [9] envoyé à intervalles réguliers par le drone permet de vérifier l'orientation et l'inclinaison du drone, si bien qu'une première possibilité serait, qu'après chaque commande de mouvement il est possible de corriger l'orientation et la direction du drone en fonction des données recueillies.

Une autre possibilité serait de se servir des coordonnées GPS mais celles-ci sont trop imprécises sur de courtes distances. Croiser des facteurs tels que les retours moteurs, la puissance du signal Wi-Fi entre les appareils et le drone et les coordonnées GPS pourraient s'avérer utiles pour obtenir une approximation de la position et des mouvements du drone.

8. `https://github.com/LexTek/RockAn-Dron`
9. Fonction de rappel