RAPPORT FINAL DE PROJET RESEAU

MONOPOLY

Falguerolle Louis
17805771

INSTALLATION

Langage C#

Installation nécessaire :
Compilateur Mono
(https://www.monoproject.com/download/stable/#download-lin)

Compilation :
cd Monopoly-master
make clean; make; ./Monopoly.exe

LE JEU

Le Monopoly est un jeu de plateau qui nécessite de la stratégie pour acheter un maximum de terrains pour ensuite faire payer à ses adversaires des loyers. Le but du jeu est de ne pas tomber en faillite, donc toujours garder son argent a plus haut que 0. Pour cela, vous avez 1500 $ dès le départ et vous avancez ainsi chacun votre tour sur le plateau (situé à gauche de l'écran) avec 2 dés qui émettent des chiffres aléatoires entre 1 et 6. Chaque case sur lesquelles vous tombez émettent des interactions. Les interactions de Propriétés se passent via une fenêtre de dialogue qui offre plusieurs choix :

- Propriétés : Si vous ne la possédez pas, vous pouvez l'acheter. Si c'est le cas, vous pouvez acheter des maison qui vont augmenter le loyer de cette propriété (Sauf les gares et les compagnies d'eau et d'électricité). Vous pouvez aussi la vendre. Si vous tombez sur une case adverse, vous payez un certains montant qui est inscrit lorsque vous passez votre souris sur la case.
- Impôts : Vous payez un certains montant.
- Chance (?) : Vous gagnez ou perdez un montant entre 200 et -200.
- Prison : Vous vous déplacez de 20 cases .
- Parc Gratuit : pas d'effet.
- Départ : Cette case n'a aucun effet si vous arrivez dessus mais vous gagnez 200 $ si vous la
- dépassez.

Vous pouvez aussi voir les icônes de joueurs qui donnent les informations de nom de joueur et d' argent. En bas de ces icônes, on peut trouver une boite qui rassemble l'historique des actions des joueurs.

LE PROTOCOLE

Le jeu utilise le protocole TCP qui permet le mode connecté.

1 CONNEXION

Coté connexion, j'ai préféré tout centraliser, c'est a dire que mon serveur et mon client sont ouverts en tant que threads de fond du thread principal. J'ai implémente un système automatique qui tente une connexion vers le serveur et si il ne réussi pas, ouvre deux threads, un de serveur, et un de client. On peut ainsi soit connecter le serveur et connecter le client au *localhost* directement ou alors se connecter depuis un autre poste au serveur distant déjà créé.

Lorsque le client est connecté, le serveur et le client lancent tout les deux un thread nommé `ReceiveThread` qui va réceptionner les informations que les deux entités s'échangent. On envoie au serveur un message commençant par **PSD** qui contient le pseudo du joueur et il nous renvoie un message commençant par **NB** qui va nous permettre de connaître l'ordre par lequel les joueurs sont arrivés. On attend que tout les joueurs soient connectés (on change la nombre de connexion en changeant la valeur « `nbconnexion` » du fichier serveur.cs).

Lorsque tout le monde est connecté, le serveur envoie un message commençant pas **INF** qui contient le nombre de joueurs et tout les pseudos. On l'envoie a tout les clients pour que chacun accède aux mêmes informations.

Lorsque les clients on reçu le message, on change le booléen « begin » qui va nous permettre de lancer une nouvelle instance de Monopoly.

2 PARTIE

Le serveur commence par lancer les dés et envoyer notre première instance de **PLT** à tout les clients.

PLT se compose de :
6 * POSITION JOUEUR (0 a 39)
6 * ARGENT JOUEUR (0 a infini)
56 * TERRAIN PROPRIETAIRE (0 à 6) / NOMBRE DE MAISON (0 à 5)
2 * NUMERO DES DES (1 à 6)
1 * NUMERO DU JOUEUR QUI JOUE (0 a 5)
1 * NUMERO DU TOUR (0 a infini) (incrémente lorsque tout les joueurs ont joués)

Chaque nombre est séparé par un « / » lors de l'envoi. On envoie le premier PLT avec la position du premier joueur pour que le joueur puisse avancer directement.
Dans son *thread*, le client reçoit le message et l'ajoute, dans une *blocking queue* nommée « `blk` », qui va permettre d 'effectuer des actions *threadsafe*. La *blocking queue* est partagée entre le client et le monopoly.
Le client ajoute les données pendant que le monopoly les retire et les utilise dans le plateau.

Le monopoly lance un *thread* nommé « `Monopoly.PltThread` » qui va effectuer qui va justement retirer les informations contenues dans la queue. Les informations de jeu sont appropriées avec la fonction « `Monopoly.getplt` » et ensuite l'affichage plateau est actualisé avec l'aide de 5 fonctions :

- `Plateau.updateDes()` qui actualise les dés affichés à l'écran.
- `Plateau.updateMoney()` qui actualise l'argent des joueurs.
- `Plateau.updatePosition()` qui actualise les positions des joueurs.
- `Plateau.updateOwnerAff()` qui actualise les informations de propriétaire.
- `Plateau.updateMaisonAff()` qui actualise les informations de maison affichés en bas à droite des cases.
-

Ensuite, on lance la fonction « `Game()` » qui va gérer les interactions des jeu pour le joueur concerné. Lorsqu'il va tomber sur un certains type de case, cette fonction va lancer l'interaction concernée.
A la fin de cette fonction, on actualise le tour des joueurs avec « `Plateau.updateTour()` » et on envoie les informations de jeu au serveur avec la fonction «`Monopoly.monopsendplt()`» qui va les renvoyer aux autres clients. Et c'est ainsi que se termine le tour du joueur.

Si le joueurs n'a plus d'argent, on envoie au serveur un signal commençant par **FIN** et qui contient le nom du joueur qui est tombé en faillite. Ce signal arrête la partie.

PROTOCOLE COMPLEMENTAIRE

Pour garder une trace des actions des joueurs, j'ai donc choisi de mettre en place un historique qui s'actualise automatiquement. Dans la fonction « Monopoly.`Game()` », on retrouve a chaque fois la fonction « `Monopoly.print()` » qui copie le message donné dans notre *TextBox* nommée « `receivebox` ». On envoie donc a chaque fois un message au serveur commençant par **HIS** et qui contient les données a retransmettre. Le serveur va le renvoyer a tout les clients sauf a celui qui l'a envoyé pour éviter les doublons. Le client le récupère via la *blocking queue* et l'affiche.

STRUCTURE

Le code contient 7 classes principales :

ECRANSELECTION.CS
Cette classe est un héritage de la classe Form de la bibliothèque, elle gère le menu de sélection du jeu . De base, le jeu devait avoir une version locale mais cela n'a pas pu être fait par faute de temps. Elle contient essentiellement des éléments graphiques. C'est cette classe qui ouvre le serveur et le client et qui gère le lancement et la fermeture du jeu. Elle est cachée dès que la partie commence.

MONOPOLY.CS
Cette classe est, comme son ancêtre, un héritage de la classe Form de la bibliothèque C#, c'est elle qui gère tout le jeu et son affichage, elle contient la classe *Plateau*, les icônes des joueurs, et l'historique de jeu. Elle contient aussi la plupart des fonctions de protocoles.

PLATEAU.CS
Cette classe est un héritage de la classe Panel de la bibliothèque C#, elle gère la création des 40 cases via la classe *Case* , les ajoute dans la liste « `plateauliste` » et les affiche sur son espace. C'est aussi elle qui contient les joueurs via la classe *Player* qui les ajoute dans la liste « `playerliste` » ou qui contient les fonctions d'actualisation contenues dans le « `Monopoly.PltThread` ».

CASE.CS
Cette classe est un héritage de la classe Panel de la bibliothèque C#, elle gère la création des cases via son constructeur, elle contient la classe *Terrain* qui dérive de la classe Case, la fonction qui permet d'obtenir le loyer, et elle gère aussi l'affichage des cartes de cases affichées lorsque l'on passe sa souris sur un terrain. On peut retrouver la classe *Prison* et la classe *Caisse* qui permettent seulement de faire une vérification du type de la case.

PLAYER.CS
Cette classe est un héritage de la classe *PictureBox* de la bibliothèque C#, elle gère les déplacements, les achats et ventes de terrain ou de maison du joueur. On peut aussi voir une fonction qui remplie les icônes de joueurs de la classe *Monopoly* .

SERVEUR.CS
C'est la classe qui construit le serveur et qui contient les *Threads de reçoit*, la liste des clients connectés ou la *Socket* du serveur.

CLIENT.CS
C'est la classe qui construit le client et qui contient le T*hread* de reçoit, la *Socket* du client et la *blocking queue*.

Pour l'affichage, on a 3 dossiers contenant des images (/img), des polices (/police) et des sons (/snd).