

Projet de Réseau

JEU DU MORPION AVEUGLE

10 janvier 2017

Université de Bordeaux

Kenji FONTAINE
Enzo PERUZZETTO

Table des matières

1	Présentation du projet	2
2	Lancement d'une partie	2
3	Fonctionnalités des fichiers	3
3.1	client.py	3
3.2	serveur.py	3
3.3	jeu.py	3
3.4	macro.py	3
3.5	grid.py	3
4	Difficultés rencontrées	4
5	Extensions	5
6	Difficultés rencontrées aux extensions	6
7	Conclusion	6

1 Présentation du projet

Ce travail a été réalisé dans un cadre universitaire par des étudiants en Licence informatique.

Le jeu de morpion est un jeu à 2 joueurs se déroulant sur une grille de 3 cases sur 3. Chaque joueur à tour de rôle marque une case dans la grille. Le premier joueur parvenant à aligner 3 de ses marques est déclaré gagnant. Si la grille est complètement remplie sans qu'il n'y ait 3 marques identiques alignées, il y a match nul.

Le jeu du morpion aveugle est une variante du jeu de morpion dans laquelle les joueurs ne voient pas les coups joués par leur adversaire respectif. Si un joueur essaie de jouer une case qui a déjà été jouée, il est informé que cette case est prise par son adversaire et doit en jouer une autre.

Le but de ce projet est d'implémenter ce jeu du morpion aveugle en réseau.

2 Lancement d'une partie

Nous avons besoin de 3 terminaux pour lancer une partie, 1 serveur et 2 clients. Se placer dans le dossier src et lancer les commandes suivantes :

- python3 serveur.py
- python3 client.py 8888
- python3 client.py 8888

Le fichier client.py prend en paramètre le port du serveur. Il est codé en dur dans le fichier serveur.py à la ligne 14 et peut être modifié.

Une fois la partie terminée, le serveur fermera et les clients s'arrêteront.

3 Fonctionnalités des fichiers

3.1 client.py

Il s'agit du fichier exécuté par le client se connectant au serveur pour jouer une partie. Il prend en paramètre le port du serveur en question. Le client crée un socket pour se connecter au serveur puis lance une boucle infinie dans laquelle il va recevoir les informations envoyées par le serveur. Les informations reçues sont des octets qui ont été générées en utilisant pickle côté serveur.

Le client réutilise pickle pour retransformer les octets afin d'être utilisables. La fonction `handle` est alors appelée, elle a un comportement différent en fonction du code porté par les données retransformées.

La structure des données envoyées par le serveur est expliquée plus en détails dans la section dédiée au fichier *macro.py*.

3.2 serveur.py

Il s'agit du fichier exécuté par le client hébergeant la partie du morpion aveugle. Le serveur commence par créer un socket et dans une boucle infinie, accepte les connexions des utilisateurs. Une fois 2 utilisateurs connectés, le serveur lance la partie en appelant la fonction `main` du fichier *jeu.py*.

3.3 jeu.py

Ce fichier permet au serveur de lancer une partie du morpion aveugle. Grandement inspiré du fichier *main.py* qui était fournis, il a été modifié afin de gérer 2 joueurs et les différents envois d'information faits par le serveur.

Le détail relatif aux informations envoyées par le serveur est donné dans la section suivante.

3.4 macro.py

Ce fichier contient les différentes informations que le serveur envoie à ses clients. On y trouve la définition de plusieurs macros ainsi que de leur "message" respectif.

Les données envoyées sont stockées dans un tableau de la forme suivante : `[code, données]`.

Où le code est un entier qui va déterminer le comportement de la fonction `handle` du client (ligne 12, *client.py*) et les données seront à traiter par le client. Le serveur va ensuite utiliser pickle pour transformer les tableaux en octets afin de pouvoir les envoyer au travers des sockets.

Les macros de la forme `MSG_XXXXX` sont les octets générés par pickle, ceux-ci permettent d'alléger le code du serveur.

3.5 grid.py

Ce fichier permet d'afficher, de modifier la grille utilisée par un jeu du morpion. Il y a eu quelques petites modifications, telles que le renommage de la classe `grid` (`grid` -> `Grid`) ou la modification du tableau des symboles pour mieux coller avec notre code.

4 Difficultés rencontrées

Au début, nous pensions tout faire faire par le serveur. Héberger la partie, gérer les coups, modifier les grilles. Le serveur ferait alors le rendu des grilles (fonction `display()`) et envoyaient les grilles aux joueurs sous forme de chaîne de caractères. Le client devait juste taper le numéro de la case qu'il souhaitait jouer, quand il lui était demandé. Cependant, il était dit dans le sujet que le client devait faire le rendu de la grille, donc nous avons abandonner cette idée.

Dans une seconde version, notre serveur communiquait avec le client en envoyant uniquement des entiers, sans utiliser pickle. Ces entiers modifiaient le comportement de la fonction `handle` du client. Le code était comparable à celui de la version actuelle. La différence majeure étant qu'au lieu d'envoyer des tableaux en utilisant pickle, le serveur envoyait des entiers convertis en octets en faisant des cast. Le fichier *macro.py* contenait par exemple des lignes de cette forme ;

```
[...]
WELCOME = 10
[...]
MSG_WELCOME = bytes(str(WELCOME).encode('utf'))
[...]
```

Tout marchait bien, sauf l'envoi de grille du serveur au client. L'implémentation n'était pas stable et l'envoi de grille échouait 2 fois sur 3. Nous n'avons pas réussi à identifier la source du problème et avons finalement abandonner cette version.

Dans notre troisième version, qui est la version actuelle, l'objectif était de reprendre la seconde version et de changer la façon dont le serveur communiquait avec les clients. Il fallait qu'on puisse envoyer un entier ainsi que des données pour pouvoir faire l'envoi de grille. C'est là que nous avons utiliser pickle qui permettait de simplement convertir des tableaux en octets ou des octets en tableaux.

5 Extensions

Partie seul

Si on lance le fichier `client.py` sans paramètre nous pouvons jouer seul contre une IA jouant des coups aléatoires. Il s'agit juste d'un appel à la fonction permettant de jouer seul qui est fournie. Elle se trouve dans le fichier `main.py`.

Initiative aléatoire

Le joueur ayant la main est déterminée de façon aléatoire par le serveur. Le joueur 1 a autant de chance de commencer que le joueur 2. Nous avons utilisé la fonction `random.randint(a, b)`. Cette fonction renvoie un entier généré aléatoirement, compris entre `a` et `b`. Cette fonction est appelée à la ligne 31 du fichier `jeu.py`.

Fonction rejouer

Lorsqu'une partie prend fin (victoire/défaite ou égalité), il est demandé aux clients si ils désirent ou non rejouer une partie. Si les deux joueurs répondent affirmativement, une nouvelle partie se lance. Sinon, le serveur se ferme et les clients s'arrêtent.

Mode spectateur

Si un troisième client se connecte au port du serveur, il devient automatiquement spectateur. Il reçoit alors la grille complète du jeu à chaque fois qu'un coup est joué. Cette extensions nous a posé beaucoup de problème et n'est pas entièrement fonctionnelle. Les détails sont dans la section suivante. Une ébauche du mode spectateur est visible en lançant le fichier `test.py` et en connectant 3 clients. Les deux premiers seront joueurs et le troisième sera spectateur.

```
python3 test.py
python3 client.py 8888 (3 fois)
```

6 Difficultés rencontrées aux extensions

Mode spectateur

Nous avons tenté d'implémenter cette fonctionnalité à plusieurs reprises. Dans notre seconde version (seconde version mentionnée dans la section **difficultés rencontrées**), l'envoi de grille était peu fiable et ajouter un spectateur qui devait recevoir une grille ne faisait qu'aggraver la situation. La tentative d'implémentation de spectateurs nous a montré les limites de notre seconde version. Ce qui nous a poussé à changer la structure de notre code.

Dans notre version finale, nous avons amélioré la stabilité ainsi que la fiabilité de l'échange de grille. Nous avons réussi à obtenir un mode spectateur qui marche à peu près comme souhaité, excepté le fait qu'il affiche bien plus de grilles que prévu.

Gestion des déconnexions (envisagée)

Nous pensions implémenter une fonctionnalité permettant de gérer la déconnexion de l'un ou des deux joueurs. Une solution envisagée était de vérifier la taille des données envoyées par le client lorsque celui-ci était demandé de jouer une des cases de la grille. Si les données reçues étaient vides, cela signifierait que le client a rencontré un problème. Dans ce cas-là, nous aurions pu fermer le serveur et mettre fin à l'autre client encore connecté.

7 Conclusion

Le projet s'est révélé être plus compliqué que ce que nous pensions. L'échange de données entre les clients et le serveur n'est pas une chose facile à gérer. Le comportement de la boucle infinie du serveur n'a pas été entièrement compris, ce qui nous a posé problème à plusieurs reprises. Nous avons pu approfondir nos connaissances en langage Python au travers de ce projet.

Cependant, nous pensons que la fin de ce semestre était relativement mal organisée. En effet, le semestre était nettement plus chargé à la fin qu'au début. Avec plusieurs projets en parallèle ainsi que les examens en même temps, s'organiser n'était pas chose simple et nous aurions aimé avoir un peu plus de temps.