INSA DE LYON - 4BIM

Christophe Pera

Réseau

Pong Game

Alice Fresko, Jacobo Levy Abitbol, Laure Talarmain

Villeurbanne, le 31 janvier 2015

INTRODUCTION

Notre projet consiste à programmer en langage PYTHON, le jeu multijoueur du "Pong game" en réseaux. Pour cela, nous avons utilisé la librairie python KIVY dont l'objectif de base est de créer des applications mobiles sous Android, iOS, Windows Phone mais aussi sur MacOS ou Linux. Nous allons implémenter le jeu en réseau grâce à l'utilisation du protocole TCP moyennant l'utilisation de threads et verrous.

LE JEU ET SES FONCTIONNALITÉS

Le Pong Game est un jeu multijoueur inspiré du tennis de table. Une balle se déplace à travers l'écran, chaque joueur possède une raquette représentée par une barre verticale et un "filet" sépare l'écran en deux. Lorsque la balle rebondit sur le côté de l'écran d'un joueur, l'autre joueur marque un point. Ce jeu se joue normalement à deux, mais nous avons fait le choix de pouvoir y jouer à 4 joueurs réels. Les joueurs réels ne jouent pas l'un contre l'autre mais contre des joueurs fictifs. Le dernier joueur à avoir toucher la balle et à la faire sortir du cadre de jeu gagne un point. Notre écran est séparé en quatre par deux filets. Les raquettes des joueurs fictifs se déplacent de façon aléatoire selon une loi normale centrée sur la position de la balle. Ceci nous permet d'introduire un décalage entre la position de la balle et la position de la raquette, ce qui permet au joueur réel de pouvoir gagner la partie. Le jeu perd en esthétique mais gagne en vitesse d'exécution car sinon on devrait garder en mémoire la position de la balle afin d'introduire un retard entre le déplacement de la raquette et celui de la balle. Chaque joueur réel a sa propre fenêtre de jeu. Le premier joueur qui a atteint le score de 11 points a gagné la partie.

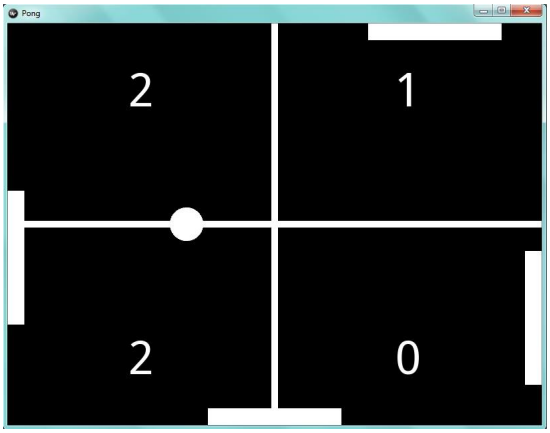

FIGURE 1 – Capture d'écran d'une partie du Pong Game

Chaque partie de la fenêtre correspond au score d'un des joueurs.

ARCHITECTURE LOGICIELLE

3.1 ORGANISATION GÉNÉRALE

Notre programme est composé de trois fichiers python et un fichier kivy :

— PongGame.py

— PongServer.py

— PongClient.py

— pong.kv

Pour lancer une partie, il faut compiler le fichier PongServer.py. Ceci nous permet de générer un serveur permettant l'échange de messages entre les clients ainsi que le transfert d'informations sur la situation du jeu.

Un client rejoint une partie en compilant le fichier PongClient.py. Ceci lance une fenêtre graphique dans laquelle le joueur s'oppose à trois joueurs fictifs dont la position, on rappelle, est actualisée aléatoirement. L'ajout des raquettes horizontales permettrait de développement des améliorations multijoueurs.

Le fichier pong.kv (qui n'a pas pu être commenté) met en forme et donne les positions des widgets de l'interface graphique, comme la balle, les scores et les barres. C'est ce fichier

qui donne les dimensions des widgets et qui définit le nom de ses objets.
Le fichier PongGame.py gère les interactions entre widgets et les interactions entre widgets et joueurs. C'est là que l'on modifie la vitesse des objets, que l'on augmente les scores, que l'on arrête le jeu quand un joueur gagne. C'est là aussi que chaque client qui va se connecter va se voir attribuer les propriétés d'un joueur du Pong Game : sa propre fenêtre de jeu avec les trois barres virtuelles, sa propre barre et son score.
Chaque joueur peut déplacer sa raquette avec les commandes suivantes :

- Raquette verticale à gauche : 'w' (en haut) et 's' (en bas)
- Raquette verticale à droite : les flèches du haut et du bas
- Raquette horizontale en haut : 'd' (à droite) et 'a' (à gauche)
- Raquette horizontale en bas : 'c' (à droite) et 'z' (à gauche)

3.2 Interaction réseaux

Choix du protocole

Nous avons choisi d'utiliser le protocole TCP car il permet de ne pas perdre des données. Le système d'accusé de réception de ce protocole permet aux applications de communiquer de manière sure. Ce que ne permet pas le protocole UDP qui possède un transport non fiable (pertes, duplications de données et des erreurs sont possibles).

Programmation multi-thread

Nous avons choisi d'implémenter un programme multi-thread avec un thread par client. Aucun joueur n'utilise le serveur lui-même pour communiquer avec les autres joueurs. Chaque joueur envoie sa commande (qui lui permet de faire bouger sa raquette) au serveur qui actualise l'affichage graphique (les déplacements peuvent aussi se faire directement sur la fenêtre graphique). De plus, chaque message envoyé par les clients sur le serveur est ré-expédier à l'ensemble des clients hormis l'expéditeur, permettant ainsi de générer un "chat" fonctionnel. Le serveur va permettre à chaque joueur de savoir de qui provient l'information. Tous les joueurs peuvent à n'importe quel moment de la partie et dans n'importe quel ordre, envoyer une commande et recevoir celles des autres. Ceci est possible grâce au fait que chaque joueur (client) possède son propre thread. Cela permet de gérer simultanément la réception et l'émission des informations. Lors de l'émission des messages, l'utilisation de

verrous devient indispensable afin d'ordonner l'emploi des threads.

PROBLÈMES RENCONTRÉS

Nous avons rencontré quelques difficultés au niveau de l'implémentation du jeu. En effet, dans un premier temps, nous avons du nous familiariser avec la librairie Kivy. Nous n'avions pas l'habitude de travailler avec cette bibliothèque, nous avons donc du nous renseigner grâce à des tutoriels.

De plus, nous avons rencontré des difficultés pour implémenter le fait que chaque joueur puisse avoir la fenêtre de jeu en simultané. Par conséquent, nous avons décidé que les joueurs ne joueraient pas les uns contre les autres mais que chaque joueur aurait sa propre fenêtre de jeu et jouerait face à des joueurs fictifs.

Nous avons eu également des difficultés au niveau de la gestion de groupe. En effet, il était difficile de travailler à plusieurs sur un même code. Pour palier cette difficulté, nous avons utilisé le logiciel subversion (SVN) qui permet de collaborer sur un même fichier simultanément.

POUR ALLER PLUS LOIN...

Dans cette partie, nous décrirons les objectifs que l'on aurait souhaité atteindre et les améliorations que nous aurions alors considéré.

Tout d'abord, comme précédemment évoqué, notre intérêt était de créer un jeu Pong multijoueur à quatre joueurs. De plus, une fois que le client se déconnecte il serait remplacé par l'ordinateur qui jouerait à sa place (le code est implémenté mais ne fonctionne pas en accord avec nos attentes).

Nous aurions aussi souhaité qu'après un certain temps fixé, la position des joueurs s'échange aléatoirement entre les raquettes afin de dynamiser le jeu.

Avec un peu plus de temps, nous aurions aimer implémenter une fenêtre d'accueil qui permettrait de choisir le nombre de joueurs. En effet, notre programme a par défaut un nombre de joueur fixé à 4.

De plus, nous aurions aimé que lorsqu'un joueur rejoint une partie, le serveur lui envoie des informations concernant son numéro de joueur ainsi que les commandes qui lui permettent de déplacer la raquette qui lui correspond.
Nous aurions pu rendre l'interface graphique plus attractive et ludique en ajoutant par exemple une couleur spécifique à chaque joueur.
Enfin, nous aurions pu implémenter des avantages pour chaque joueur lors du déroulement du jeu qui leur permettrait de gagner une partie plus facilement.

CONCLUSION

Ce projet nous a permis de programmer le jeu du Pong en réseau. Nous avons pu nous familiariser avec une nouvelle librairie python. Ce projet nous a également permis de s'intéresser à la problématique de la gestion de groupe. Ce projet nous aura permis de nous confronter aux problèmes de la programmation en réseau. Nous avons également pu mettre en exergue nos connaissances acquise lors de notre formation en programmation Python. Cela a été intéressant de réaliser un projet sur un domaine qui ne semble pas directement lié à la bio-informatique.