# Introduction

L’idée du projet est de permettre à un client de se connecter à un serveur.

L’étape suivante est que le serveur attribut un bateau à ce client , par la suite, il peut réaliser les actions suivantes :

* Editer la vitesse ou la direction de son bateau
* Voir la navigation de tous les bateaux dans l’océan.Le but derrière ce projet est de mettre en œuvre les compétences acquises tout au long dans ce module pour créer une architecture client/serveur.

# Outils de travail

1. Code Block

Notre travail a été fait en utilisant l’IDE code block.

Code::Blocks est un environnement de développement intégré libre et multiplate-forme. Il est écrit en C++ et utilise la bibliothèque wxWidgets. Code::Blocks est orienté C et C++, mais il supporte d'autres langages comme FORTRAN ou le D.

Code::Blocks existe pour Linux, Windows et Mac OS X. Des utilisateurs indiquent avoir réussi à compiler le code source sous FreeBSD[réf. nécessaire]1.

Il obtient une note de satisfaction de 4.7 (sur un maximum de 5) de la part de ceux qui l'ont téléchargé sur SourceForge.

1. Language C

Ce projet a été réalisé avec le langage C, qui a été inventé aux débuts des années 70 pour réécrire UNIX. Plusieurs langages modernes, comme par exemple le C++, C#, ou encore PHP se sont basés sur sa logique et aussi sa syntaxe.

1. Architecture client Serveur :

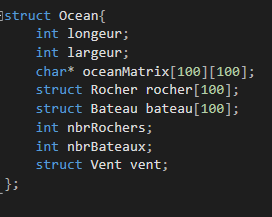
L'environnement client–serveur désigne un mode de communication à travers un réseau entre plusieurs programmes : l'un, qualifié de client, envoie des requêtes ; l'autre ou les autres, qualifiés de serveurs, attendent les requêtes des clients et y répondent. Par extension, le client désigne également l'ordinateur ou la machine virtuelle sur lequel est exécuté le logiciel client, et le serveur, l'ordinateur ou la machine virtuelle sur lequel est exécuté le logiciel serveur. Les serveurs sont des ordinateurs généralement destinés au logiciel serveur qu'ils abritent, et dotés de capacités supérieures à celles des ordinateurs personnels en ce qui concerne la puissance de calcul, les entrées-sorties et les connexions réseau. Les clients sont souvent des ordinateurs personnels ou des appareils individuels (téléphone, tablette), mais pas systématiquement. Un serveur peut répondre aux requêtes d'un grand nombre de clients.

Notre architecture client/serveur a été réalisée avec les sockets, qui permettent de créer un flux de communication entre chaque client et le serveur, à l’aide des adresses IPs. Les sockets peuvent avoir plusieurs utilisations : jeux en lignes, systèmes distribués, applications …

# Résolution du problème

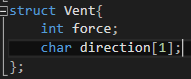
1. Les structures utilisés

Pour resoudre ce problème nous avons concrétisé l’ocean à travers une structure qui s’appelle à son tour Ocean

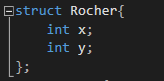


La structure Ocean contient tout les rochers, les bateaux et une matrice permettant de savoir ou se trouve chaque bateau ou rocher dans l’ocean.

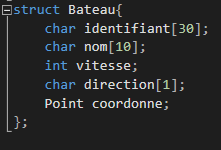
Le vent en lui-même est representé par une structure nommée Vent



Le vent est caractérisé par sa force et sa direction.



Un Rocher est caractérisé par une abscisse et ordonnée dans l’ocean.

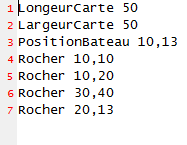


Dernièrement un Bateau est caractérisé par une vitesse, une direction et ses coordonnées dans l’ocean.

1. Lecture du fichier de config

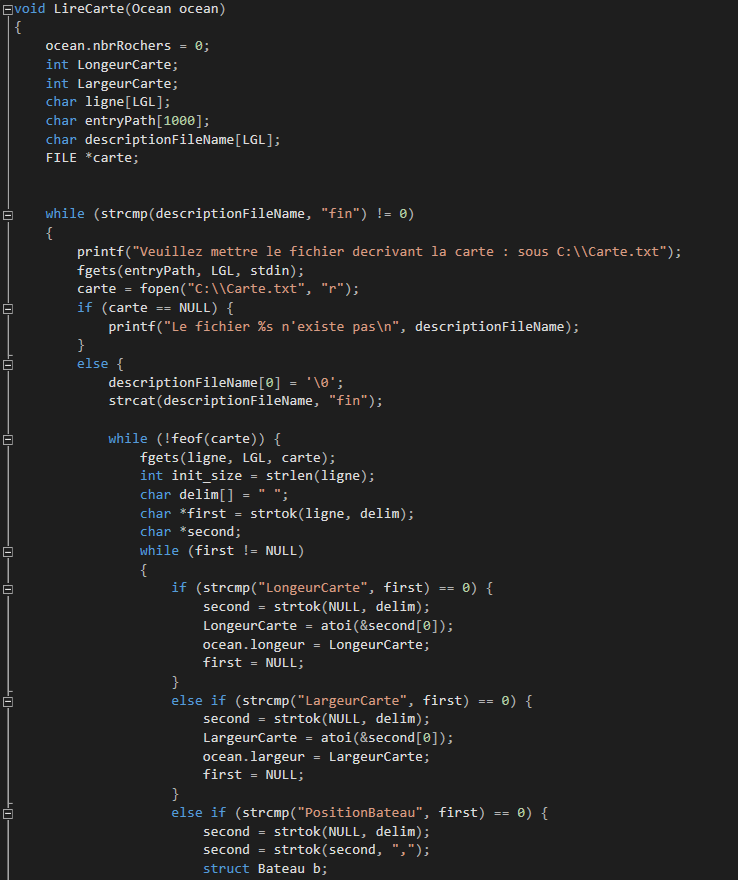
L’initialisation de l’ocean et de son contenue passe par la lecture d’un fichier de config (Carte.txt) qui doit être mis sous c :

Ceci est un extrait de ce fichier :



La lecture du fichier est faite à travers la méthode LireCarte qui permet d’initialiser la structure Ocean, une seule contrainte est de trouver les tag qui sont figé en dur dans le code.

* LongeurCarte : pour lire la longueur de l’ocean
* LeargeurCarte : pour lire la largeur de l’ocean
* PositionBateau : pour initialiser un bateau au position X,Y
* Rocher : pour positionner un rocher à la position X,Y



On a commencé par déclarer les différents types des structures océan, Dans la classe serveur.

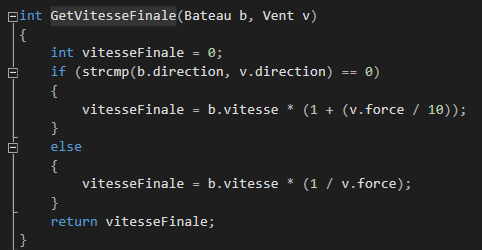
Ici on a défini la structure basique des sockets :



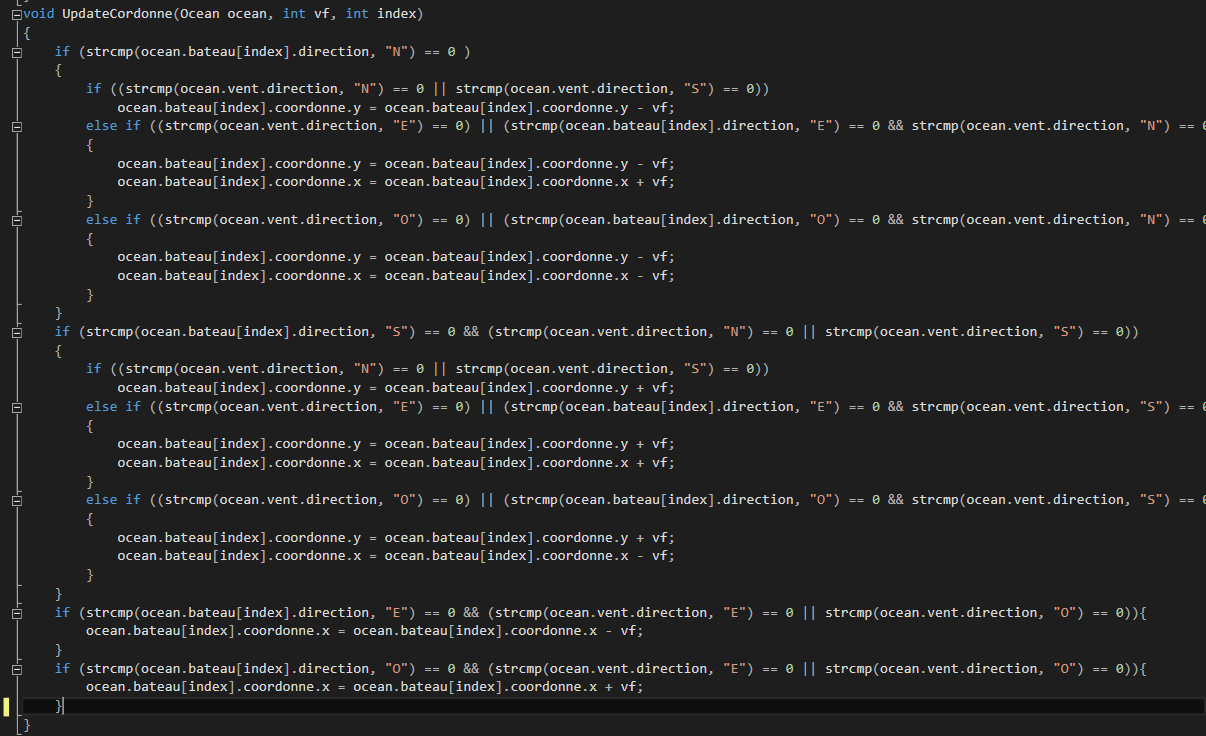
Pour le socket client :



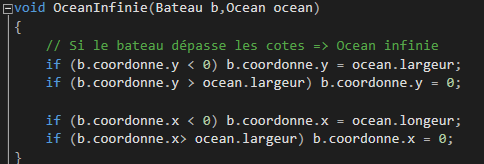
Pour calculer la vitesse finale du bateau nous appelons la methode **GetVitesseFinale**



Pour mettre à jour les coordonnées du bateau nous appelons la methode **UpdateCoordonne**

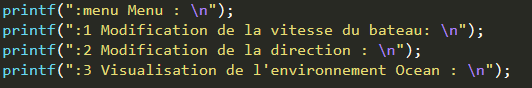


Afin que l’ocean semble infinie nous avons crée la methode OceanInifinie qui modifie les coordonnées du bateau s’il dépasse l’ocean d’une côté ou de l’autre.



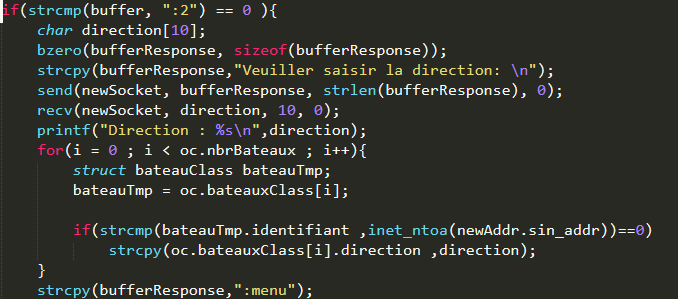
La même que Serveur, mais avec une autre condition pour connecter sur le serveur.

On a défini ici le Menu de client :

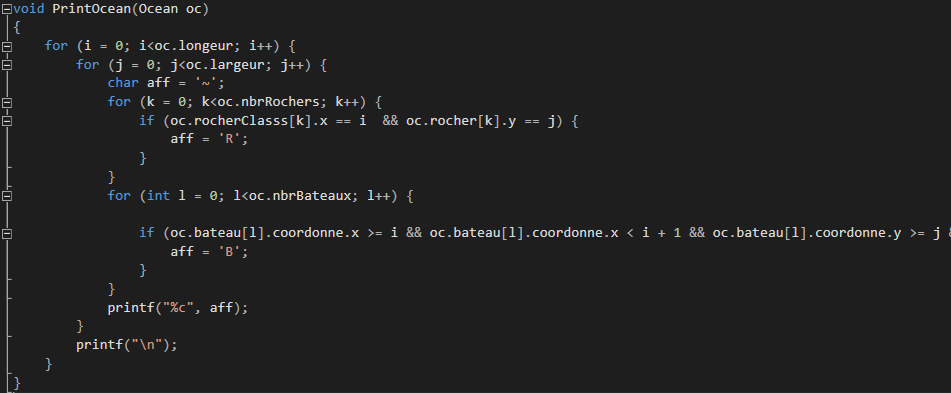


A travers ce menu, le client peut modifier la vitesse de son bateau, sa direction, ou encore visualiser l’environnement Océan.

Pour la modification de la direction :



Pour afficher l’ocean nous avons crée la methode **PrintOcean**



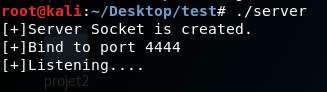
# Exécution

Evidemment, la première des choses à faire est de compiler les deux classes :

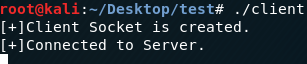




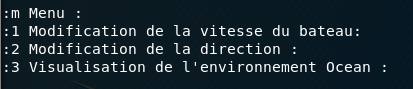
On commence par lancer le serveur :



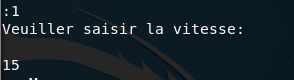
On connecte un client :



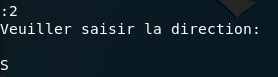
Le menu qui s’affiche au client :



La modification de la vitesse de son bateau :



La modification de la direction de son bateau :



La visualisation de l’océan avec 2 bateaux :

