

# Rapport de projet réseau

DELAR Emmanoe, RAKOTOARIJAONA Camille

26 avril 2017



## Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Présentation du jeu . . . . .	3
<b>2</b>	<b>Fonctionnement du programme</b>	<b>3</b>
2.1	Jouer contre la machine . . . . .	3
2.2	Jouer en réseau local . . . . .	3
<b>3</b>	<b>Structure</b>	<b>4</b>
3.1	Serveur . . . . .	4
3.2	Connexion client . . . . .	5
3.3	Protocole de communication . . . . .	5
<b>4</b>	<b>Extensions</b>	<b>5</b>
<b>5</b>	<b>Conclusion</b>	<b>5</b>

# 1 Introduction

Lors de ce projet, réalisé en binômes, nous avons pour objectif de développer un jeu de bataille navale en réseau. Pour cela, nous avons utilisé le langage de programmation objet Python. On est parti du code source fourni par notre enseignant. Ce code nous permettait de jouer contre la machine uniquement. Dès lors, nous avons dû l'améliorer afin de pouvoir jouer à 1 contre 1 sur le réseau.

## 1.1 Présentation du jeu

La bataille navale est un jeu de société dans lequel deux joueurs doivent placer des navires sur une grille tenue secrète et tenter de toucher les navires adverses. Le gagnant est celui qui parvient à torpiller complètement les navires de l'adversaire avant que tous les siens ne le soient.

# 2 Fonctionnement du programme

## 2.1 Jouer contre la machine

Lorsqu'on lance le programme avec le nom du serveur en argument ( : : ou localhost), si on choisi de ne pas jouer en réseau alors le jeu commence directement. Quand le jeu commence, on choisi la colonne et la ligne à viser. Quand c'est au tour de l'ordinateur, une fonction fait de même en choisissant des coordonnées aléatoirement et joue son coup. Et ainsi de suite jusqu'à la fin de la partie.

## 2.2 Jouer en réseau local

Si on choisi de jouer en réseau, comme demandé dans le sujet, il faut dun côté, démarrer le serveur en lançant le programme «main.py» sans arguments, et du côté client, il faut lancer le programme avec ladresse ip du serveur en argument.

Lorsque ceux-ci sont lancés, le client aura la possibilité de choisir d'abord si il veut se connecter au serveur, ou si il veut jouer seul face au robot comme décrit au dessus. Si il décide de jouer en réseau, le programme attendra qu'une autre personne se connecte au serveur pour qu'ils puissent jouer ensemble.

Ensuite, chacun jouera à tour de rôle jusqu'a ce que la partie soit terminée, lorsque la partie est terminée chaque joueur verra le score, et le serveur se déconnectera et le jeu de chaque joueur aussi.

Il est à noter que les joueurs peuvent donc jouer tout seul peu importe le moment sans affecter le serveur lorsqu'ils jouent en local.

## 3 Structure

Contrairement à la partie contre l'ordinateur, en mode réseau, les coordonnées choisies par le joueur client doivent être envoyés au serveur pour qu'il puisse les retransmettre à l'adversaire. Nous avons alors créé un protocole de communication serveur/client TCP sur le port 7777. Ce port nous permettra de recevoir ou d'émettre des informations.

### 3.1 Serveur

La programmation du serveur à été un peu plus complexe car il a pour tâche de faire le lien entre les joueurs.

Au lancement du programme, nous nous retrouvons dans la fonction principale «main » en attente de demande de connexion, tout cela se fait par la méthode «.select» vu en cours. Lorsque 2 joueurs se connectent, le serveur leur envoie à l'aide de la méthode «socket.send », leur numéro, 0 ou 1 selon leur ordre de connexion. Ce numéro définit celui d'entre eux qui jouera le premier.

La deuxième étape, nous a demandé beaucoup plus de réflexion car nous avons choisi d'envoyer le jeu créé par le serveur, aux clients connectés.

En effet, créer un jeu aléatoire depuis le serveur puis l'envoyer aux clients nous offre beaucoup de possibilités. Par exemple implémenter l'extension qui nous permet d'avoir des spectateurs devient plus simple car nous pouvons donc leur envoyer ce même jeu.

Pour parvenir à cela, nous avons dû modifier la fonction «randomConfiguration» qui génère aléatoirement une table de bateaux.

Dans cette fonction, tout se passe normalement, puis à chaque fois que l'on génère les coordonnées et la position d'un navire, on stocke ces informations dans un tableau. Une fois la génération terminée, on envoie le contenu du tableau, bit à bit (en utilisant la méthode «client.send » ), aux joueurs/spectateurs connectés avec une boucle qui parcourt le tableau.

Cette partie nous a posé plusieurs problèmes, par exemple, au niveau de la taille des bateaux, les valeurs envoyées devaient être de type «bytes» . La taille de ces variables pouvait varier, de 1 à 2 bytes car les valeurs générées par la fonction «randomConfiguration» allaient de 1 à 10 et 10 est représenté sur 2 bytes. On a donc décidé de décaler de 1 chaque valeur pour avoir un intervalle de 0 à 9 et ainsi avoir une taille au maximum de 1 byte pour chaque valeurs envoyées. La valeur reçue puis convertie en entier de l'autre côté par le(s) joueur/spectateur(s) serait incrémentée de un.

Le deuxième problème était aussi un peu complexe, mais après débogage, on a pu voir que l'on envoyait avant directement chaque valeur aléatoire sans avoir au préalable vérifié si on avait une configuration valide travers la fonction isValidConfiguration . On a alors compris qu'il fallait prendre les dernières valeurs de configuration valide et envoyer le tout à chaque joueurs après.

Et enfin comme la fonction principale, celle-ci retourne un bateau. On répète alors ceci 2 fois et pour avoir la table de jeu. Ensuite, c'est le début de la partie, la boucle qui fait jouer le serveur est la même que celle qui fait jouer le robot ou

chaque joueur, cependant, celui-ci se place d'abord dans la partie du premier joueur de la boucle, et attendra que le joueur envoie son coup, il mettra le coup sur sa table de jeu et l'enverra l'autre joueur, tout cela avec la méthode `recv` et `send`. Et passera ensuite la partie du joueur 2 de la boucle et fera de même. Et ainsi de suite jusqu'à ce que le jeu soit fini. Ensuite il sort de la boucle et ferme les socket et la fonction et le serveur.

### 3.2 Connexion client

Pour que le client se connecte au serveur, il nous faut le nom de l'hôte et le numéro de port.

Ensuite, on va créer une socket (client) de type `ipv6` qui va nous permettre d'ouvrir une connexion avec une machine locale et d'échanger des informations (à l'aide de la méthode `«bind»` ).

Pour récupérer les premières informations, notamment le numéro du joueur, les navires, on intercepte les premiers bytes envoyés par la socket serveur (à l'aide de la méthode `«client.recv»` ).

Avec ces informations, on génère une partie à partir d'une fonction `Game`.

Une fois la partie créée puis lancée, si c'est au joueur actuel de jouer, il entre les coordonnées de la grille adverse à viser. Ensuite, on va envoyer ces coordonnées (x,y) sous forme de paquets de bytes au serveur. Le serveur va retransmettre ces paquets de bytes à l'autre joueur (à l'aide de la méthode `«client.send»` ).

Par contre si c'est au tour de l'autre joueur, la socket client se met en mode réception et attend les coordonnées choisies par l'adversaire (à l'aide de la méthode `«client.recv»` ). Ces données sont également envoyées par le serveur.

Les données reçues sont converties en coordonnées de type entier puis passent en arguments de la fonction `«addShot »` . Cette fonction tente de viser la flotte ennemie à ces coordonnées. Si le tir est juste, l'adversaire voit la case correspondante barrée, sinon la case est entourée pour représenter un échec.

Ce protocole est répété jusqu'à la fin de la partie.

### 3.3 Protocole de communication

Une fois la connexion établie, la communication serveur/client est très simple. Chaque information nécessaire à l'initialisation d'une partie est envoyée par le serveur bit à bit et ainsi récupérée du côté client. Il faut noter la nécessité de convertir les bytes reçus en entier (`int`).

## 4 Extensions

## 5 Conclusion