Rapport de réalisation - Projet Morpion en réseau

Important

Pour lancer le projet, il est nécessaire de se rendre dans le répertoire Src et de lancer le script main.py à l'aide de la commande:

```
python3 main.py [adresse serveur]
```

Lancé sans argument, le script lance un serveur. Si 'adresse serveur' est précisé, le script lance un client et tente de se connecter au serveur situé à l'adresse 'adresse serveur'.

Il est important de noter que le projet ne fonctionne qu'à partir de python 3.5 ! Il s'agit normalement de la version de python 3 installée sur les machines de l'Université.

I. Sujet/objectif du projet

L'objectif du projet est de réaliser en binômes un jeu de morpion aveugle en réseau. Le projet doit se concrétiser sous la forme d'un logiciel python présentant deux modes de fonctionnement:

- Si l'utilisateur lance le programme sans argument, le logiciel s'exécute en mode serveur.
- Si l'utilisateur précise une adresse ip en argument, le logiciel se lance en mode client.

II. Lexique

Terme	Définition
Client	ici, logiciel permettant à un utilisateur de se connecter à un serveur et d'interragir avec ce dernier, afin de participer à des parties de morpion.
Serveur	Logiciel chargé de mettre en comunication plusieurs clients et de gérer le déroulement de la partie.
Joueur	Client disposant de la possibilité de jouer au jeu du morpion sur un serveur (2 joueurs par serveur).
Observateur	Client ne pouvant jouer au jeu du morpion sur un serveur. A la place, il reçoit l'état de la partie à chaque fois qu'un joueur joue un coup (une infinité d'observateurs par serveur).

III. Règles de jeu

Chaque joueur joue tour à tour. Durant son tour de jeu, un client peut choisir de poser un jeton sur la grille ou de quitter la partie en entrant 'exit' dans l'invité de commande.
Si le joueur pose un pion:

- Si le joueur peut jouer sur la case, alors un pion sera placé et se sera au tour du joueur suivant.
- Sinon la case n'est pas disponible car l'autre joueur l'occupe deja. Dans ce cas, le joueur découvre le pion adverse qui devient visible sur sa grille, et le joueur passe son tour.

Si un joueur arrive a placer 3 pions en ligne, colonne ou diagonale, il est déclaré vainqueur. L'autre joueur est donc perdant. Une fois la victoire annoncée, les joueurs et observateurs peuvent essayer de se reconnecter au serveur. Premier connecté, premier servi, les deux premiers clients connectés seront joueurs.

IV. Description des modes de fonctionnement

Mode Serveur

En mode serveur, le logiciel propose un point d'ancrage permettant à deux clients de jouer au morpion en ligne, ou d'observer deux joueurs s'affronter.

Mode client

En mode client, le logiciel peut se connecter à un serveur. Ce dernier transmet au client son status (joueur/observateur). Si il est joueur et que le nombre de joueur connecté au serveur est suffisant, le client peut transmettre des coups au serveur qui l'informe de l'état de la partie, donne la main à l'autre joueur et ainsi de suite. Si il est observateur le client recoit l'état de la partie à chaque coup placé par les joueurs.

Si le joueur quitte le jeu (en tapant exit, ou pour toute autre raison), un compte-à-rebours est lancé. Le joueur a trois minute pour se reconnecter sinon, le serveur sera relancé afin de permettre à d'autres clients de jouer.

IV. Fonctionalité

Fonctionalités implémentées

Intitulé	Description
Connexion de participants/Gestion joueur/observateur	Si le nombre de joueurs est inférieur à 2, le nouveau client connecté est considéré comme joueur. Tant que le nombre de joueur est inférieur à 2, le joueur connecté est en attente. Si deux joueurs sont connecté la partie commence. Tout les clients qui se connecterons plus tard seront considérés comme observateurs.

Déroulement du jeu	Les clients considérés comme joueur peuvent communiquer au serveur les coups qu'ils souhaitent jouer à tour de rôle jusqu'à la fin de la partie. Lorsque la partie est terminée, les clients observateurs sont notifiés de la fin de la partie et les clients joueurs sont mis au courant de leur situation (gagnant/perdant).
Possibilité de jouer plusieurs parties	Lorsqu'une partie est terminée, tout les status sont remis à zéro. Le serveur attend de nouveaux paquets 'CONNECT' de la part des clients connectés. Les deux premiers reçus seront considérés comme joueurs. En revanche, les scores ne sont pas comptés étant donné que les status joueur/observateur changent à chaque partie. Nous ne considérions pas la conservation de score comme pertinente.
Gestion de l'interruption de la connexion entre les clients et le serveur (Crash ou déconnexion)	Si le client déconnecté (pour quelque raison que ce soit) est un des deux joueurs, le serveur attend un certain temps la reconnexion du joueur. Si il ne se reconnecte pas, la partie est relancée et tout les clients sont expulsés du serveur. Si le client est un simple observateur, le serveur ne réalise aucune action autre que libérer ses données.

Fonctionalités non-implémentées

Intitulé	Description
Robot jouant aléatoirement des coups	Etant donné l'architecture de notre projet, nous avons jugé que la modification du code pour permettre l'ajout d'un robot aurait excessivement complexifié le code en générant des exceptions de traitement (cas particuliers).

V. Informations techniques et organisationnelles

V.II Le partage des tâches et organisation de travail

L'ensemble du projet pouvant facilement se découper en deux grand sous-ensembles client et serveur, le partage des tâche a été relativement simple à mettre en place. Ainsi Célia Paqué a été chargée de développer le client, tandis que Clovis Portron devait produire un serveur fonctionnel.

Afin de permettre un développement parallèle et un bon environnement de développement, le logiciel de gestion de version Git a été utilisé. Le dépôt distant commun aux participants étant hébergé par la plateforme Github.

V.I Le protocole

Afin de permettre un développement simultanné du client et du serveur, le choix a été rapidement fait de commencer par établir un protocole de communication commun au client et au serveur. Ce protocole est visible ci-dessous.
Concrétement, dans l'implémentation du code source, chacune des commandes ci-dessous prend la forme d'un 'paquet' (Voir plus bas).

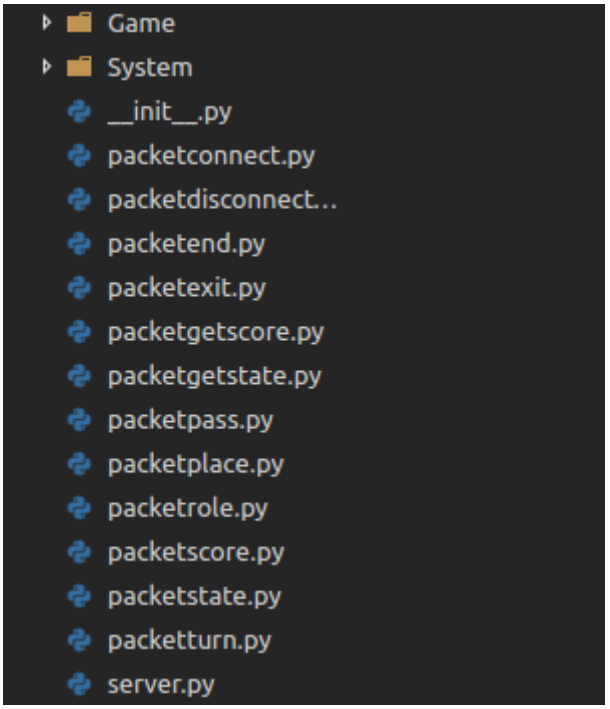
Commande client au serveur	Arguments	Effet	Réponse	Action joueur
CONNECT	NULL	Entraine initialisation du joueur coté serveur	ROLE si réussi ou NOP si erreur	NON
GETSTATE	NULL	Récupération de l'état de la partie en cours	STATE 5,8,1 "La case 5 < joueur 1, la case 8 < joueur 2, la case 1 < joueur 1	NON
GETSCORE	NULL	Récupération du score des joueurs	SCORE a,b "a : score joueur 1, b : score joueur 2"	NON
EXIT	NULL	Le joueur se déconnecte	OK	OUI
PLACE	a "a: numéro de la case"	Place le jeton sauf si case déja occupée	OK / PASS / NOP "OK: si placé, PASS: si le client doit passer son tour, NOP: si erreur"	OUI

Commande serveur au client	Arguments	Effet
OK	NULL	Acquittement de la dernière commande
NOP	NULL	Refus de la dernière commande
PASS	NULL	Le client doit passer la main à son adversaire
STATE	a,b,c....n	Retourne la liste des coups joués, impair: joueur 1, pair: joueur 2
SCORE	a,b	a: score joueur 1, b: score joueur 2
DISCONNECTED	ip	ip: Ip du deconnecté. L'autre joueur s'est déconnecté. Si on est joueur, on doit attendre.
SHUTDOWN	NULL	Le serveur s'est arrêté
TURN	NULL	C'est a ton tour de jouer (faire getState et autoriser placement)
ROLE	observer ou player	observer si observateur, player si joueur. Indique au client son role
END	win ou loose ou rien	win si le joueur a gagné, loose sinon, ou rien si envoye a un observer. Indique la fin d'une partie

(A noter que dans le produit fini, Score n'est pas implémenté coté client).

V.II Organisation générale du code

Coté client et serveur, le code est organisé en trois grand sous-ensembles:

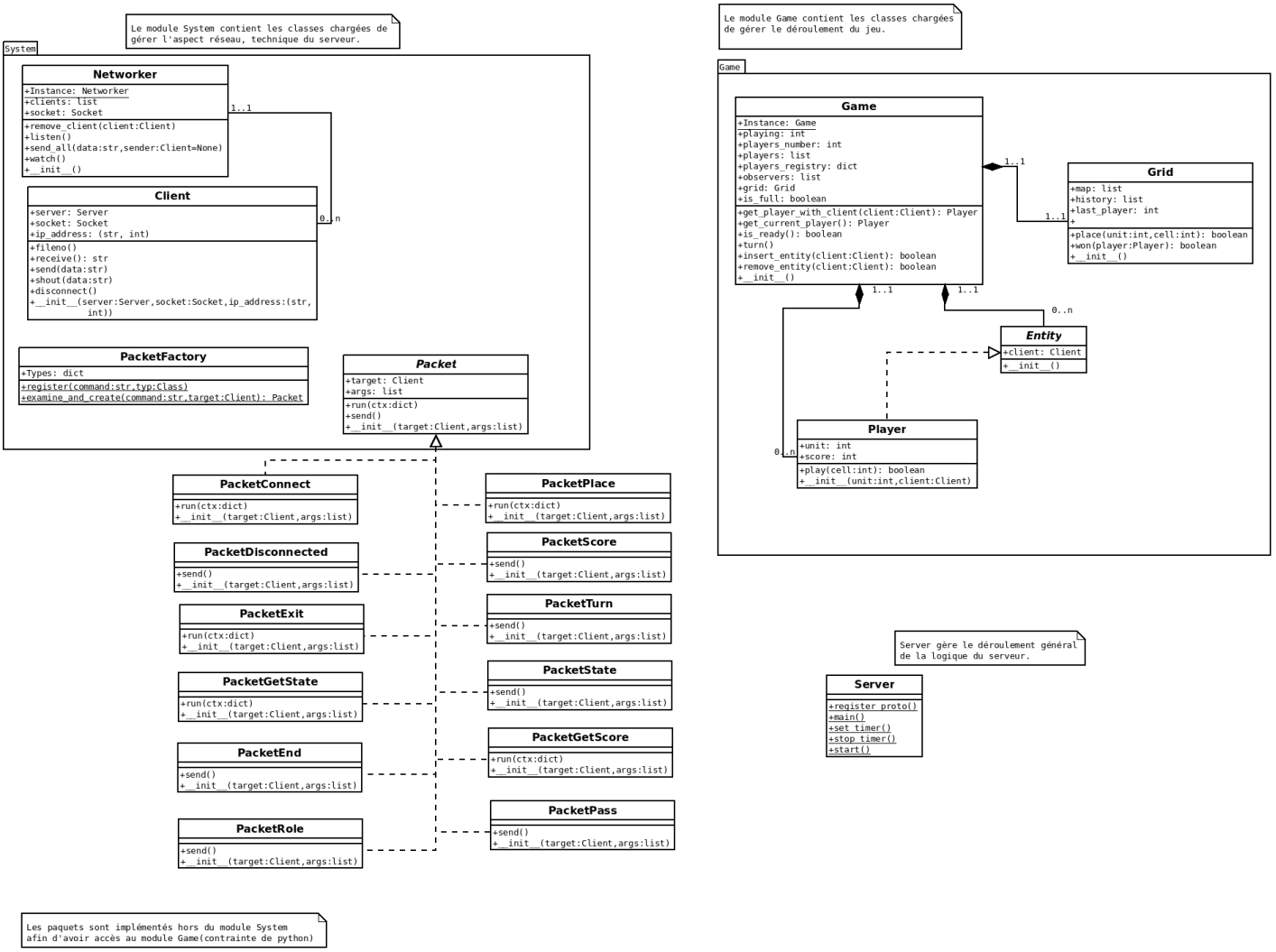


Le module Game contient l'ensemble du code gérant la logique de jeu. Le module System contient les ressources de base nécessaire au bon fonctionnement du code en réseau. L'ensemble des fichier 'packet' correspondent chacun à une commande du protocole. Une routine principale attend de manière indéfinie des données depuis le serveur ou le client. Lorsque la commande qui leur est associée est reçue par le logiciel, une nouvel instance du paquet correspondant est générée, c'est ensuite cette dernière qui va prendre en main le déroulement de la logique logiciel. Une fois sa tâche terminée, le paquet rend la main à la routine principale du logiciel qui va attendre une nouvelle réception de paquet et ainsi de suite.

D'une manière générale, il existe deux types de paquets. Les paquets sensés être reçus (disposant d'une implémentation de la méthode run), qui prennent la main sur la routine principale, et les paquets sensés être envoyés (disposant d'une implémentation de la méthode send), qui ne font rien d'autre que d'envoyer leur commande textuelle sur le réseau. Un paquet reçu par le serveur est un paquet envoyé par le client et inversement.

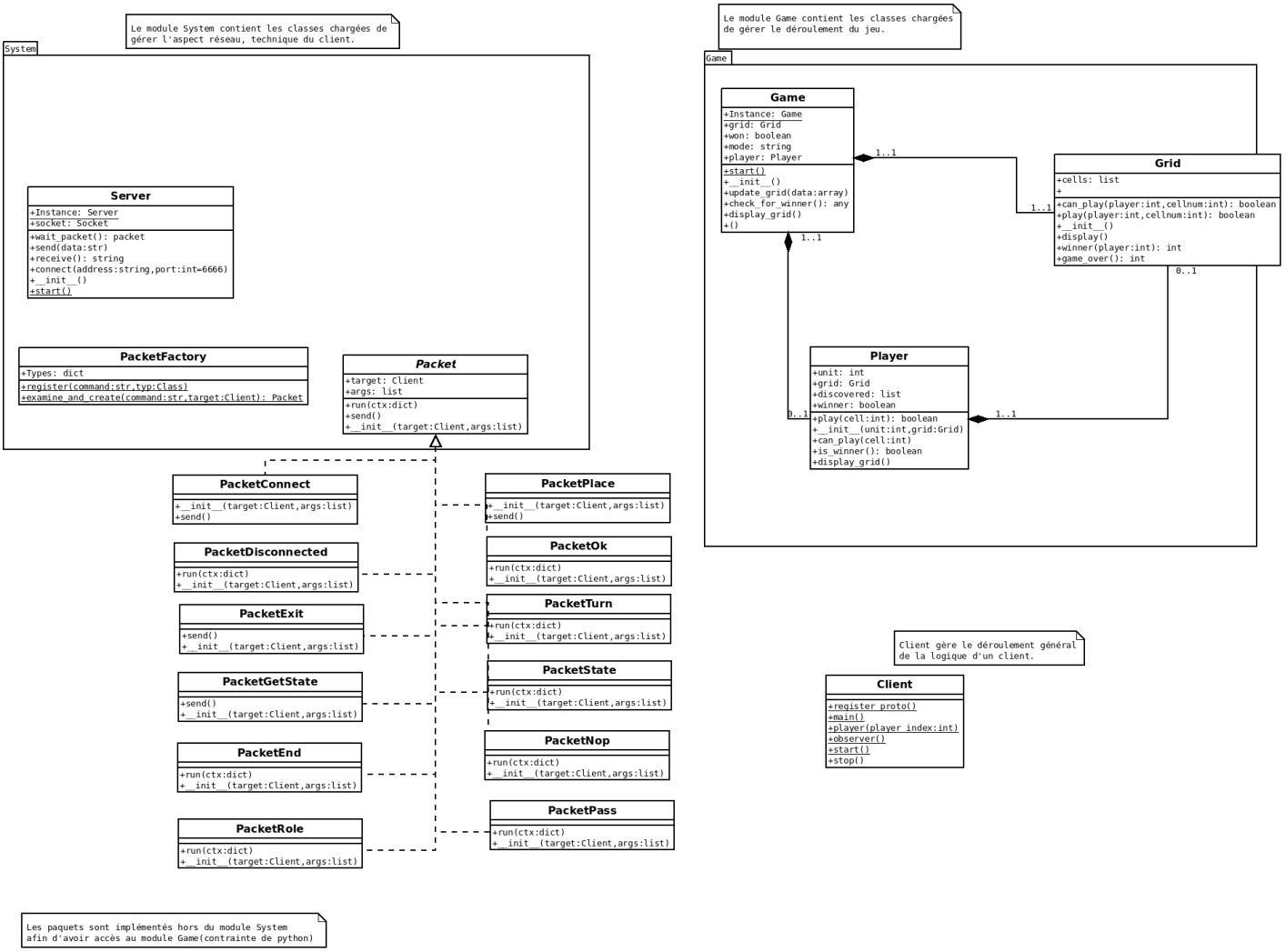
Ce choix de conception a été réalisé afin de permettre de modifier simplement le programme à l'avenir. En effet, pour ajouter n'importe quelle fonction au serveur ou au client, il suffit de créer une nouvelle classe paquet héritant de 'Packet', de l'ajouter à la méthode register_proto de client.py ou serveur.py (s'il s'agit d'un paquet sensé être reçu) et de n'ajouter qu'une ou deux lignes de code aux emplacement nécessaires. Ainsi même si la création de classe n'est pas suffisante et qu'il est tout de même nécessaire de modifier le code, on évite le coté usine à gaz d'une conception plus linéaire.

V.II.a UML du code coté serveur



(Vous pouvez retrouver le schéma UML ici)

V.II.b UML du code coté client



(Vous pouvez retrouver le schéma UML ici)

V.III Procédure qualité

Afin de produire un code de la meilleure qualité possible un certain nombre de mesures ont été prises:

- Attention particulières aux commentaires, que nous avons fait le choix de rédiger en français afin de garantir la meilleure compréhension possible au sein de l'équipe.
- L'ensemble des commentaires du projets suivent le standard pydoc. De fait, il est tout à fait possible de produire une documentation du code à part à l'aide du fameux utilitaire.
- Durant le développement nous avons utilisé pylint. Un linter permettant de produire un code respectant un certain nombre de standards.
- Le projet ayant été développé de manière simultanée et étant donné le fait que nous utilisions le logiciel de gestion de version 'Git', nous avons fait le choix de développer notre partie du projet deux branches différentes, une pour le client et une autre pour le serveur. Ceci afin d'éviter un maximum de merges pouvant nous faire perdre un temps précieux.
- Une fois les deux parties client et serveur terminées, nous avons fait le choix de réaliser une revue de code par paire. Chacun reprenant le code de l'autre afin de repérer d'éventuelles améliorations possibles ou des erreurs certaines.

VI. Capitalisation d'expérience

Ce projet s'est révelé être excessivement enrichissant à plusieurs niveaux: techniques, humain, apprentissage relatif à la conception d'un "jeu vidéo". Néanmoins, le choix de python comme langage de développement s'est révélé être relativement handicapant. En effet, notre code reposant énormément sur l'orienté objet, user d'un langage qui n'effectue pas de contrôle des types avant l'exécution fut compliqué, python ne se prétant pas spécialement à l'usage d'un tel design de code.
Concernant le modèle de conception utilisé pour notre projet, en plus du fait qu'il aurait été judicieux d'en changer étant donné l'usage obligatoire de python, celui-ci aurait nécessité un plus grand travail en amont afin d'en maximiser les bénéfices.