Rapport projet IN104

La Bataille Navale

JOURNE Arthur

CADART Victor

Introduction

Notre projet informatique porte sur l’implémentation d’un jeu de bataille navale, selon les règles classiques de ce jeu. Ce jeu oppose deux joueurs qui s’affrontent, en posant sur leur grille cinq bateaux. Le but est de détruire les bateaux de son adversaire, le jeu s’effectuant en tour par tour, chaque jour ciblant une case sur la grille numérotée de A à J horizontalement et de 1 à 10 verticalement (grille 10x10 cases).

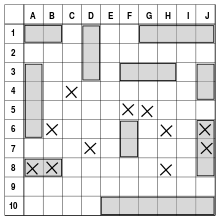


Figure 1 : Grille de jeu

Travail demandé

Il nous était demandé pour ce projet d’implémenter le jeu avec le langage Python, avec la possibilité de jouer en réseau contre un autre joueur ou bien de jouer contre une IA développée par nos soins. Une interface graphique était également demandée pour rendre le jeu plus agréable. Le développement réseau devait être effectué avec la librairie ZeroMQ de Python, tandis que l’interface graphique devait être réalisée à l’aide de Tkinter

Travail effectué

La mise en réseau ainsi que l’IA ont été finalisé et sont opérationnels. L’interface graphique n’a pas été implémentée.

Concernant la mise en réseau : l’interface est constituée de deux fichiers, un partie client et une partie serveur. Lors de sa mise en route, le serveur ouvre un port P1 sur lequel le client se connecte automatiquement et envoie une requête signalant sa connexion. Le serveur recevant cette première connexion demande alors le mode de jeu souhaité par le joueur (Joueur contre Joueur ou bien Joueur contre IA) puis son nom.

* Si le joueur choisit de jouer contre l’IA, le joueur doit lancer le fichier IA adapté (facile ou difficile) et l’IA se connecte en tant que joueur n°2. Le serveur établit un proxy entre le joueur et l’IA, et la partie commence. Le joueur est toujours le premier à jouer.
* Si le joueur choisit de jouer contre un autre joueur, il affiche au joueur 1 qu’il se met en attente d’autre joueur, ouvre son port P2 et attends qu’un autre joueur se connecte. Lorsque le joueur 2 se connecte, il lance le fichier client.py. Tentant de se connecter au port 1, le serveur lui indique qu’il y a déjà un joueur 1 et qu’il doit se connecter au port 2. Le joueur 2 se connecte donc ensuite automatiquement au port 2, le serveur lui demande son nom, et lorsque le joueur répond, le serveur établit un proxy entre les deux joueurs et la partie commence. Le joueur 1 est toujours le premier joueur à commencer.

Le proxy permet de faire circuler tout envoi du joueur 1 directement au joueur 2 et vice-versa et donc facilite la communication. Il permet également de détecter si un troisième joueur se connecte. En effet, si un joueur se connecte alors que le proxy est déjà établit (donc quand la partie est lancée), il reçoit son propre message. Ceci permet donc d’évincer ce troisième joueur, qui saura que deux joueurs sont déjà connectés.

La difficulté de la mise en réseau via ZeroMQ consiste à gérer les dialogues entre deux interlocuteurs. Il ne faut pas que les deux parlent en même temps, sinon les messages ne sont pas reçus, et il ne faut pas que les deux interlocuteurs attendent un message simultanément, sinon ils se retrouvent bloqués en mode « attente ». C’est pourquoi le joueur 1 est de type REQ : il envoie un message puis se met en réception, tandis que le joueur 2 (physique ou IA) est de type REP : il attend un message puis peut en envoyer un. Cette difficulté a posé des problèmes dans la gestion des messages envoyés ou reçu par le serveur. En effet, le serveur doit se mettre en réception de message pour le joueur 1 (car le joueur 1 est REQ) mais doit envoyer les messages au joueur 2 (car le joueur 2 est REP), et il fallait qu’après le dialogue avec le serveur, le joueur 1 soit en envoie de message tandis que le joueur 2 soit en attente de message. La graphique ci-dessous synthétise la connexion entre les deux joueurs (le temps va de haut en bas) :

|  |  |  |
| --- | --- | --- |
| Joueur 1 | Joueur 2 | Serveur |
|  |  | Ouvre le port P1 |
|  |  | Attends un message sur P1 |
| Connecte sur P1 |  |  |
| Envoi de "Hello i want to play" |  |  |
|  |  | Reçoit "Hello i want to play" sur P1 |
|  |  | Demande du mode au J1 |
| Entre le mode |  |  |
|  |  | Demande le nom |
| Entre son pseudo |  |  |
| Se mets en attente |  | Ouvre le port P2 |
|  |  | Attends un message sur P1 |
|  | Connecte sur P1 |  |
|  | Envoie "Hello i want to play" au P1 |  |
|  |  | P1 déjà occupé, envoit à J2 de se connecter à P2 |
|  | Reçoit P1 déjà occupé, connexion à P2 |  |
|  |  | Demande le nom de J2 |
|  | Entre son nom |  |
|  |  | Envoie "début de la partie" aux deux joueurs |
| Partie commence | Partie commence | Etablissement du proxy |

Concernant le déroulement d’une partie : Le joueur 1 est toujours le premier à jouer (du fait qu’il soit REQ). Un joueur peut tirer deux fois au même endroit s’il n’est pas attentif et il est averti s’il a touché -ou coulé- un bateau adverse. Lorsque tous les bateaux d’un des deux joueurs sont coulés, la partie se finit et les deux joueurs se voient déconnectés (et avertis de leur succès ou de leur défaite)

Concernant l’implémentation du jeu : les fonctions principales sont dans le fichier client.py. Ces fonctions définissent les actions du jeu (placer les bateaux, tirer). Des fonctions auxiliaires ont été mises en place dans le fichier aux.py. Ces fonctions transforment notamment les entrées de l’utilisateur en données exploitables sur la grille du jeu (Transformation de chaîne de caractère en entier).

Concernant la partie IA : deux modes de jeu ont été implémentés, une IA dite « facile » et une IA dite « difficile »

L’IA facile place aléatoirement ses bateaux et tire aléatoirement sur le plateau. Elle ne réagit en fonction des coups qu’elle a fait précédemment : si elle touche un navire, elle ne va pas spécialement tenter de le couler. Cependant, elle ne peut tirer deux fois au même endroit, car elle tire sur la grille restreinte des possibilités (Grille de jeu privé des tirs déjà effectués)

L’IA difficile place également aléatoirement ses bateaux, mais tire selon une stratégie. Chaque case est pondérée et voit sa valeur changer au cours de la partie en fonction des résultats obtenus : si une case affiche « touchée », alors les cases alentours voient leur valeur augmenter, et seront donc plus susceptibles d’être attaquées au tour suivant.

Travail à améliorer

Le serveur actuel est un simple pont entre les deux joueurs (physiques ou IA). Par conséquent, il ne gère pas les déconnections des joueurs. Autrement dit, si un joueur se déconnecte, le second joueur n’est pas averti et se voit attendre que l’autre joueur joue son tour. De plus, le code initial qui empêche la connexion d’un troisième joueur empêche la reconnexion du second en cas de problème.

Il peut également y avoir des soucis théoriques en cas de connexion d’un troisième joueur durant la phase de préparation (c’est-à-dire avant que le serveur envoie « la partie commence »), mais que l’on observe que dans des cas spécifiques (connexion du troisième joueur à une étape précise)

Enfin, lorsque la partie se finit, il faut fermer manuellement le serveur afin de pouvoir relancer une autre partie.

Du côté de l’IA, il faudrait coder une IA qui réagirait plus comme un humain : si elle touche un navire, elle tire sur les cases adjacentes. Concernant l’IA difficile, un problème de lien entre l’IA et le serveur empêche celle-ci de fonctionner. Il faudrait revoir la structure du code.