Chapitre 1

Client Python - Matthieu Dien

1.1 Choix Techniques

Pour ce client, le choix du langage d'implémentation a été Python car c'est un langage haut niveau, simple, avec une librairie standard très fournie et dont une majorité de librairie graphique implémente un binding pour lui. La librairie graphique choisie est wxPython, s'appuyant sur wxWidgets (écrite en C++), elle est libre, portable (disponible à l'ARI), complète (nombreux widgets) et simple d'utilisation. Ce choix Python/wxPython garantit la portabilité du client et facilite un peu sa conception.

1.2 Manuel d'utilisation

1.2.1 Connexion

La connexion se fait en entrant l'adresse du serveur ainsi que son pseudo. Dans le cas où vous choisiriez un mot de passe le client vous demandera si vous êtes déjà inscrit, si non il le fait pour vous (en envoyant "REGISTER/" à la place de "LOGIN/"). Dans le cas où vous seriez déjà inscrit ou que vous vous seriez trompé de mot de passe, le serveur vous refoulera et vous devrez réessayer.

1.2.2 Placement des bateaux

Le placement des bateaux se fait en cliquant sur les cases que vous voulez. Si vous vous trompez, le serveur vous demandera de réessayer.

1.2.3 Phase de jeu

Vous devez attendre votre tour pour vous déplacer. Vous ne pouvez vous déplacer que sur les cases surbrillées en vert et la position actuelle de votre drône est surbrillée en rouge. Vous pouvez continuer à vous déplacer après avoir activé le laser de votre drône s'il vous reste des points d'action. Vous pouvez à tout moment passer la fin de votre tour de jeu en appuyant sur la touche "s" de votre clavier.

1.2.4 Chat

Un chat est à votre disposition pour toute conversation ou tentative de triche avec vos adversaires.

Bon jeu.

1.3 Implémentation

Le client se présente sous la forme d'un fichier *main.py*, de deux dossiers *connection* et *client*, ainsi que de deux fichiers *mybuttons.py* et *mysocket.py*.

1.3.1 Socket

Dans *mysocket.py* on trouve la fonction *readline* qui permet la lecture sur socket. C'est une petite surcouche au socket fournie par la librairie standard python qui ne permettait pas cette lecture ligne à ligne et qui m'a valu pas mal de bug.

1.3.2 Bouton

Dans *mybuttons.py* on trouve une surcouche au "BitmapButton" qui permet la gestion des différents types de case que l'on peut rencontrer durant la partie en utilisant un système de "flag" binaire pour pouvoir ajouter facilement des caractéristiques aux cases. Par exemple, une case peut être avec un sous-marin, touché et rouge (car sous le drône).

1.3.3 Client

Le principal du code se trouve dans le dossier *client*, notamment dans le fichier *controler.py*. On y trouve une classe principale *Controler* qui initialise l'interface graphique (UI) et sert de gestionnaire d'événements pour procéder à tous les traitements graphiques, ainsi que trois autres classes : *WaitPlayers*, *PutShip* et *Action*. Chacune de ces classes hérite de *threading.Thread*, équivalent Java de *Thread*, ce qui permet de ne pas bloquer l'UI lors de l'attente d'événements tels que la réponse du serveur ou l'appui d'un bouton. Ces classes gèrent chacune une phase de jeu d'une partie de "Bataille Navale Royale". La structure globale de ces classes est une boucle qui lit sur le socket à chaque itération et attend, en fonction de la phase de jeu, une ou des commandes particulières du serveur.
Le code contenu dans *view.py* est le code l'UI, il se compose donc de 256 boutons sous forme de quadrillage (la grille de jeu) et d'une zone de chat surmontée par la liste des joueurs présents dans la partie.
Cette partie du code utilise les *threading.Event* fourni par Python, qui permettent une synchronisation entre thread, mais aussi les événements fournis par wxPython, qui permettent une communication entre les threads et l'UI par passage d'objets.

1.3.4 Connection

Le dossier *connection* contient le code de la première fenêtre que vous trouverez en lançant le client. le fichier *connection/view.py* contient l'interface, tandis que le fichier *connection/controler.py* contient la phase de communication avec le serveur et la gestion des événements graphiques.

1.4 Conclusion

L'architecture de mon client et loin d'être parfaite, elle est assez monolithique et donc peu modulable, mais elle a quand même eu un avantage : la gestion du recommencement de partie, le client n'a qu'à tuer la fenêtre de jeu et en relancer une nouvelle. Les extensions demandées et implémentées sont la gestion du "REGISTER/LOGIN/ACCESSDENIED" et le chat.
Mes principales difficultés ont été la connaissance du langage et de la programmation d'interface graphique. J'avais choisi le langage Python malgré le fait que je ne le connaissais que très peu car il me semblait simple et rapide de coder avec. Ce sentiment s'est confirmé. Pour la conception d'interface graphique, cela a été un peu plus délicat car quelques bugs résident encore dans wxPython, mais globalement cela s'est bien passé grâce aux forums utilisateurs.

1.5 Remerciements

Je remercie tous les anonymes (ou pas) qui ont répondu à des questions liées à Python (surtout ses sockets) et à wxPython.