Université de Technologie de Compiègne

Rapport mini-projet MI12

Interaction en temps-réel entre smartphones

15 juin 2016

BOSQ Romain

GUYET Florian

ZINS Pierre

Table des matieres

1 Introduction

Dans le cadre du mini-projet de MI12, nous avons développé une application Android en rapport avec notre sujet : “Interaction en temps réel entre smartphones”. Notre application consiste en un jeu de Pong, pour lequel plusieurs modes sont possibles. Le jeu peut se dérouler entre deux appareils, ou alors entre 2 appareils et un troisième. Par ailleurs, les communications peuvent se faire par Wifi Direct ou par le réseau local.

2 Cahier des charges

2.1 Sujet

L'objectif de ce sujet est d'analyser la possibilité d'obtenir un comportement déterministe de fonctions implantées sur un Smartphone, et qui sollicitent les capteurs intrinsèques (GPS, gyroscopes, etc.) L'idée est de mettre en oeuvre ceci en faisant interagir deux Smartphones entre eux en temps réel. On peut imaginer par exemple un ping-pong virtuel :

- Le premier lance la balle
- Elle est reçu par le second et renvoyée
- Etc.

2.2 Déroulement

Afin de mettre en œuvre ce sujet dans les meilleures conditions, il faudra :

- S'assurer que chaque fonction logicielle sera analysée selon les principes de bases des contraintes temps réel :
  - Déterminisme
  - Calcul du WCET (Worst Case Execution Time)
  - Estimer la garantie de Périodicité
- Garantir des fréquences d'échantillonnage des capteurs. Chaque échantillon sera suivi par un traitement dont le WCET aura été calculé avec précision

2.3 Plateforme

Plateforme Android fonctionnant au minimum sous SDK 4.4.4.

2.4 Approche

Les travaux doivent commencer par la mise au point des exigences fonctionnelles et extra-fonctionnelles du produit. Ensuite, les spécifications fonctionnelles doivent être modélisées en utilisant une ou plusieurs méthodes

connues dans l'état de l'art de la conception. Les travaux de développement doivent suivre une approche incrémentale. La première version de la fonction ou de l'algorithme doit être simple, réalisant le cœur de la fonction demandée. Ensuite, plusieurs versions d'amélioration permettront de répondre à la totalité des spécifications.

3 Exigences

Plusieurs exigences ont été définies en amont du projet. Elles ont permis de définir les directions à suivre tout au long de la réalisation de ce projet. Nous avons établi 2 exigences extra-fonctionnelles ainsi que 13 exigences initiales. Au cours du développement, nous avons affiné et ajouté quelques exigences.

Exigences extra-fonctionnelles

Id	Exigence
Req-A	L'application finale devra fonctionner sur les appareils Android.
Req-B	L'application finale devra être compatible avec les versions Android 4.4 (KitKat) et supérieures.

Exigences initiales

Id	Exigence
Req-01	Il y aura un mode de fonctionnement principal avec 2 smartphones. Le jeu devra être dupliqué sur l'écran des deux appareils. Chaque smartphone représentera un joueur.
Req-02	Mode 2 smartphones : La rotation autour de l'axe Y devra faire bouger le joueur horizontalement (capteur de gravité)
Req-03	Mode 3 smartphones : La rotation autour de l'axe X des deux "manettes" fera bouger le joueur horizontalement (capteur de gravité)
Req-04	La latence de transmission entre les deux smartphones (joueurs) devra rester raisonnable.

Req-05	La connexion entre les deux smartphones se fera par Wif-Fi direct ou par le réseau local.
Req-06	Une fois la connexion faite, un des smartphones devra être désigné comme serveur ou *group owner* et comme client.
Req-07	Le client va récupérer ses déplacements à partir du capteur de gravité et enverra les déplacements de son joueur au *group owner*.
Req-08	Le *group owner* devra récupérer la position du joueur client et utiliser également son capteur gravité pour obtenir ses propres déplacements. Il déroulera ensuite le jeu à partir de ces données. Il s'occupera notamment du déplacement réaliste de la balle.
Req-09	Comme le jeu se déroule uniquement sur le téléphone *group owner*, il devra ensuite envoyer des éléments du jeu (positions des joueurs et de la balle) à l'autre smartphone. Ce dernier effectuera un simple affichage.
Req-10	Les rôles *group owner* et *client* ne devront pas influencer sur le jeu. La difficulté devra être la même.
Req-11	Un second mode de fonctionnement est envisagé avec 3 smartphones. Un smartphone devra faire office de maître et affichera le jeu. Les deux autres smartphones seront les joueurs. Chaque joueur enverra ses déplacements au maître. Les smartphones joueurs seront ainsi les manette du jeu, tandis que le smartphone maître servira d'écran.
Req-12	L'ensemble des modes différents devront être accessibles facilement dans l'application.
Req-13	Pour le WifiDirect, la déconnexion entre deux smartphones devra également être envisageable.

Ajout et élicitation des exigences

Req-14	Afin d'avoir un jeu fluide, il sera nécessaire de conserver une latence de transmission entre les deux smartphones, inférieure à quelques dizaines de millisecondes (15ms-30ms).
Req-15	Dans les deux modes de jeu (2 smartphones / 3 smartphones), la connexion devra être faite au sein de l'application avant le début du jeu et de manière simple.

Req-16	Le jeu devra être lancé sur les deux ou trois smartphones à partir d'un bouton. Ce dernier permettra de créer la connexion et de lancer le jeu. Un des smartphones devra initier la connexion et le second (cas 2 smartphones) ou les deux autres (cas 3 smartphones) rejoindra/ons simplement la partie.
Req-17	Afin que l'affichage soit fluide pour l'œil humain, sa fréquence de rafraîchissement devra rester au minimum au alentour de 60Hz. (Un rafraîchissement toutes les 16 secondes).

4 Application

4.1 Principe

Notre application consiste donc en un jeu de pong dans lequel les joueurs se déplacent en bougeant leur téléphone. Nous avons développé deux modes différents :

- **mode 2 smartphones** : Chaque téléphone représente un joueur et le jeu est affiché sur les deux téléphones. Les joueurs se déplacent dans le jeu en bougeant leur smartphone.
- **mode 3 smartphones** : Un smartphone est utilisé comme écran et les deux autres smartphones font office de "manette de jeu" pour chaque joueur. Dans ce mode, le jeu n'est affiché que sur un smartphone principal.

Les deux jeux sont très similaires, seules les communications entre les appareils changent.

4.2 Capteurs

Dans notre application, chaque joueur doit pouvoir bouger de manière horizontale sur dans le jeu.

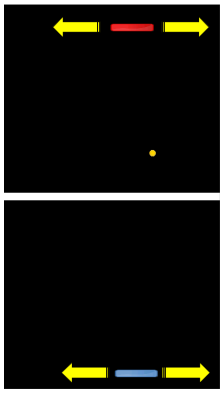


FIGURE 1 – Principe du jeu

Pour cela, nous avons choisi d'utiliser le capteur "Gravity" sur l'axe X pour le mode 2 smartphones et sur l'axe Y pour la version avec 3 appareils.

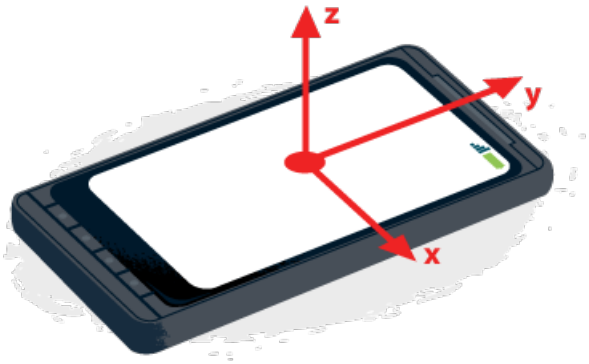


FIGURE 2 – Orientation Smartphone

L'utilisation de ce capteur présente plusieurs avantages :

- la gravité est nulle au départ quand le téléphone est à “plat”. Une légère inclinaison vers la droite fera augmenter la valeur de la gravité et une inclinaison vers la gauche la fera décroitre. La valeur obtenue par ce capteur sera donc simple à traiter et à utiliser.

- Comme le jeu se déroule sur l'écran du téléphone, il fallait trouver un mouvement qui permettait au jeu de rester visible pour le joueur pendant les mouvements. Ainsi, il n'était pas vraiment possible d'utiliser le capteur d'accélération linéaire pour le déplacement des joueurs. Il aurait été difficile pour le joueur de suivre le jeu pendant qu'il déplace le téléphone. Par ailleurs, une donnée représentant une accélération linéaire suivant un axe aurait également été plus difficile à traiter et à utiliser pour le déplacement des joueurs.

Pour la version avec 2 smartphones, nous avons donc utilisé la gravité suivant l'axe X, car le mouvement est très simple pour le joueur lorsqu'il tient son téléphone en main. Cela, lui permet aussi de suivre le jeu en même temps. Cependant, dans le version avec 3 smartphones, nous avons utilisé l'axe Y. Dans ce mode, le jeu n'est pas affiché sur appareil des joueurs, et un mouvement suivant l'axe Y est plus adapté en raison de la forme allongée (verticalement) des smartphones, comme si le joueur tenait une manette.

4.3 Connexion

Afin de pouvoir faire interagir les deux smartphones, il a fallu créer une connexion entre eux. Nous avons dans un premier temps étudié la connexion par Bluetooth. Elle est rapide à mettre en place, stable mais ne présente pas une portée et un débit suffisamment important. Nous avons donc décidé d'utiliser du Wifi Direct.

Il s'agit d'une connexion Wifi P2P qui se fait directement entre deux appareils. Il n'est donc pas nécessaire d'être connecté à un même réseau local, ou encore à Internet. La connexion en Wifi Direct est intégrée à notre application, nous n'avons donc pas besoin de créer une connexion entre les deux smartphones avant le démarrage de l'application.

Lors des échanges préalables à la connexion, les deux appareils recevront une adresse IP et l'un d'entre eux recevra le statut de *group owner*. Le *group owner* pourra ouvrir et écouter un port tandis que le second appareil pourra envoyer des messages au *group owner* puisque son adresse est connue. Une

fois que le *group owner* a reçu un message de la part de l'autre smartphone, il connaîtra son adresse IP et les communications pourront se faire dans les deux sens.

Pour la version avec trois smartphones, le principe reste similaire. L'appareil affichant le jeu sera le *group owner* et sera connecté en Wifi Direct à chacun des appareils joueur. Dans un second, temps, nous avons décidé de mettre également en place une version de connexion par le réseau local. Le principe reste le même, seule l'obtention des adresses IP change. Nous avons ajouté un champs de saisie dans lequel un des deux joueurs doit entrer l'adresse IP de son adversaire (cas avec 2 smartphones) ou alors les deux joueurs entrent l'adresse IP de l'appareil qui affiche le jeu (cas avec 3 smartphones).

4.4 Communication

4.4.1 Mode avec 2 smartphones

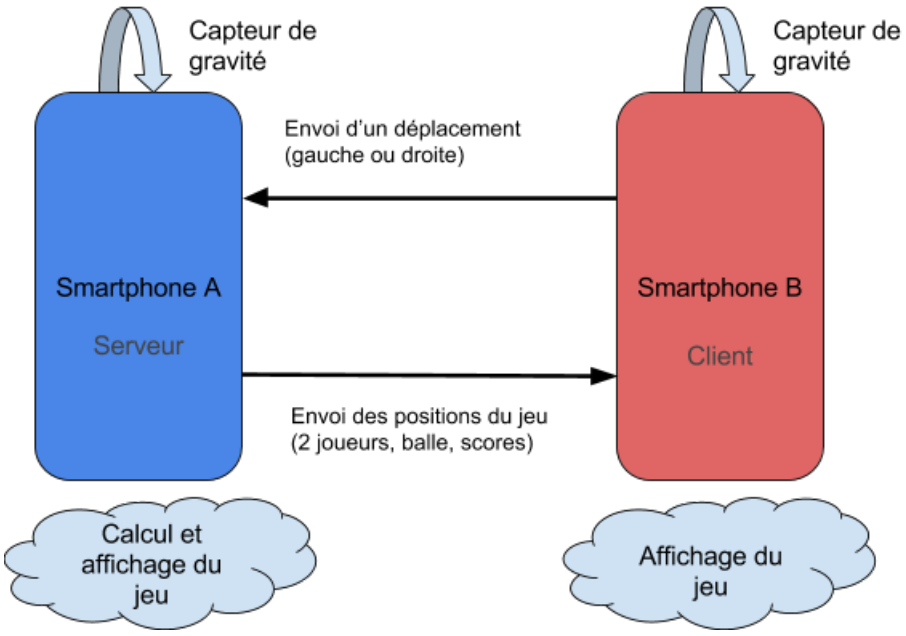


FIGURE 3 – Mode de communication entre les 2 smartphones

Dans ce mode de jeu, nous distinguons un smartphone “serveur” qui va calculer et dérouler le jeu et un smartphone “client” qui enverra simplement les déplacements du joueur et qui recevra les données du jeu et les affichera. Au niveau du code, nous avons utilisé des “AsyncTask” et de simples “Thread” pour la partie communication.

4.4.2 Mode avec 3 smartphones

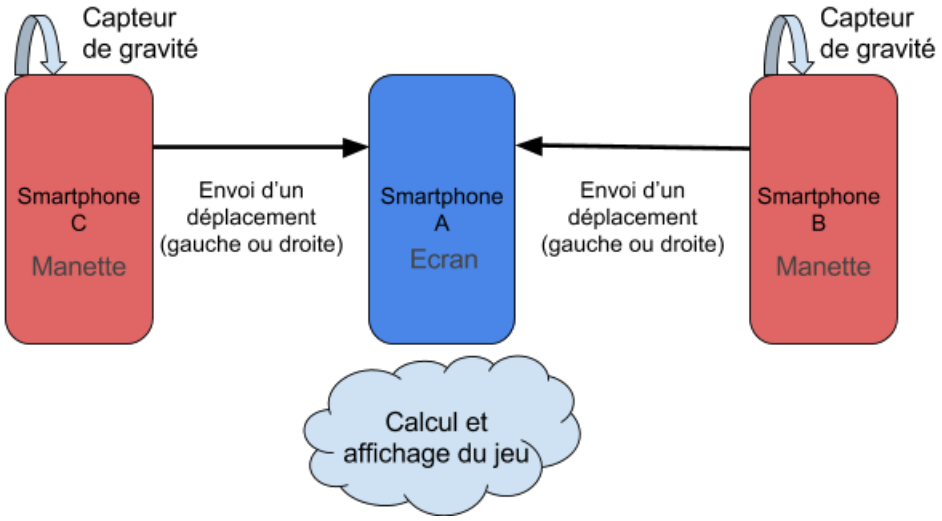


FIGURE 4 – Mode de communication entre les 3 smartphones

Dans cette version, les communications sont simplifiées. Seuls les appareils “manette” des joueurs envoient des informations concernant leur déplacements au serveur “écran”. Ce dernier ouvre donc 2 ports et écoute. Chaque smartphone “manette” enverra ses données sur un des deux ports au serveur. Le jeu se déroulera uniquement sur l'appareil ”serveur”.

4.5 Diagrammes

Voici un diagramme de séquences qui présente les liens entre les différentes activités du jeu.

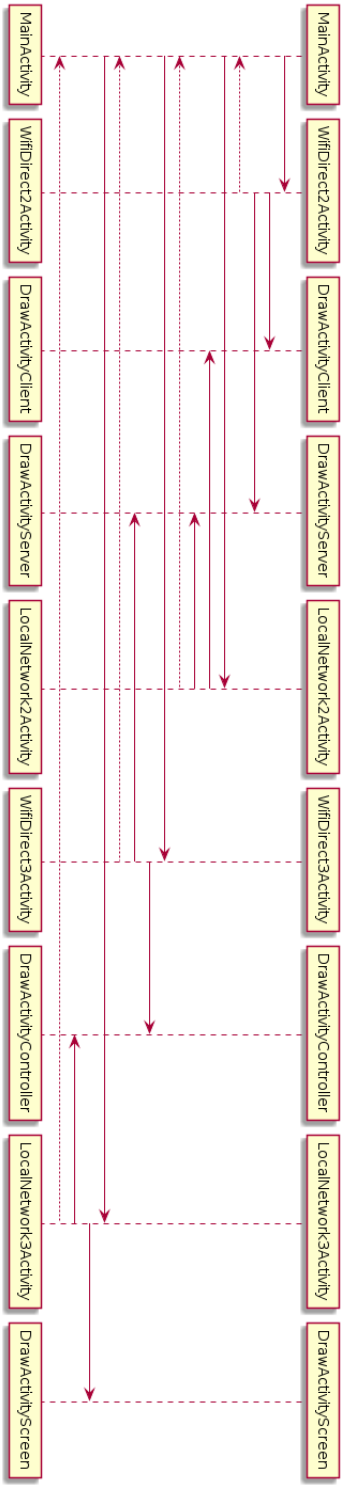


FIGURE 5 – Diagramme de séquences

Les deux diagrammes suivant (UML) présentent l'architecture des classes nécessaires au jeu et aux communications. Le premier s'intéressent à la version du jeu avec 2 smartphones et le second à la version avec 3 smartphones.

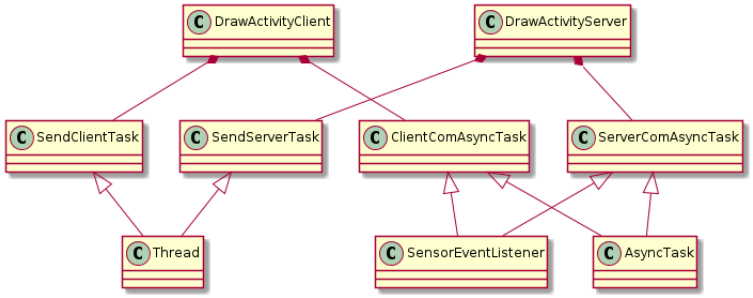


FIGURE 6 – UML : jeu et communication 2 smartphones

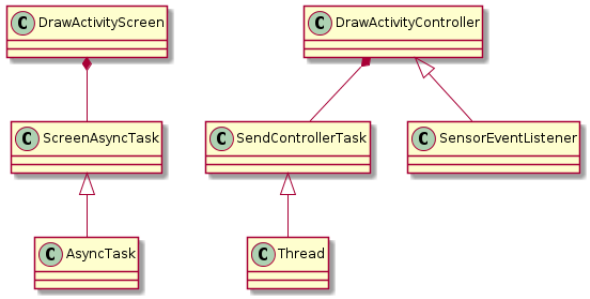


FIGURE 7 – UML : jeu et communication 3 smartphones

4.6 Performances

4.6.1 Capteurs

Le capteur de gravité est un outil qui se trouve au cœur notre projet puisqu'il est utilisé pour le déplacement des joueurs. Nous allons donc analyser la fréquence d'acquisition des valeurs de ce capteurs afin de nous assurer que cette fréquence est suffisamment grande pour le bon déroulement du jeu.

Nous avons effectué une série de tests afin de mesurer la différence de temps entre deux actualisations de la valeur du capteur gravité.

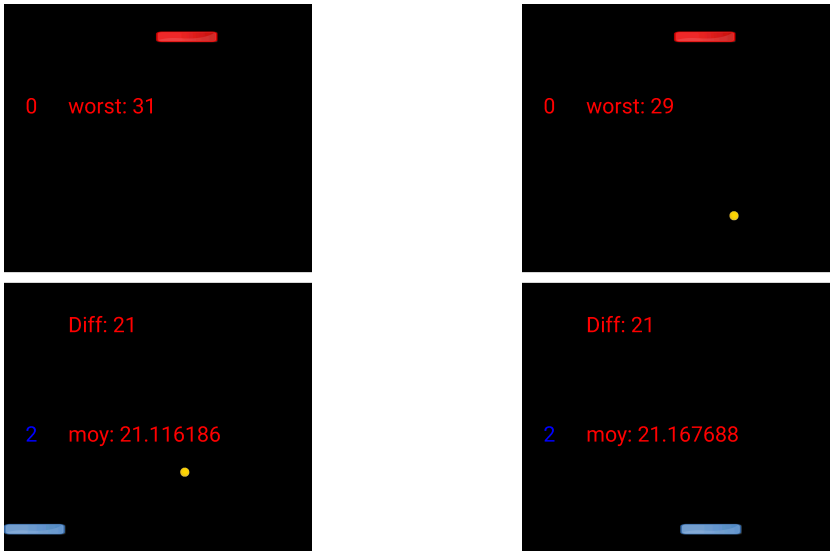


FIGURE 8 – Captures d'écran de la série de test

Nous avons effectué trois séries de mesures qui consistaient à faire tourner le jeu pendant une minute. Comme nous pouvons le voir sur les captures d'écran ci-dessus, la valeur du capteur gravité s'actualise toutes les 21 millisecondes en moyenne et nous avons mesuré un WCET de 31 millisecondes.

Ces mesures ont été effectuées depuis un Samsung Galaxy S6. Il faut noter que les performances des capteurs dépendent du smartphone, et qu'elles peuvent varier d'un téléphone à un autre.

Nous concluons tout de même que ce WCET étant très petit, il est acceptable dans le cadre de notre projet et son impact sera faible sur les performances finales de l'application.

4.6.2 Communication

Mode avec 2 smartphones Dans notre jeu, les performances des communications sont primordiales. Cela est surtout important pour le smartphone "client". En effet, la durée entre le moment où il envoie un déplacement et le moment où il reçoit et affiche les positions du jeux avec ce déplacement pris en compte, ne doit pas être trop longue. Dans le cas contraire, le joueur pourrait ressentir un décalage anormal entre ses mouvements et les déplacements du joueur à l'écran.

L'envoi des déplacements du téléphone "client" se fait sur un thread à part qui est réveillé à chaque fois que la valeur de gravité suivant l'axe X dépasse un certain seuil, en valeur absolue. Au niveau du serveur, une AsyncTask s'occupera de lire ces valeurs et de mettre à jour la position du joueur. C'est sur ce smartphone serveur que le jeu se déroulera en permanence. A nouveau, un thread à part va envoyer à intervalle régulier les positions du jeu (2 joueurs, scores et balles) au smartphone client. L'envoi se fait donc dans une boucle infinie ayant un "sleep()", qui va permettre de définir une fréquence d'envoi.

Après de nombreux tests, nous nous sommes rendu compte que le jeu sur le smartphone client n'était pas fluide en permanence. Le principal problème réside dans l'envoi des données du jeu par le serveur au client. Cet envoi prend parfois beaucoup plus de temps. Voici un graphique représentant l'intervalle de temps en millisecondes entre deux envois de positions :

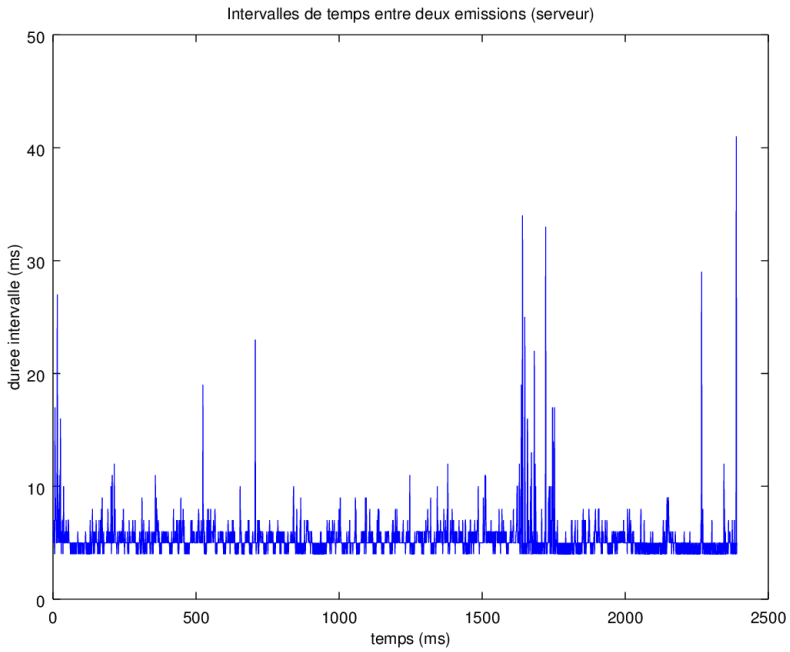


FIGURE 9 – Émissions serveur

On remarque qu'en moyenne les envois sont espacés de 5 ms, ce qui est cohérent puisque nous avions un "sleep()" de 4 ms dans notre boucle d'envoi. Cependant à un moment, les envois ont été beaucoup plus espacés ($> 30ms$). On retrouve le même phénomène en regardant les statistiques réseaux, qui présentent les données envoyées et reçues.

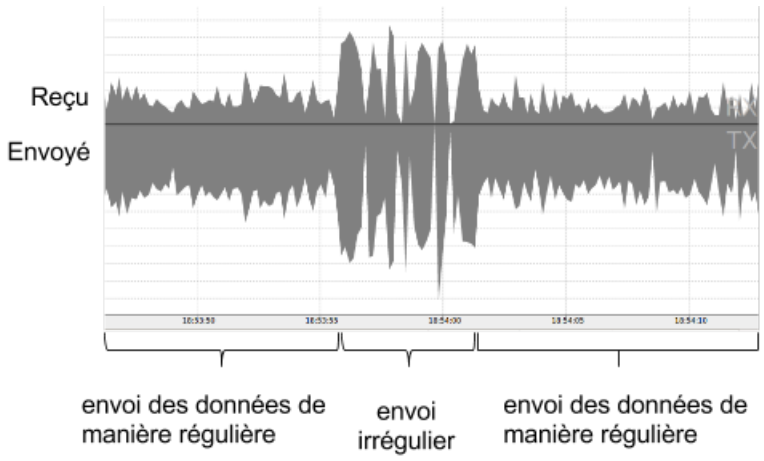
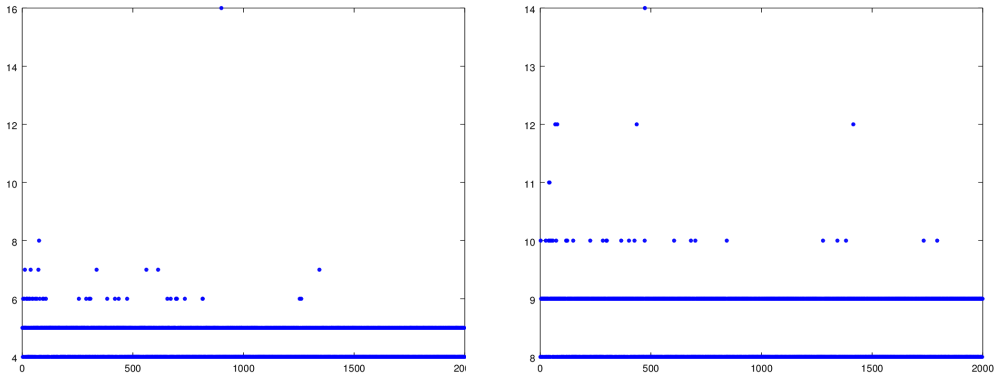


FIGURE 10 – Statistiques réseaux au niveau du serveur

Ces envois irréguliers ont une influence sur la fluidité du jeu sur l'appareil "client". La fluidité reste acceptable mais par moment les déplacements peuvent êtres légèrement "hachés" au niveau du client.

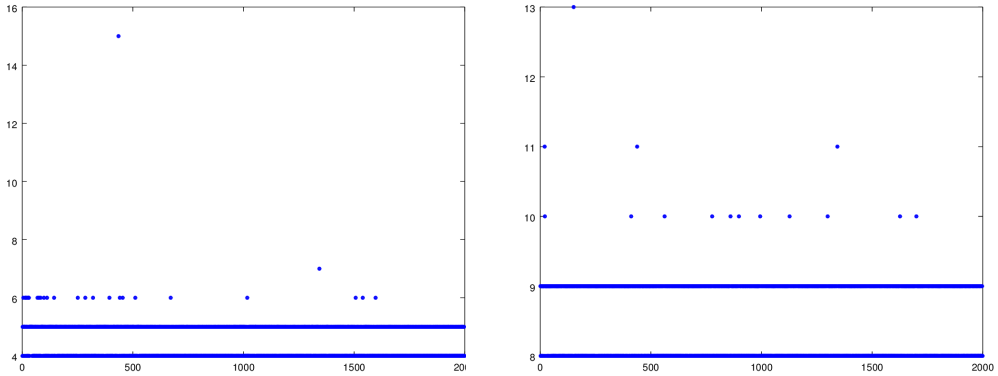
Nous avons donc effectué des tests supplémentaires. Pour cela, nous avons décidé de nous concentrer sur un paramètre : la fréquence d'envoi des données du jeu de la part du serveur. En effet, elle semble être un point central, puisqu'elle va ordonner l'envoi des données. Nous l'avons fixer à deux valeurs 250Hz *(sleep(4ms))* puis 125Hz *(sleep(8ms))*.

Voici les résultats :



(a) f = 250 Hz (b) $f = 125Hz$

FIGURE 11 – Durée (ms) entre 2 émissions du serveur (Wifi Direct)

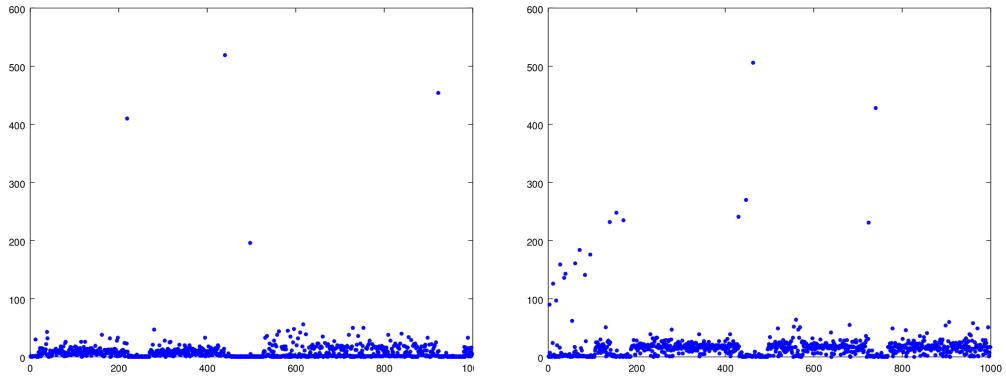


(a) f = 250 Hz (b) $f = 125Hz$

FIGURE 12 – Durée (ms) entre 2 émissions du serveur (Réseau Local)

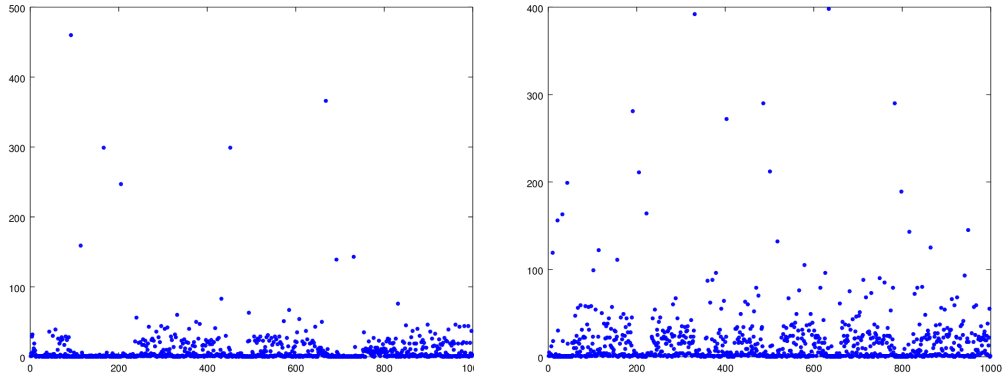
On remarque bien que les résultats sont cohérents. Pour un fréquence de 250 Hz, il est normal d'avoir en moyenne une durée entre 2 envois légèrement supérieure à 4 ms ($1/250 = 4ms$). De même pour une fréquence 125 Hz : 8 ms ($1/125 = 8ms$). On remarque également, qu'il n'y a que très peu de différence entre les valeurs obtenues par Wifi Direct ou par le réseau local. Cependant, on repère le même phénomène que précédemment : il arrive que la durée entre deux envois successif dépasse largement la valeur normale.

Nous nous intéressons désormais à la réception des valeurs au niveau du client :



(a) f = 250 Hz (b) $f = 125Hz$

FIGURE 13 – Durée entre 2 réception du client (Wifi Direct)



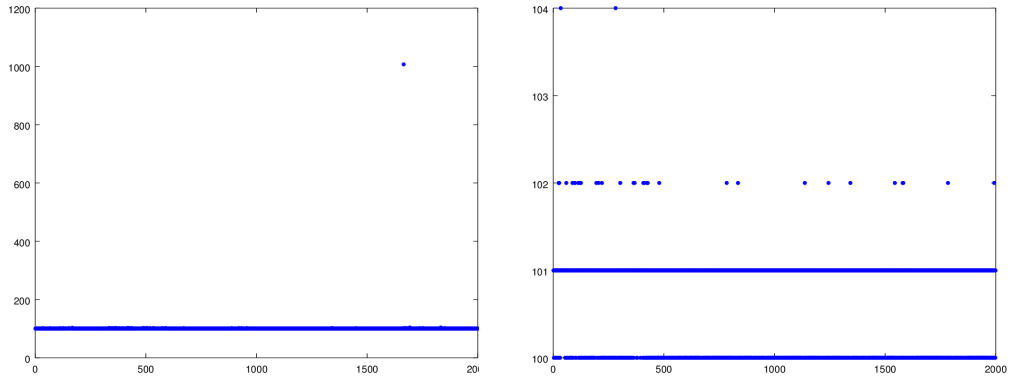
(a) f = 250 Hz (b) $f = 125Hz$

FIGURE 14 – Durée (ms) entre 2 réception du client (Réseau Local)

Ces résultats mettent bien en avant le problème. La majorité du temps la durée entre deux réceptions est correcte, c'est à dire de l'ordre de quelques millisecondes (légèrement supérieur si la fréquence d'envoi est plus faible). Mais à nouveau, à certains moment, cette durée peut atteindre des centaines

de millisecondes. Ce phénomène est directement lié à la sensation de "hachure" sur l'écran du client.

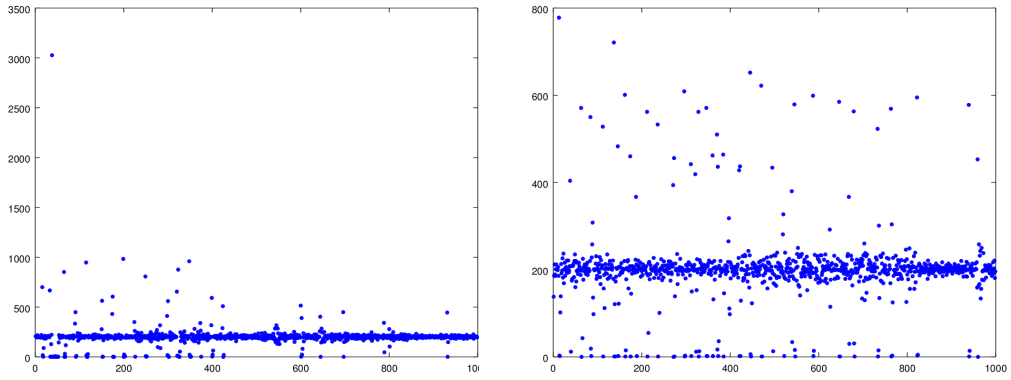
Nous pensons que ce problème peut venir de deux causes principales. Le premier peut concerner la manière d'ordonnancer les tâches de Android. Dans notre application, trois threads sont lancés pendant le fonctionnement du serveur : le thread principal (ou thread UI) responsable notamment de l'affichage, le thread pour recevoir les données du client et le thread d'envoi des données au client. Un mauvais ordonnancement pourrait provoquer des envois différés. Nous avons voulu tester, en réduisant la fréquence d'envoi (10 Hz). Voici les résultats :



(a) Wifi Direct (b) Réseau Local

FIGURE 15 – Durée (ms) entre 2 émissions du serveur (fréquence d'envoi de 10Hz)

Bien que la fréquence d'envoi de 10Hz, ne serait pas raisonnable au niveau de la fluidité, les résultats sont intéressants. On voit que les écarts entre les envois sont vraiment constants. Pour notre test, il ne reste même plus qu'une valeur problématique dans le cas du Wifi Direct. Ainsi, les retards à l'envoi des données sont d'une certaine manière liés à Android et son ordonnancement des threads (Thread UI, thread envoi, thread réception). Voici les résultats pour le même cas, à la réception des données par le client :



(a) Wifi Direct (b) Réseau Local

FIGURE 16 – Durée (ms) entre 2 réceptions du client (fréquence d'envoi de 10Hz)

On remarque que le problème à la réception persiste malgré la fréquence d'envoi. On retrouve toujours quelques durées vraiment importantes entre deux réceptions. Dans le cas du Wifi Direct, les durées entre deux émissions successives étaient très régulières (1 seule valeur absurde), alors qu'au niveau de la réception plusieurs valeurs sont très largement supérieure à la valeur normale. Le transfert des données a donc aussi une grande influence. Dans ce cas, on peut voir par ailleurs une différence entre la version Wifi Direct et celle réseau local. Le nombre de valeurs problématiques est plus important lors de l'utilisation du réseau local. En Wifi-Direct, moins de valeurs posent problèmes mais elles s'éloignent beaucoup plus de la valeur normale :

- Durée > 3000 pour le Wifi-Direct
- Durée < 800 en réseau local

Nous avons également réalisé un programme pour calculer la durée mise par un message pour aller d'un smartphone à l'autre. Pour cela, nous avons envoyé un message du smartphone A au smartphone B avec une date et au retour, nous avons pu déterminer le temps mis par un message pour l'aller-retour smartphone A-B. Ainsi, dans la plupart des cas, nous obtenons des résultats de WCET de quelques dizaines de ms (en général entre 15ms et 20ms). Cependant, à nouveau nous obtenons parfois des WCET beaucoup plus importants pouvant atteindre quelques centaines de millisecondes. On

retrouve donc le même phénomène qu'en analysant la durée entre deux envois ou réceptions successifs. On peut donc conclure, que l'ordonnancement des tâches peut avoir un léger impact sur le respect des durées entre des envois successif, mais que le problème principal vient de la communication par Wifi (réseau local, ou Wifi Direct)

Mode avec 3 smartphones Dans ce mode là, les communications sont simplifiées puisque seuls les appareils "manettes" envoient des déplacements à l'appareil "écran". Le jeu est légèrement plus fluide, mais malgré tout, les performances ne sont pas parfaites, il y a par moment quelques "sauts" pendant l'affichage du jeu. A nouveau, la principale cause vient de la communication par Wifi.

5 Conclusion

Au travers de ce mini-projet, nous avons donc tenté de faire interagir plusieurs smartphones en temps-réel. Nous nous sommes rendu compte que cela posait de nombreuses difficultés. Cela s'explique notamment par la fait que Android n'est pas un OS temps-réel, mais aussi que les communications par Wifi (Wifi-Direct ou réseau local) ne sont vraiment adaptées à ce genre de sujets.

Nous avons pu commencer par spécifier nos exigences, puis notre développement a pu se faire de manière incrémentale en exploitant différentes possibilités. Enfin, une analyse des performances nous aura permis de mieux comprendre l'ensemble des difficultés.