

---

# Rapport de projet réseau

## *C++*

---

EDGAR OBLETTE  
edgar.oblette@etud.univ-paris8.fr

ÉTUDIANT LICENCE INFORMATIQUE  
UNIVERSITÉ PARIS VIII  
NO ÉTUDIANT: 17812871

RAPPORT DE PROJET  
21 Avril, 2019

# 1 Introduction

Le projet décrit ici est composé d'une partie Client et d'une partie serveur, la partie client utilise une interface graphique réalisée grâce à la librairie Qt

Le Projet est un client/serveur de chat/jeu en ligne utilisant le protocole TCP/IP.

Le client fournit au serveur les données pour effectuer des traitements.

Le serveur rend le service et retourne au client les résultats.

Le serveur peut être sollicité par plusieurs clients pour le même service.

Chaque application peut être distante : réseau. Clients et serveur peuvent s'exécuter sur des machines différentes.

Vous trouverez un dossier Serveur avec son propre Makefile et un dossier Client avec un Makefile généré par Qmake.

## 2 Compilation

### 2.1 Programme du Serveur

Pour compiler et lancer le *serveur*:

```
make broadcast ; broadcast numero_de_port
```

Exemple :

```
make broadcast ; broadcast 4567
```

### 2.2 Programme de Client

Pour compiler et lancer le *client*:

```
make ; open Gui_projet_reseau.app
```

PS: il se peut que la compilation ne marche pas, dans ce cas ouvrez juste l'app

Gui\_projet\_reseau.app

et si cela ne marche toujours pas, allez page 11.

## 3 Serveur

Le serveur s'occupe d'attendre les demandes de connections de nouveau client et de répondre aux commandes envoyées par les clients déjà connectés.

À la demande d'un client, il doit utiliser le protocole défini pour analyser la demande et répondre en utilisant une réponse définie par le protocole. Le modèle de serveur utilisé ici est le serveur parallèle, le serveur traite plusieurs demandes simultanément.

Le fonctionnement du serveur est découpé en 2 phases:

1. `Start()`
2. `Listen()`

### 3.1 `Start()` :

La méthode *Start()* de la classe *Server* fait successivement:

1. Création de la socket.
2. Bindage de la socket.
3. Mise de la socket en Non bloquant.

Si tout se passe comme prévu on continue sinon on quitte le programme et affichant un code d'erreur.

### 3.2 `Listen()` :

La méthode *Listen()* de la classe *Server* fait successivement :

1. L'écoute de la socket (non-bloquante).
2. Initialise le vector de pollfd *mConnectionPool*.
3. Initialise le vector de client *mClientPool*.

puis elle fait en boucle:

1. Lecture des socket grâce à poll.
2. Si nouveau client:
  - (a) Accepte le nouveau client.
  - (b) Si le nombre de personnes max est dépassé on le déconnecte et on lui indique pourquoi.
  - (c) Sinon on exécute la fonction *Identity()*.
3. Si un client envoie un message sur la socket

- (a) On lance une thread avec la methode *Manage()* de la classe *Server*
- (b) On cherche qui parle.
- (c) Si le *read()* nous renvoie 0 alors on déconnecte le client (client à quitte son programme sans notifier son départ)
- (d) Sinon on vérifie qu'il est identifié:
  - i. Si client non identifie on rappelle la fonction *Identity()*
  - ii. Si client identifie on appelle la fonction *HandleMessage()*

FIN de *Listen()*.

## 4 Client

Le client est à l'initiative de la demande, il doit s'adresser au bon serveur, lui fournir toutes les données pour effectuer le service, attendre les résultats et assurer la présentation.

Le client encode et décode les message envoyer avec le serveur grâce au protocole défini.

Pour faire la partie GUI du client j'ai utilisé la librairie Qt, et utilisé la méthode MVC (Modèle-vue-contrôleur) pour faire le pont être la partie communication serveur et affichage utilisateur.

Étant donné que c'est la première fois que j'utilise cette librairie et cette méthode, il se peut que mon code ne soit pas écrit de façon optimale.

Le fonctionnement du client est découpe en 2 partie:

1. Connexion au serveur
2. Échange avec le serveur

### 4.1 Connexion au serveur :

C'est la première fenêtre que utilisateur voit, lui demandant d'indiquer le serveur a joindre, le port et le pseudo utilise. Si toutes les informations sont correctes on lance la méthode *ConnectPageSetting()* de la classe *EventHandler* fait successivement:

1. La connexion au serveur avec la méthode *Connect()* de la classe *Client*.
2. Fermeture de la première fenêtre (Connect page) et ouverture de la fenêtre principale (Chat).
3. Lancement en thread de la méthode *Talk()* de la classe *Client*.

### 4.2 Échange avec le serveur

C'est la fenêtre principale qui permet à l'utilisateur de recevoir/envoyer des message public/prive et de lancer/rejoindre un jeu en ligne.

La thread *Talk()* de la classe *Client* récupère les messages envoyé par le serveur et faire une demande d'affichage à la méthode *IncomingMessage()* de la classe *EventHandler* qui fait:

1. Analyse de la demande protocole.
2. Si c'est un message public, alors on affiche le message dans la fenêtre principale.
3. Si c'est un message privé, on affiche le message dans une fenêtre dédié (Private Message).

4. Si c'est l'arrivée d'un nouveau client on ajoute le client à la liste des utilisateur en ligne.
5. Si c'est le départ d'un client on le supprime de la liste des utilisateur en ligne.
6. Sinon par défaut on affiche le message dans la fenêtre principale

Quand l'utilisateur envoie un message public à partir de la fenêtre principale ou un envoi un message privé à partir de la fenêtre dédiée (Private Message) . On fait une demande d'envoi de message à la méthode *SendingMessage()* de la classe *EventHandler* qui fait lancer en thread la méthode *HandleMessage()* de la classe *Client* qui envoie le message au serveur.

Quand l'utilisateur quitte l'application, on appelle la méthode *Disconnecting()* de la classe *EventHandler* qui appelle la méthode *Disconnect()* de la classe *Client* qui envoie un message formaté indiquant le départ.

Voici la table de construction des fenêtres:

### 1. Connect Page

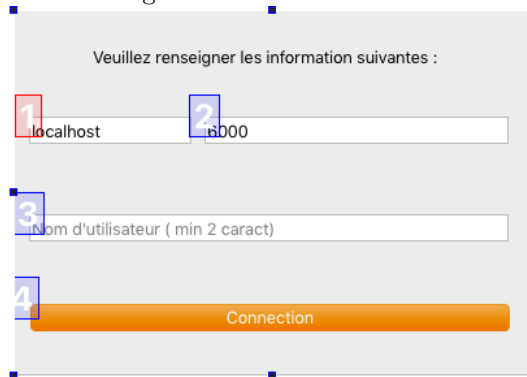


Diagram of the Connect Page window. It features a light gray background with a title bar. The text "Veuillez renseigner les information suivantes :" is centered at the top. Below this, there are four numbered callouts: 1 (red box) points to a text input field containing "localhost"; 2 (blue box) points to a text input field containing "8000"; 3 (blue box) points to a text input field containing "Nom d'utilisateur ( min 2 caract)"; and 4 (blue box) points to an orange button labeled "Connection".

### 2. Fenêtre Principale

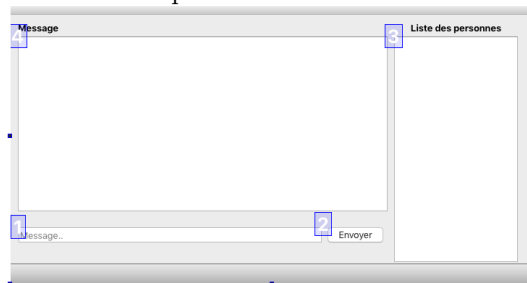


Diagram of the Fenêtre Principale window. It has a light gray background and a title bar. The window is divided into two main sections. The top section is labeled "Message" (1) and contains a large text area. The bottom section is labeled "Liste des personnes" (3) and contains a list box. At the bottom of the window, there is a text input field labeled "Message..." (1) and an "Envoyer" button (2).

### 3. Private Page

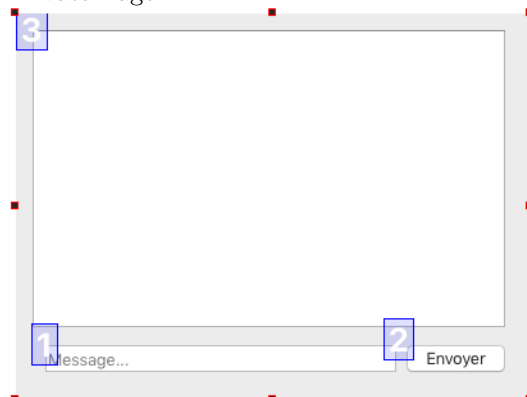
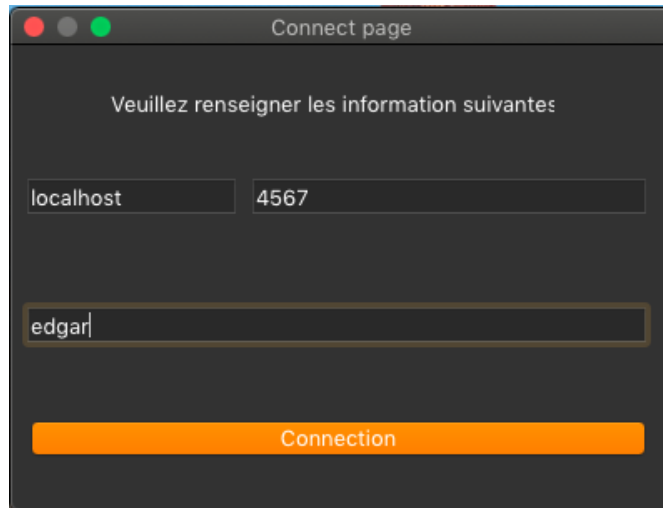


Diagram of the Private Page window. It has a light gray background and a title bar. The window is divided into two main sections. The top section is labeled "Message" (1) and contains a large text area. The bottom section is labeled "Liste des personnes" (3) and contains a list box. At the bottom of the window, there is a text input field labeled "Message..." (1) and an "Envoyer" button (2).

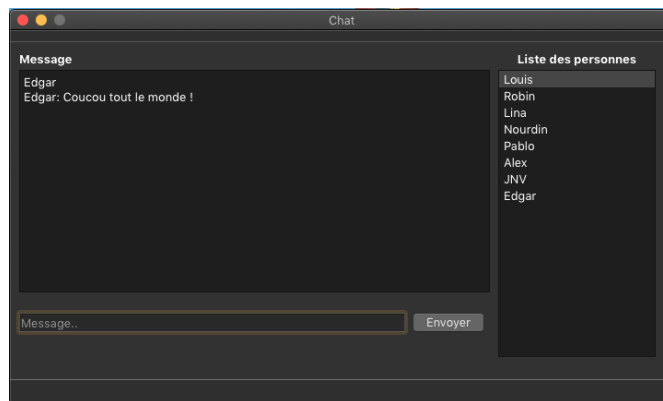


## 5 Client exemple:

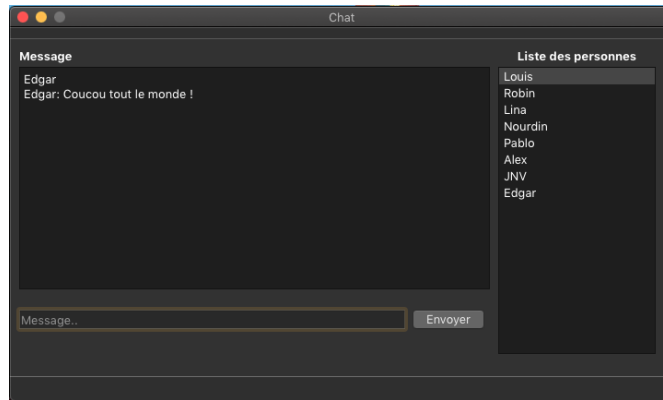
1. On se connecte au serveur (localhost, 4567) avec le pseudo Edgar.



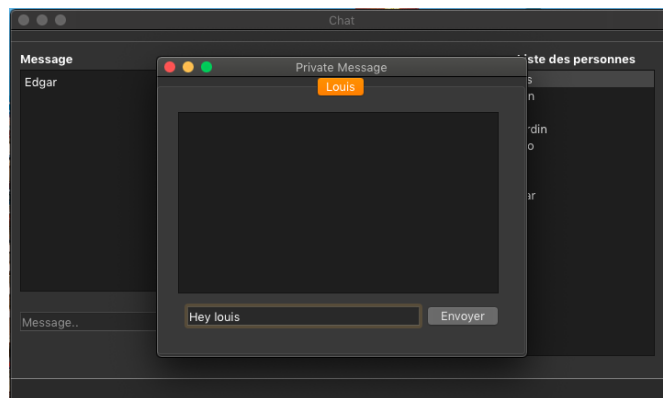
2. On arrive sur la page principale.



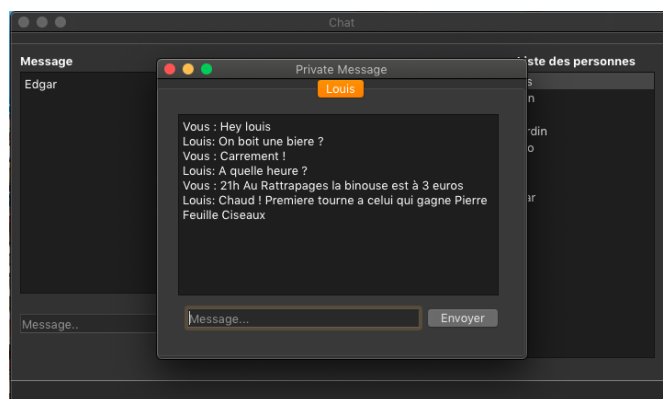
3. On envoi un message à tout le monde.



4. On clique sur "Louis" et une fenêtre de message privé s'ouvre et on écrit notre message.



5. On discute avec "Louis"



6. On lance le jeu Pierre Feuille Ciseaux (non implémenté dans cette version, voir page explication page 11 )
7. On savoure notre victoire !



## 6 Notes:

- Les programmes se compile avec la version 17 de c++
- La version du Makefile pour le client n'est peut être pas fonctionnelle pour une machine autre que *macos* (dite moi si vous n'arrivez pas à compiler, je vous enverrais une version compilable sur votre système)

### 6.1 Implémentations des jeux

Je n'ai malheureusement pas réussi à implémenter dans cette version, les jeux (Morpion et Pierre Feuille Ciseaux) par manque de temps. Néanmoins malgré la date de remise dépassé, je continue à finaliser et faire une version fonctionnelle incluant les jeux décrits. Je vous enverrais celle ci d'ici une semaine à compter du 21/04/2019. Même si celle là n'est pas évalué, je compte vous témoigner de mon sérieux et de l'envie de mener à bien ce projet.

Merci de votre lecture.