Projet de Programmation réseau
Du rififi au paradis

L'objectif du projet est de réaliser en binômes un jeu de rôle basique en réseau.

On rappelle qu'il s'agit d'un projet de programmation *réseau*, c'est donc cet aspect qui sera évalué avant tout.

1 Sujet

1.1 Quelques liens utiles :

— Sockets : `https://docs.python.org/library/socket.html`
— Select : `https://docs.python.org/library/select.html`
— Pygame : `http://www.pygame.org/docs/`

1.2 Description générale

Rien ne va plus au paradis, Adam et Even jouent de vitesse pour manger la dernière pomme, et le serpent mord ceux qui l'approchent.

Ce jeu se joue donc à deux, chacun contrôlant un joueur. Dans la version fournie (voir section suivante), le jeu se joue avec qsdz pour Eve et avec les flèches pour Adam. C'est un "jeu de rôle" plutôt qu'un jeu de vitesse : les personnages ne se déplacent réellement qu'une fois que chacun a choisi dans quelle direction il veut aller. Dans la version fournie, on montre le choix de la direction en affichant une deuxième fois le personnage à la nouvelle position.

L'objectif est de passer ce jeu en réseau, chacun jouant sur sa propre machine pour contrôler l'un des joueur, en voyant comment se déplace l'autre joueur.

1.3 Information relative au code source

Le projet est disponible à `http://dept-info.labri.fr/~thibault/Reseau/eden.tgz` , il contient simplement les images pour le jeu et un bout de code python utilisant pygame pour implémenter une version basique du jeu.

La structure du code est très simple : on a une carte du jeu indiquant les positions des pierres et les positions initiales des objets mobiles. On a alors les objets *woman*, *man*, *snake*, et *apple* qui représentent les protagonistes.

La boucle principale teste d'abord la pression des touches, puis calcule le déplacement des objets, détecte les collisions, affiche le résultat, et teste si un joueur a perdu avant d'attendre de nouveau la pression de touches suivante.

2 Passage en réseau

Il s'agit maintenant de jouer avec deux machines plutôt qu'une seule.

Sur une machine, on lance le programme sans argument, et le programme doit alors ouvrir une socket TCP en écoute. Sur l'autre machine, on lance le programme avec le nom de la première machine en paramètre, le programme doit, dans ce cas, se connecter en TCP à la machine passée en paramètre.

Maintenant que l'on a une socket établie entre les deux programmes, il s'agit de transmettre les informations sur les déplacements des objets, chaque machine envoie le déplacement de son joueur une

fois choisi, et attend de recevoir le déplacement envoyé par l'autre joueur avant de déplacer réellement les objets et passer à la suite. Il faut donc établir un protocole pour exprimer les déplacements. Il est recommandé d'établir un protocole textuel ascii, qu'il sera ainsi facile de débugguer à la main.

Réfléchissez et implémentez dans un premier temps votre propre protocole.

3 Extensions

Une fois cette version de base développée, de nombreuses extensions sont possibles et intéressantes à développer. Il n'est pas obligatoire de toutes les développer pour avoir une bonne note, mais cela y contribue bien sûr fortement, il faut en développer au minimum quelques-unes. La liste n'est également pas exhaustive. Si vous avez des idées, discutez-en avec votre chargé de TD.

3.1 Choix de la carte

Pour l'instant, la même carte est toujours choisie, il faudrait pouvoir varier : l'une des machines envoie la carte choisie aux autres machines.

Mais comment va-t-on choisir la carte ? Si c'est l'un des joueurs qui choisit la carte, il pourrait se donner un avantage dans la carte en se plaçant proche de la pomme.

Une solution classique venant de la théorie des jeux est que l'un des joueurs choisit la carte, et les autres joueurs choisissent quels rôles il vont jouer dans la carte, celui ayant choisi la carte prenant le rôle restant. Ainsi, le joueur qui choisit la carte a intérêt à équilibrer les avantages, pour que les autres joueurs ne puissent pas se donner un avantage en choisissant judicieusement leurs rôles.

3.2 Multi-joueur

On peut vouloir jouer à plus que 2 joueurs : pour piloter le serpent, ou pour introduire un autre personnage.

Dans un premier temps pour simplifier, le troisième joueur lance le programme en passant en paramètre le nom des machines des deux autres joueurs, et il se connecte à ces deux machines. Ainsi, chacune des 3 machines est connectée aux 2 autres.

On peut passer à quatre joueurs, etc. sur le même principe, mais cela devient fastidieux de devoir passer le nom de toutes les autres machines. C'est plus simple pour les joueurs que lorsqu'une machine se connecte à une autre, il en récupère la liste de toutes les autres machines auxquelles il faut se connecter. Il suffit ainsi à un joueur de juste passer en paramètre au programme le nom d'une des machines participant au jeu, et le programme s'occuper de se connecter aux autres machines.

3.3 Déconnexion

Il se peut qu'un des joueurs veuille quitter la partie en cours, voire que son ordinateur crashe... On préfère éviter d'abandonner complètement le jeu dans ce cas, il vaudrait mieux juste faire disparaître son personnage du jeu.

3.4 Éviter la triche

Un défaut de la version fournie est que le premier joueur qui choisit une direction a un désavantage : les autres joueurs peuvent alors choisir leur direction en fonction de cela. Du coup chacun va attendre que l'autre se décide... Mais puisque notre jeu est en réseau on peut corriger cela.

Dans une version simpliste, on peut simplement éviter d'afficher la nouvelle position des personnages, pour que les joueurs ne voient pas la direction choisie. Ce n'est cependant pas suffisant : il suffit qu'un joueur utilise une version du programme qui affiche la position pour pouvoir conserver l'avantage.

Pour corriger cela au niveau du protocole lui-même, on peut procéder en trois temps :

— quand un joueur choisit sa direction, au lieu d'envoyer cette direction « en clair », il l'envoie de manière hachée avec un sel : au lieu d'envoyer `"left"`, on envoie par exemple :

```
sel = "%02x" % ord(os.urandom(1)[0])
s = crypt.crypt("left", sel)
```

(il faut donc utiliser `import crypt, os`).

— une fois que les joueurs ont reçu les directions des autres joueurs, les joueurs envoient leur direction choisie, cette fois en clair, et le sel utilisé pour le hachage.
— les joueurs vérifient que la version hachée qu'ils avaient reçue initialement correspond bien au choix de direction et au sel finalement révélés.

Ainsi, les joueurs fournissent leur choix de manière cachée, mais irrévocable : chacun peut vérifier que les autres n'ont pas changé d'avis entre l'envoi haché et l'envoi en clair.

Mathématiquement, cela s'appuie sur le fait qu'il est difficile de retrouver la direction en ayant seulement la version hachée sans connaître le sel utilisé (c'est donc caché), et qu'il est difficile d'obtenir la même version hachée en choisissant un sel approprié (c'est donc irrévocable).

La version montrée ici est simpliste : on utilise un seul octet de données aléatoires, donc deux chiffres hexadécimaux, c'est donc facile à casser avec un *brute-force*, alors que crypt() peut prendre deux lettres alphanumériques voire '.' et '/'.

Mieux, on peut changer d'algorithme de hachage, consultez la section Notes de `man 3 crypt` : il suffit de passer par exemple comme sel `$6$1234567890123456` où `$6$` choisit l'algorithme SHA-512 (le plus fort actuellement), et les 16 caractères suivant peuvent être aléatoirement choisis parmis les caractères alphanumériques et '.' et '/'.

3.5 Items spéciaux

Pour complexifier un peu le jeu, on peut ajouter des items spéciaux à la carte : un steak magique qui permet d'avancer de deux cases par tour, une potion magique qui permet de se faufiler entre les pierres, une carte magique qui permet de traverser les bordures de la fenêtre, un pistolet pour tuer l'autre joueur, etc.

Ou inversement des malus qui ralentissent, etc.. On peut même introduire des bonus/malus aléatoires : en vous inspirant du hachage utilisé pour ne pas dévoiler son déplacement, faites en sorte que cela puisse être choisi aléatoirement sans possibilité de triche.

4 Organisation

Le projet est étudié et travaillé en binôme. Le projet *doit* être réalisé avec un outil de gestion de révision (svn, git, ...) : directement dans votre *home* ou sur le gitlab du CREMI (`https://gitlab.emi.u-bordeaux.fr/`). Afin de permettre un travail profitable, il est conseillé de ne pas créer de groupes avec des niveaux trop différents.

5 Rapport

Il s'agit de mettre en valeur la qualité de votre travail à l'aide d'un rapport. Pour cela le rapport doit explicitement faire le point sur les fonctionnalités du logiciel (lister les objectifs atteints, lister ce qui ne fonctionne pas et expliquer - autant que possible - pourquoi).

Ensuite le rapport doit mettre en valeur le travail réalisé sans paraphraser le code, bien au contraire : il s'agit de rendre explicite ce que ne montre pas le code, de démontrer que le code produit a fait l'objet d'un travail réfléchi et même parfois minutieux. Par exemple, on pourra évoquer comment vous avez su résoudre un bug, comment vous avez su éviter/éliminer des redondances dans votre code, comment vous avez su contourner une difficulté technique ou encore expliquer pourquoi vous avez choisi un algorithme plutôt qu'un autre, pourquoi certaines pistes examinées voire réalisées ont été abandonnées.

Il s'agira aussi de bien préciser l'origine de tout texte [1] ou toute portion de code empruntée (sur internet, par exemple) ou réalisée en collaboration avec tout autre binôme. Il est évident que tout manque de sincérité sera lourdement sanctionné.

1. Directement dans le texte, par une note en bas de page comme celle-ci ou par une référence bibliographique entre crochet [1].