HangmanSocket

Université de Bordeaux

Jérémy Morin
Louis Laine
Pierrick Sailler
Mathieu Merceron
Julien Couraud

Contents

Présentation du sujet

Il s'agira de réaliser un jeu du pendu basique en réseau en s'appuyant sur les sockets TCP/IP en mode connecté. Le travail consistera à définir un protocole de couche applicatif pour permettre un dialogue entre le client et le serveur. Ce mini-projet sera développé en groupe de 4/5 étudiants aux maximum.

Environnement

GNU/Linux `<3` , Sublime Text

Contraintes de développement

Nous utiliserons : un repo Github pour le versionning, la CamelCase pour la lisibilité, la séparation du code (hpp, cpp), et des commentaires du style JavaDoc pour faciliter votre lecture.

Le pendu

Nous voulions un pendu entièrement jouable en réseaux. Le serveur est lancé par l'hôte de la partie. Ensuite plusieurs clients peuvent se connecter sur celui-ci pour entamer une partie.

Pour cela nous avons implémenter une classe joueur, qui permet d'avoir un nombre de joueur dynamique sur notre serveur et ainsi avoir plusieurs jeux en cours.

Principe du jeu

Le serveur à la connexion d'un nouveau client demande au client s'il veux commencer la partie dans le cas d'une réponse favorable. Le serveur initialise un mot qu'il parse dans une liste préablement remplie de mots.

Le serveur grace à une classe joueur, peux creer plusieur joueurs, il envoit au(x) client(s) un mot avec des caractère vide à l'emplacement des lettres non repondu par le(s) joueur(s).

le joueur à un nombre limité de 7 tentative, il peux suivre l'evolution des ces erreurs grace à un graphique du pendu ainsi qu'un indicateur des lettre déjà utilisé et du nombre de tentatives restantes.

Une fois le mot découvert le jeu est terminé, le joueur ainsi que ces attributs privées sont détruit par un `delete`.

Les classes

La classe joueur

```
class joueur
{
public:
    joueur();
    ~joueur(){};
    void setLetterList(string _letter);
    string getLetterList();
    string getWord();
    void setBlank(string blankWord);
    string getBlank();
    string actionBlankWord(string wordBlank, string letterUti);

protected:
    string word;
    string letterList;
    string blank;
};
```

La classe joueur est la solution, la plus pertinante pour un pendu entierement dynamique.

La classe client

Dans la classe client, nous avons ajouté une fonction *answerChar* qui demande au client s'il veut commencer une partie *Do you want to play ? (y/n)*. Ce dernier doit répondre *y* ou *n*. Enfin, si la réponse est *n*, la fonction *answerChar* quitte l'application.

Pour finir, nous avons créé la fonction *hangingMan* affiche le personnage du pendu du nombre d'erreurs faîte par le client.

```
    class SocketClientTest : public SocketClient
    {
    public:
        SocketClientTest(std::string hostname, int port): SocketClient(
            hostname,port) {};
        virtual void work(int);
        string answerChar(int);
        void hangingMan(int);
    };
```

La classe server

Dans la classe server nous avons ajouté une fonction *askConnexion* qui envoi une question au client, puis attend une réponse de ce dernier.

```
class SocketServerTest : public SocketServer,joueur
{
public:
    SocketServerTest(int port) : SocketServer(port) {};
    ~SocketServerTest(){}
    virtual void work(int fdw);
    string askConnexion(int fwd,string question);
};
```

Contact

Mail : jer.morin@free.fr