

4SI1

Damien MICHAUDEL - Daniel ZEITOUN

Projet Programmation Système et Réseaux

-

Bataille Navale

Sommaire

Sommaire	1
Introduction	3
Documentation	4
Répartition des travaux	4
Découpage des modules	4
Découpage du Serveur	4
Gestion	5
Communication	5
Découpage du Client	6
Gestion	6
Communication	7
Protocole de communication	8
La structure Packet	8
Les types de messages	9
MESSAGE_TYPE_LOGIN_REQUEST	9
MESSAGE_TYPE_LOGIN_REPLY_ADMIN_SUCCESS	10
MESSAGE_TYPE_LOGIN_REPLY_SUCCESS	10
MESSAGE_TYPE_LOGIN_REPLY_BAD_USERNAME	10
MESSAGE_TYPE_LOGIN_REPLY_BAD_PASSWORD	10
MESSAGE_TYPE_CLOSE_CONNECTION	10
MESSAGE_TYPE_ACCOUNT_CREATE_REQUEST	10
MESSAGE_TYPE_ACCOUNT_CREATE_ALREADY_EXIST	11
MESSAGE_TYPE_ACCOUNT_CREATE_OK	11
MESSAGE_TYPE_ACCOUNT_DELETE_REQUEST	11
MESSAGE_TYPE_INTERNAL_ERROR	11
MESSAGE_TYPE_ACCOUNT_DELETE_SUCCES	11
MESSAGE_TYPE_ACCOUNT_DELETE_BAD_USERNAME	12
MESSAGE_TYPE_GAME_ENUMERATE_REQUEST	12
MESSAGE_TYPE_GAME_CREATE_REQUEST	12
MESSAGE_TYPE_GAME_CREATE_ALREADY_EXIST	12
MESSAGE_TYPE_GAME_CREATE_SUCCESS	12
MESSAGE_TYPE_GAME_NUMBER_REPLY	12
MESSAGE_TYPE_GAME_JOIN_REQUEST	13
MESSAGE_TYPE_GAME_JOIN_OK	13
MESSAGE_TYPE_GAME_JOIN_FULL	13
MESSAGE_TYPE_KEEP_ALIVE	13
MESSAGE_TYPE_GAME_PLAYER_NUMBER_REQUEST	13

MESSAGE_TYPE_GAME_PLAYER_EJECT	14
MESSAGE_TYPE_GAME_PLAYER_NUMBER_REPLY	14
MESSAGE_TYPE_GAME_START_REQUEST	14
MESSAGE_TYPE_GAME_START_SUCCESS	14
MESSAGE_TYPE_GAME_START_NOT_FULL	14
MESSAGE_TYPE_GAME_JOIN_ERROR	15
MESSAGE_TYPE_USER_ENUMERATE_REQUEST	15
MESSAGE_TYPE_GAME_PLAYER_EJECT_SUCCESS	15
MESSAGE_TYPE_GRID_REQUEST	15
MESSAGE_TYPE_SEND_SHOT	15
MESSAGE_TYPE_GRID_REPLY	16
MESSAGE_TYPE_PLAYER_IN_GAME_RUNNING	16
MESSAGE_TYPE_PLAYER_IN_GAME	16
MESSAGE_TYPE_ITS_YOUR_TURN	16
MESSAGE_TYPE_ITS_NOT_YOUR_TURN	16
MESSAGE_TYPE_TURN_REQUEST	16
MESSAGE_TYPE_YOU_WIN	17
MESSAGE_TYPE_YOU_LOSE	17
MESSAGE_TYPE_USER_ENUMERATE_REPLY	17
MESSAGE_TYPE_GAME_SHOW_ADMIN	17
MESSAGE_TYPE_SHOW_ADMIN_REPLY	17
MESSAGE_TYPE_GRID_ADMIN_REQUEST	18
MESSAGE_TYPE_GRID_ADMIN_REPLY	18
MESSAGE_TYPE_SCORE	18
MESSAGE_TYPE_FIRST_SCORE	18
MESSAGE_TYPE_GAME_STOP_SHOW_ADMIN	18
Mode d'emploi	19
Serveur	19
Client	19
Admin	20
Création d'un compte	21
Supprimer un compte	22
Créer une partie	22
Liste des parties	23
Se déconnecter	27
Joueur	27
Rejoindre une partie	29
Démonstration	33

Introduction

Le but du projet du cours de programmation système et réseaux est la réalisation d'un jeu fonctionnant en réseau.

Il y a donc un programme qui, dans notre cas, peut fonctionner sous Linux ou Windows et qui a pour objectif de servir de serveur. Autrement dit, il va réceptionner les données des clients, les traiter, et envoyer une réponse.

Les clients, quant à eux, fonctionnent seulement sur des environnements Windows et ont pour simple objectif d'interagir avec le serveur et afficher les informations qu'il retourne (dont le plateau du jeu).

Le jeu imposé est une bataille navale légèrement revisitée. Le but est simple, il y a une seule et unique grille contenant les bateaux pour l'ensemble des joueurs. Les joueurs doivent tenter de toucher le plus de cases possibles appartenant à des bateaux. Ils gagnent des points à chaque fois qu'ils touchent un bateau et lorsque tous les bateaux sont coulés, le joueur gagnant est celui qui aura le score le plus élevé.

Nous avons utilisé deux éditeurs différents pour concevoir les programmes :

- Pour le serveur, nous avons utilisé le logiciel Code Blocks version 17.12.
- Pour le client, nous avons utilisé Visual Studio 2019 (édition Community) car cet IDE est plus approprié pour le développement d'applications Windows.

Un fichier *Makefile* est disponible dans le dossier des sources du serveur et permet de compiler rapidement le serveur sur un environnement Linux.

Pour que le **serveur** puisse être compilé correctement sous Windows, il faut lier avec le linker la librairie "**libws2_32.a**". Celle-ci permet la gestion des fonctionnalités réseaux sous Windows.

Pour que le **client** puisse être compilé correctement, il faut lier avec le linker la librairie "**libws2_32.a**", pour la gestion des fonctionnalités réseaux sous Windows, ainsi que les librairies "**libcomctl32.a**" et "**libgdi32.a**" pour la gestion des contrôles et des fonctionnalités liées à l'interface graphique.

Les programmes déjà compilés sont disponibles dans un dossier *Executables* :

- **Client.exe** correspond à l'exécutable Windows du client ;
- **Serveur.exe** correspond à l'exécutable Windows du serveur ;
- **server_linux** correspond au binaire Linux du serveur.

Il n'y a aucun compte préexistant mis à part celui de l'administrateur. Ses informations d'authentification sont les suivantes :

- Identifiant : **admin**
- Mot de passe : **ESGI**

Documentation

Dans cette partie seront mises à dispositions les informations nécessaires à la maintenance de l'application et à la compréhension de son fonctionnement.

Répartition des travaux

Daniel ayant le plus de connaissances en C, il a guidé Damien dans l'exécution des tâches.

Tenant à faire une interface graphique, il s'est occupé de la conception des programmes clients. Damien quant à lui s'est occupé du serveur et notamment de tout ce qui touche directement au jeu. Daniel a bien entendu aussi participé à la conception de certaines fonctionnalités du serveur.

Pour ce qui est de l'implémentation des sockets et des threads c'est Daniel qui s'en est principalement chargé tout en expliquant à Damien le fonctionnement de ces mécanismes et leurs implémentations dans le programme du serveur.

Le protocole de communication a été réfléchi à deux, Daniel ayant besoin de recevoir certaines informations du serveur, et Damien du client. L'objectif étant de trouver le moyen le plus simple possible de transmettre les informations souhaitées du serveur vers le client, et inversement.

Ainsi, la répartition a été plutôt simple et claire, et s'est faite tout naturellement.

Le projet a ainsi été enrichissant pour l'un comme l'autre, permettant de parfaire les connaissances de chacun et de gagner en compétences.

Découpage des modules

L'entièreté des fichiers sources et headers évoqués dans les parties qui suivent sont disponibles dans le dossier *.zip* en annexe du présent rapport.

Découpage du Serveur

Notre serveur est découpé en plusieurs fichiers *sources* et *headers*. Seulement deux fichiers *sources* et un fichier *header* sont destinés au module de communication. Les autres fichiers permettent de gérer la partie, jouer, gérer les comptes, etc.

Par ailleurs, le serveur utilise des *threads*, il en crée un pour chaque client connecté. Il a donc fallu protéger les accès aux données (par exemple la liste des parties, des joueurs, ...) et mettre en place des *mutex* afin de gérer les **accès concurrentiels**. Il n'y a pas d'accès *multi-thread* aux données partagées à l'ensemble des clients car ces accès sont désormais sécurisés.

Gestion

includes.h

Ce fichier *header* regroupe tous les prototypes des fonctions, toutes les constantes, toutes les variables globales et toutes les définitions de structure/énumérations/macros liés au bon fonctionnement du serveur.

account.c

Ce fichier regroupe toutes les fonctions liées à la gestion des comptes.

game.c

Ce fichier regroupe toutes les fonctions liées à la gestion de la partie.

list.c

Ce fichier regroupe toutes les fonctions relatives aux listes (ajout/suppression/initialisation de noeuds, ...) qui sont fréquemment utilisées dans l'ensemble du programme.

Par exemple, elles sont utilisées pour stocker les bateaux, les joueurs, les threads, etc.

login.c

Ce fichier regroupe toutes les fonctions liées à l'authentification d'un utilisateur.

main.c

Ce fichier est le point d'entrée du programme.

randomGrid.c

Ce fichier regroupe toutes les fonctions nécessaires à la génération de la grille de jeu et au placement aléatoire des bateaux sur celle-ci.

Communication

communication.h

Ce fichier *header* regroupe toutes les constantes (type de message) et les structures nécessaires à la communication et à l'envoi de données à travers les sockets non nommées. Ce fichier est commun au serveur et au client.

server.c

Ce fichier regroupe toutes les fonctions permettant d'écrire et de lire au travers des sockets non nommées établies entre le client et le serveur.

clientCallback.c

Ce fichier permet de traiter les messages reçus de la part des clients et d'appeler les bonnes fonctionnalités du serveur (gestion de la partie, gestion des comptes, ...).

Découpage du Client

Notre client est lui aussi découpé en plusieurs fichiers *sources* et *headers*. Certains permettent la gestion de l'interface et des fonctionnalités, et quelques autres seulement de la communication.

Gestion

resource.h

Ce fichier *header* regroupe les ressources externes qui peuvent être utilisées dans un programme. Dans notre cas, il contient l'image en *.ico* de notre application client.

client.h

Ce fichier *header* regroupe tous les prototypes des fonctions, toutes les constantes, toutes les variables globales et toutes les définitions de structure/énumérations liés au bon fonctionnement du client.

adminWindow.c

Ce fichier permet de créer la fenêtre d'accueil de l'administrateur et de définir les fonctionnalités qui y seront liées.

createAccountWindow.c

Ce fichier permet de créer la fenêtre permettant à l'administrateur de créer des comptes, et de définir les fonctionnalités qui y seront liées.

createGameWindow.c

Ce fichier permet de créer la fenêtre permettant à l'administrateur de créer une partie, et de définir les fonctionnalités qui y seront liées.

deleteAccountWindow.c

Ce fichier permet de créer la fenêtre permettant à l'administrateur de supprimer des comptes, et de définir les fonctionnalités qui y seront liées.

gameWindow.c

Ce fichier permet de créer la fenêtre permettant de jouer (avec la grille de la partie, les tableau des scores, le bouton "Tirer") et de définir les fonctionnalités qui y seront liées.

init.c

Ce fichier contient la fonction permettant de définir les spécificités des fenêtres et de lancer la première fenêtre du client via un appel vers la fonction *loginWindowProc()*.

joinGameWindow.c

Ce fichier permet de créer la fenêtre permettant à un joueur de rejoindre une partie, et de définir les fonctionnalités qui y seront liées.

listGameWindow.c

Ce fichier permet de créer la fenêtre permettant à l'administrateur de visualiser la liste des parties, et de définir les fonctionnalités qui y seront liées.

loginWindow.c

Ce fichier permet de créer la fenêtre permettant l'authentification des utilisateurs (administrateur ou joueurs), et de définir les fonctionnalités qui y seront liées. C'est la première fenêtre du client à apparaître.

manageGameWindow.c

Ce fichier permet de créer la fenêtre permettant à l'administrateur gérer une partie (exclusion des joueurs, lancement de la partie, devenir spectateur de la partie), et de définir les fonctionnalités qui y seront liées.

toolsWindow.c

Ce fichier permet de créer la fenêtre permettant de configurer les paramètres réseaux (adresse IP du serveur et port de connexion), et de définir les fonctionnalités qui y seront liées.

userWindow.c

Ce fichier permet de créer la fenêtre d'accueil du joueur et de définir les fonctionnalités qui y seront liées.

waitingWindow.c

Ce fichier permet de créer la fenêtre qui signale que le lancement de la partie est en attente du serveur et de l'administrateur, et de définir les fonctionnalités qui y seront liées.

main.c

Ce fichier est le point d'entrée du programme.

Communication

communication.h

Ce fichier *header* regroupe toutes les constantes (type de message) et les structures nécessaires à la communication et à l'envoi de données à travers les sockets non nommées. Ce fichier est commun au serveur et au client.

client.c

Ce fichier regroupe toutes les fonctions permettant d'écrire et de lire au travers des sockets non nommées établies entre le client et le serveur.

serverCallback.c

Ce fichier permet de traiter les messages reçus de la part du serveur et d'appeler les bonnes fonctionnalités/fenêtres du clients.

Protocole de communication

Ci-dessous sera expliqué et détaillé le protocole de communication.

Celui-ci est relativement triviale car il est similaire pour les deux sens de communication et il se contente d'envoyer un paquet de données qui est une structure appelée **Packet** à travers la socket établie entre le client et le serveur.

Cette structure comporte toutes les informations dont on pourrait avoir besoin pour le bon fonctionnement des divers programmes (serveur et client), ainsi qu'un message permettant d'identifier le rôle des informations contenues dans le paquet.

En effet, un paquet peut être une demande faite au serveur ou une réponse de celui-ci.

La structure *Packet*

La structure Packet permet donc de regrouper les informations qu'on souhaite envoyer à travers la socket.

Elle est composée de 3 éléments, dont 1 très important, qui décrit le type du message et ne peut jamais être vide.

```
typedef struct Packet
{
    int      messageType;
    Packet_Account account;
    Packet_Game game;
} Packet;
```

Packet_Account

C'est une structure qui permet de regrouper toutes les informations liées au compte d'un utilisateur. Autrement dit, c'est dans cette partie qu'on stocke son nom d'utilisateur et/ou son mot de passe si besoin.

Par exemple, cette partie est principalement lors de l'authentification du client.

```
typedef struct Packet_Account
{
    char username[SIZE_USERNAME + 1];
    char password[SIZE_PASSWORD + 1];
} Packet_Account;
```

Packet_Game

C'est une structure qui permet de regrouper toutes les informations liées à une partie.

Cette structure est très souvent utilisée, que ce soit pour la création d'une partie, la suppression de celle-ci ou, plus généralement, pour transmettre les informations liées à une partie en cours.

```
typedef struct Packet_Game
{
    char gamename[SIZE_GAMENAME + 1];
    char grid[SIZE_GRID][SIZE_GRID];
    int nbBoats;
    int nbPlayers;
    int isRunnig;
    Point point;
    int score;
} Packet_Game;
```

On peut remarquer que cette structure utilise une autre structure appelée **Point**. C'est dans cette petite structure qu'on va, par exemple, placer les coordonnées de la case visée.

```
typedef struct Point
{
    int x;
    int y;
} Point;
```

messageType

Ici est placé un numéro défini préalablement dans un fichier header et associé à un nom pour que ce soit plus simple lors de l'utilisation de celui-ci dans le code.

Il s'agit tout simplement d'un numéro servant à la communication et permettant d'identifier le type du paquet envoyé à travers la socket.

Ces numéros doivent être les mêmes pour le client et pour le serveur, sinon des problèmes surviendront dans la communication entre les deux.

Dans la partie suivante sont détaillés ce que chaque type de message permet de faire.

Les types de messages

MESSAGE_TYPE_LOGIN_REQUEST

Communication : Client → Serveur

Objectif :

Permet de faire une demande d'authentification.

Le paquet contient le nom d'utilisateur (*username*) et le mot de passe (*password*) entrés dans la fenêtre de connexion.

MESSAGE_TYPE_LOGIN_REPLY_ADMIN_SUCCESS

Communication : Serveur → Client

Objectif :

Message de réponse de la part du serveur signifiant que le couple identifiant/mot de passe correspond bien à celui de l'**administrateur** et que l'utilisateur est autorisé à se connecter.

MESSAGE_TYPE_LOGIN_REPLY_SUCCESS

Communication : Serveur → Client

Objectif :

Message de réponse de la part du serveur signifiant que le couple identifiant/mot de passe correspond bien à celui d'un **joueur** et que l'utilisateur est autorisé à se connecter.

MESSAGE_TYPE_LOGIN_REPLY_BAD_USERNAME

Communication : Serveur → Client

Objectif :

Message de réponse de la part du serveur signifiant que le l'identifiant fourni lors de l'authentification n'est pas bon (utilisateur inexistant).

MESSAGE_TYPE_LOGIN_REPLY_BAD_PASSWORD

Communication : Serveur → Client

Objectif :

Message de réponse de la part du serveur signifiant que le mot de passe fourni lors de l'authentification est incorrect.

MESSAGE_TYPE_CLOSE_CONNECTION

Communication : Client → Serveur

Objectif :

Ce message est envoyé par le client lors de la déconnexion ou de la fermeture de la fenêtre pour signaler au serveur qu'il peut terminer le thread lié à cet utilisateur.

MESSAGE_TYPE_ACCOUNT_CREATE_REQUEST

Communication : Client → Serveur

Objectif :

Message envoyé lorsque l'administrateur crée un nouveau compte utilisateur. Le paquet contenant ce *messageType* contient aussi les informations liées à la création du nouveau compte (*username* et *password*).

MESSAGE_TYPE_ACCOUNT_CREATE_ALREADY_EXIST

Communication : Serveur → Client

Objectif :

Message permettant de signaler à l'administrateur qu'il ne peut pas créer le nouveau compte souhaité car un compte existe déjà avec ce nom d'utilisateur (*username*).

MESSAGE_TYPE_ACCOUNT_CREATE_OK

Communication : Serveur → Client

Objectif :

Message permettant de signaler à l'administrateur que le nouveau compte a été créé avec succès.

MESSAGE_TYPE_ACCOUNT_DELETE_REQUEST

Communication : Client → Serveur

Objectif :

Message permettant de faire une demande au serveur de suppression d'un compte. Le paquet envoyé contient aussi le nom d'utilisateur (*username*) du compte à supprimer.

MESSAGE_TYPE_INTERNAL_ERROR

Communication : Serveur → Client

Objectif :

Message envoyé par le serveur dans le cas où une erreur (peu importe laquelle) se produit. Cela permet de signaler au client qu'il y a eu un problème du côté du serveur.

MESSAGE_TYPE_ACCOUNT_DELETE_SUCCES

Communication : Serveur → Client

Objectif :

Message permettant de confirmer à l'administrateur que le compte a bien été supprimé.

MESSAGE_TYPE_ACCOUNT_DELETE_BAD_USERNAME

Communication : Serveur → Client

Objectif :

Message permettant de signaler à l'administrateur que le compte qu'il souhaite supprimer n'existe pas et qu'il y a donc une erreur sur le nom de l'utilisateur (*username*).

MESSAGE_TYPE_GAME_ENUMERATE_REQUEST

Communication : Client → Serveur

Objectif :

Message demandant au serveur de renvoyer la liste des parties. Ce message est envoyé lorsqu'un utilisateur clique sur "Liste des parties", que ce soit pour rejoindre une partie si c'est un joueur, ou pour la gérer si c'est un administrateur.

MESSAGE_TYPE_GAME_CREATE_REQUEST

Communication : Client → Serveur

Objectif :

Message envoyé au serveur lorsqu'un administrateur souhaite créer une partie. Le paquet envoyé contient toutes les informations nécessaires (nom de la partie, nombre de joueurs, nombre de bateaux).

MESSAGE_TYPE_GAME_CREATE_ALREADY_EXIST

Communication : Serveur → Client

Objectif :

Message signalant à l'administrateur qu'une partie porte déjà le nom de celle qu'il tente de créer.

MESSAGE_TYPE_GAME_CREATE_SUCCESS

Communication : Serveur → Client

Objectif :

Message signalant à l'administrateur que la nouvelle partie a été créée avec succès.

MESSAGE_TYPE_GAME_NUMBER_REPLY

Communication : Serveur → Client

Objectif :

Message de réponse de la part du serveur quant à la demande de renvoyer la liste des parties. Permet de remplir la liste des parties que peut voir un utilisateur sur son interface client. Il y a autant de paquet envoyé avec ce message qu'il y a de parties et chaque paquet contient le nom de la partie.

MESSAGE_TYPE_GAME_JOIN_REQUEST

Communication : Client → Serveur

Objectif :

Message envoyé par le client lorsqu'un joueur veut rejoindre une partie afin de demander au serveur de l'ajouter à la liste des participants de ladites partie.

MESSAGE_TYPE_GAME_JOIN_OK

Communication : Serveur → Client

Objectif :

Message de réponse du serveur signalant à l'utilisateur qu'il a bien rejoint la partie.

MESSAGE_TYPE_GAME_JOIN_FULL

Communication : Serveur → Client

Objectif :

Message de réponse du serveur signalant à l'utilisateur qu'il ne peut pas rejoindre la partie car elle est complète (il n'y a plus de place).

MESSAGE_TYPE_KEEP_ALIVE

Communication : Client → Serveur

Objectif :

Message envoyé par le client au serveur et permettant au serveur de savoir si le client est toujours actif. Ce message est envoyé toutes les 5 secondes par le client. Dans le cas où le serveur n'en reçoit plus pendant plus de 10 secondes, il coupe la connexion avec lui (fermeture de la socket et du thread).

MESSAGE_TYPE_GAME_PLAYER_NUMBER_REQUEST

Communication : Client → Serveur

Objectif :

Message envoyé par l'administrateur pour voir la liste des joueurs actuellement connectés à la partie qu'il souhaite gérer.

MESSAGE_TYPE_GAME_PLAYER_EJECT

Communication : Client → Serveur

Objectif :

Message envoyé par l'administrateur pour demander à éjecter un joueur d'une partie.

Communication : Serveur → Client

Objectif :

Message aussi utilisé par le serveur pour signaler au joueur exclu qu'il a été éjecté de la partie.

MESSAGE_TYPE_GAME_PLAYER_NUMBER_REPLY

Communication : Serveur → Client

Objectif :

Message de réponse du serveur permettant d'envoyer à l'administrateur la liste des joueurs de la partie qu'il gère actuellement. Il y aura autant de paquet envoyé contenant ce message que de joueurs dans la partie et chaque paquet contiendra le nom d'utilisateur (*username*) du joueur.

MESSAGE_TYPE_GAME_START_REQUEST

Communication : Client → Serveur

Objectif :

Message envoyé par l'administrateur pour demander au serveur de démarrer la partie.

MESSAGE_TYPE_GAME_START_SUCCESS

Communication : Serveur → Client

Objectif :

Message de réponse du serveur envoyé à l'administrateur et aux joueurs pour leur signaler que la partie a bien démarré.

MESSAGE_TYPE_GAME_START_NOT_FULL

Communication : Serveur → Client

Objectif :

Message de réponse du serveur envoyé à l'administrateur pour lui signaler qu'il ne peut pas lancer la partie car elle n'est pas complète (il n'y a pas assez de joueurs).

MESSAGE_TYPE_GAME_JOIN_ERROR

Communication : Serveur → Client

Objectif :

Message envoyé par le serveur à un joueur pour lui signaler qu'il est déjà dans la partie.

MESSAGE_TYPE_USER_ENUMERATE_REQUEST

Communication : Client → Serveur

Objectif :

Message envoyé par l'administrateur pour obtenir la liste des joueurs/comptes dans le cadre de la suppression d'un compte.

MESSAGE_TYPE_GAME_PLAYER_EJECT_SUCCESS

Communication : Serveur → Client

Objectif :

Message de réponse du serveur pour signaler à l'administrateur qu'un joueur a bien été éjecté de la partie.

MESSAGE_TYPE_GRID_REQUEST

Communication : Client → Serveur

Objectif :

Message envoyé par le joueur pour demander au serveur de lui retourner l'état actuel de la grille de jeu (état visible par les joueurs).

MESSAGE_TYPE_SEND_SHOT

Communication : Client → Serveur

Objectif :

Message envoyé par le joueur pour donner au serveur les coordonnées de la case sur laquelle il vient de tirer.

MESSAGE_TYPE_GRID_REPLY

Communication : Serveur → Client

Objectif :

Message de réponse du serveur permettant d'envoyer à un joueur la grille actuelle du jeu (état du jeu à jour visible par le joueur).

MESSAGE_TYPE_PLAYER_IN_GAME_RUNNING

Communication : Serveur → Client

Objectif :

Message de réponse du serveur pour signaler au joueur lors de sa reconnexion qu'il est dans une partie en cours (donc commencée).

MESSAGE_TYPE_PLAYER_IN_GAME

Communication : Serveur → Client

Objectif :

Message de réponse du serveur pour signaler au joueur lors de sa reconnexion qu'il est dans une partie en attente de lancement/démarrage.

MESSAGE_TYPE_ITS_YOUR_TURN

Communication : Serveur → Client

Objectif :

Message envoyé par le serveur au joueur dont c'est le tour de jouer. Cela permet d'activer le bouton "Tirer".

MESSAGE_TYPE_ITS_NOT_YOUR_TURN

Communication : Serveur → Client

Objectif :

Message envoyé par le serveur aux joueurs dont ce n'est pas le tour de jouer. Cela permet de désactiver le bouton "Tirer".

MESSAGE_TYPE_TURN_REQUEST

Communication : Client → Serveur

Objectif :

Message envoyé par un joueur pour demander au serveur si c'est son tour de jouer.

MESSAGE_TYPE_YOU_WIN

Communication : Serveur → Client

Objectif :

Message envoyé par le serveur au joueur qui a gagné. Dans le cas d'un ex-aequo, ce message est envoyé aux divers gagnants.

MESSAGE_TYPE_YOU_LOSE

Communication : Serveur → Client

Objectif :

Message envoyé par le serveur aux joueurs qui ont perdu.

MESSAGE_TYPE_USER_ENUMERATE_REPLY

Communication : Serveur → Client

Objectif :

Message de réponse de la part du serveur permettant de renvoyer à l'administrateur la liste des comptes (dans le cadre de la suppression d'un compte). Il y a autant de paquet envoyé contenant ce message qu'il y a de comptes et chaque paquet contient le nom de l'utilisateur (*username*).

MESSAGE_TYPE_GAME_SHOW_ADMIN

Communication : Client → Serveur

Objectif :

Message envoyé par l'administrateur pour signaler au serveur qu'il souhaite regarder une partie en cours. Le nom de la partie est fourni dans le paquet envoyé au serveur.

MESSAGE_TYPE_SHOW_ADMIN_REPLY

Communication : Serveur → Client

Objectif :

Message de réponse du serveur pour signaler à l'administrateur qu'il a bien pris en compte sa demande et qu'il peut regarder la partie. Le programme client va donc lancer la fenêtre de visualisation de la partie.

MESSAGE_TYPE_GRID_ADMIN_REQUEST

Communication : Client → Serveur

Objectif :

Message envoyé par l'administrateur pour demander au serveur de lui fournir l'état actuel complet de la grille du jeu (il verra tous les coups et l'emplacement des bateaux).

MESSAGE_TYPE_GRID_ADMIN_REPLY

Communication : Serveur → Client

Objectif :

Message de réponse du serveur permettant de renvoyer à l'administrateur la grille du jeu dans son état actuel et dans sa version complète (tous les coups et les emplacements des bateaux sont visibles).

MESSAGE_TYPE_SCORE

Communication : Serveur → Client

Objectif :

Message envoyé par le serveur aux joueurs pour signaler la mise à jour du score. Ce message vient après MESSAGE_TYPE_FIRST_SCORE (visible ci-dessous). Tous les paquets envoyés concernant les scores comportent les scores à jour ainsi que le nom du joueur.

MESSAGE_TYPE_FIRST_SCORE

Communication : Serveur → Client

Objectif :

Message envoyé par le serveur aux joueurs signalant que c'est le premier élément de la liste des scores de la partie qui sont actuellement envoyés. Signaler que c'est le premier élément permet de réinitialiser le tableau des scores pour ensuite pouvoir afficher les scores à jour.

MESSAGE_TYPE_GAME_STOP_SHOW_ADMIN

Communication : Client → Serveur

Objectif :

Message envoyé par l'administrateur pour signaler au serveur qu'il souhaite arrêter de regarder la partie. Le paquet envoyé contient le nom de la partie que l'utilisateur regarde.

Mode d'emploi

Ci-dessous sont spécifiées les explications d'utilisation des différents programmes et leurs fonctionnalités.

Serveur

Le serveur ne comporte pas de fonctionnalités particulières servant à interagir directement avec lui.

Pour le démarrer, il suffit simplement de l'exécuter comme n'importe quel autre programme si vous êtes sur une machine Windows ou via un terminal si vous êtes sur une machine Linux.

Sous linux

```
[daniel@SERVEUR] - [~/projetc/Server_2.8_Full_Comments]
$ ./server
#####
##### Bataille navale #####
#####
# Daniel ZEITOUN -- Damien MICHAUDEL #
#####
SERVEUR - LINUX

Attente d'un client ...
```

Sous Windows

```
#####
##### Bataille navale #####
#####
# Daniel ZEITOUN -- Damien MICHAUDEL #
#####
VERSION : 2.8

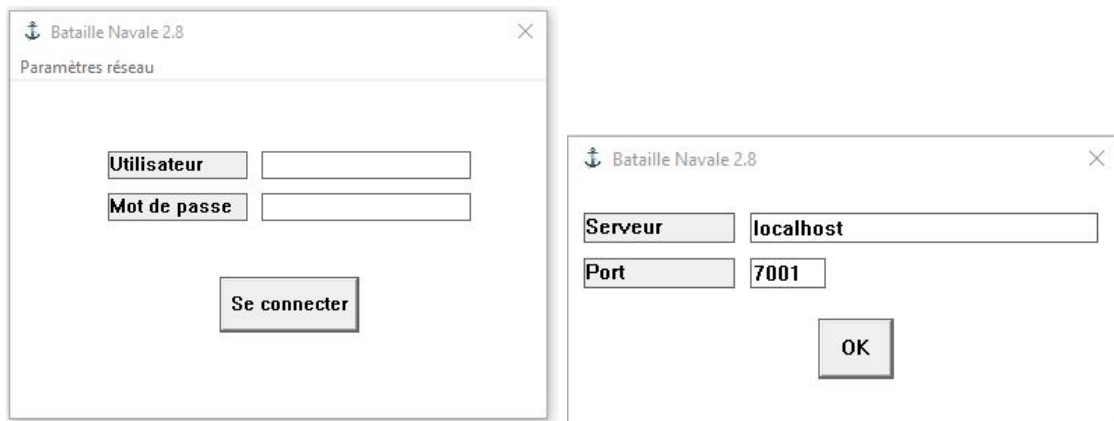
SERVEUR - WINDOWS

Attente d'un client ...
```

Client

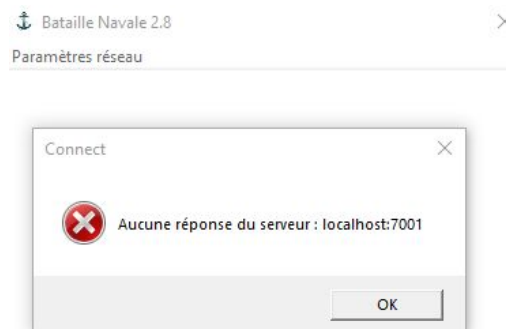
Le client, quant à lui, est un programme exclusivement Windows. Pour le démarrer, il suffit simplement de double-cliquer dessus.

Une fenêtre d'authentification va alors s'ouvrir. En cliquant sur le bouton "Paramètres réseau", vous aurez la possibilité de choisir l'adresse IP ainsi que le port sur lequel se trouve votre serveur de jeu si jamais vous avez une configuration réseau particulière ou que vous avez modifié le code du serveur.



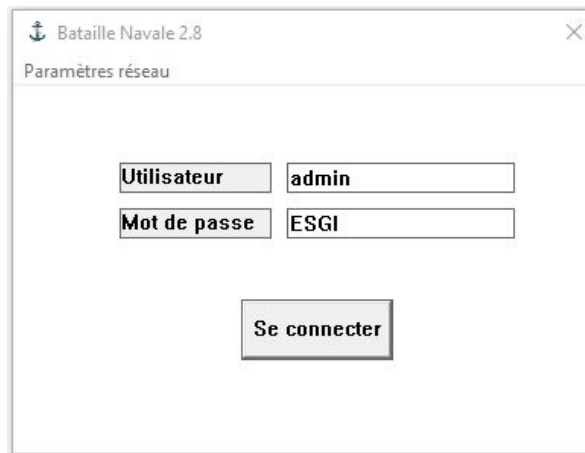
Au niveau de la connexion, vous avez deux types de compte : le compte administrateur, et les comptes des joueurs.

Dans le cas où les paramètres réseaux ont été mal configurés, ce message d'erreur apparaît.



Admin

Pour se connecter en tant qu'administrateur, il suffit de rentrer le couple identifiant/mot de passe correspondant à celui de l'administrateur et de cliquer sur "Se connecter".



Bataille Navale 2.8

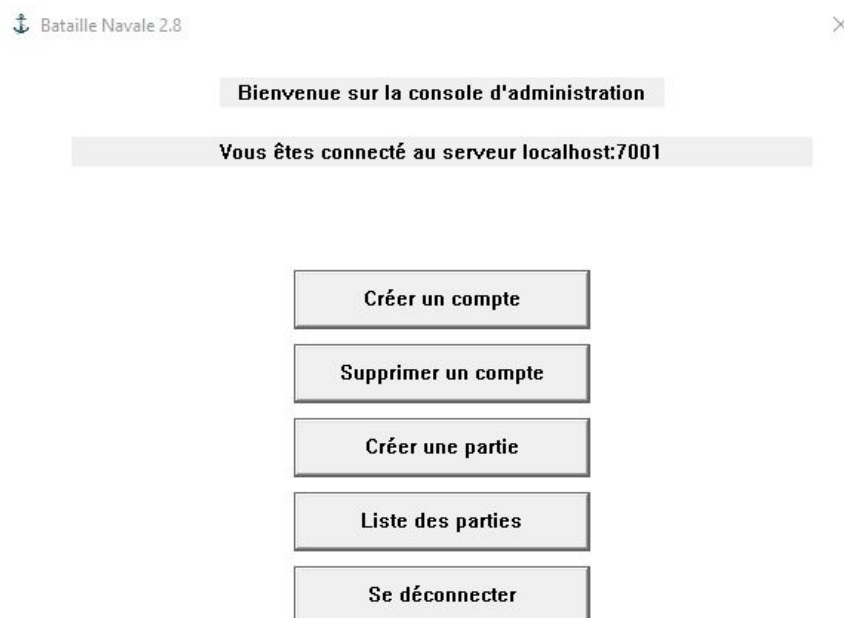
Paramètres réseau

Utilisateur admin

Mot de passe ESGI

Se connecter

Vous tomberez ensuite sur l'interface d'administration qui propose un certains nombre de fonctionnalités et spécifiant l'adresse sur laquelle se trouve le serveur ainsi que le port.



Bataille Navale 2.8

Bienvenue sur la console d'administration

Vous êtes connecté au serveur localhost:7001

Créer un compte

Supprimer un compte

Créer une partie

Liste des parties

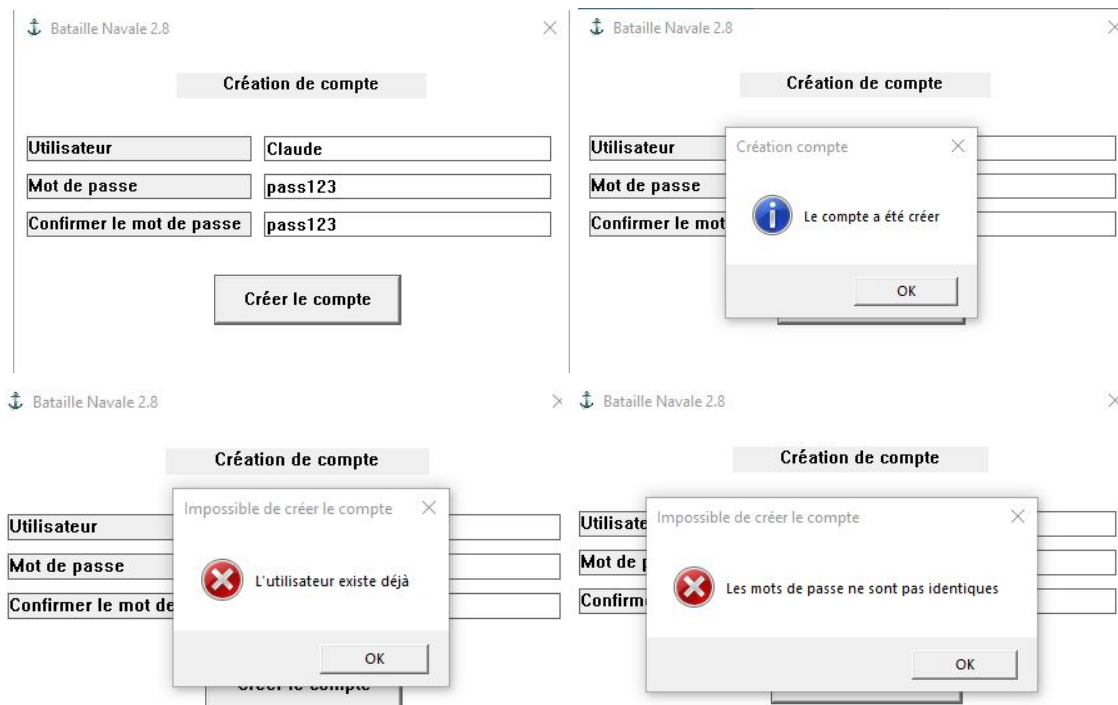
Se déconnecter

Création d'un compte

Pour créer un compte, on clique sur le bouton "Créer un compte" qui va nous afficher la fenêtre correspondante.

Dans cette fenêtre on spécifie le nom de l'utilisateur ainsi que son mot de passe qu'il faudra bien entendu confirmer en le tapant une seconde fois.

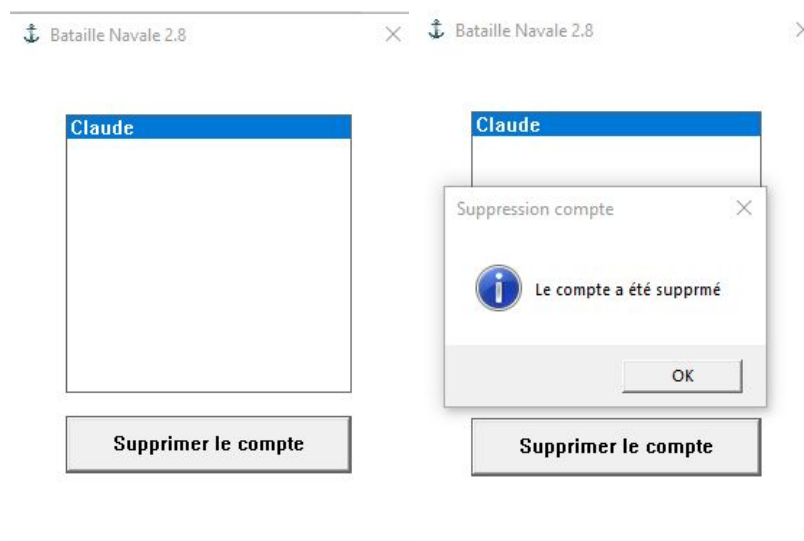
Il ne nous reste plus qu'à cliquer sur "Créer le compte" et une petite fenêtre popup s'affiche vous indiquant si oui ou non le compte a bien été créé. Auquel cas, c'est que l'utilisateur existe déjà, ou que les deux mots de passe ne correspondent pas.



Supprimer un compte

Pour supprimer un compte, on clique sur le bouton “Supprimer un compte” qui va nous afficher la fenêtre correspondante.

On sélectionne ensuite dans la liste l'utilisateur qu'on souhaite supprimer et on clique sur “Supprimer le compte”. Une petite fenêtre popup nous confirmera la bonne suppression.



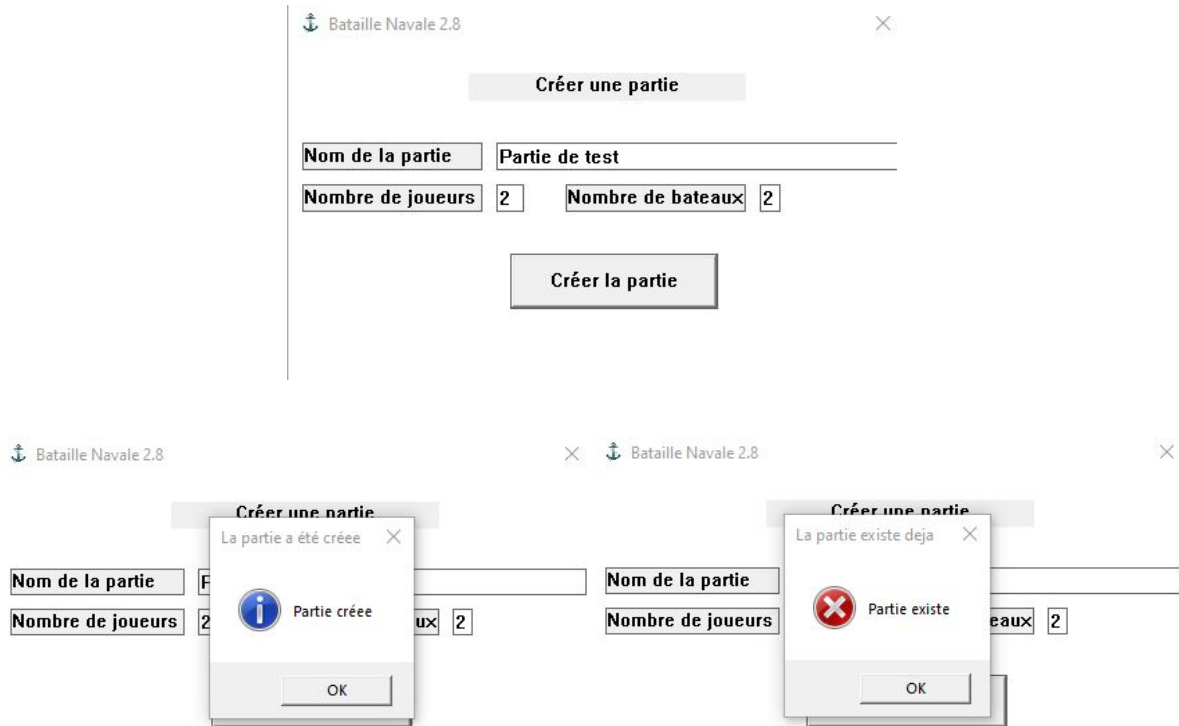
Créer une partie

Pour créer une partie, on clique sur le bouton “Créer une partie” qui va nous afficher la fenêtre correspondante.

On peut ensuite y saisir le nom qu'on souhaite donner à la partie et qui sera visible par le joueurs.

On peut aussi saisir le nombre de joueurs qui pourront rejoindre notre partie (1 minimum, 99 maximum), ainsi que le nombre de bateaux qui seront positionnés sur la grille (1 minimum, 9 maximum). A noter que les bateaux sont positionnés aléatoirement sur la grille et que leur taille aussi est aléatoire.

On termine en cliquant sur le bouton "Créer la partie". Une petite fenêtre popup s'affiche pour nous indiquer si la partie a bien été créée ou non. Auquel cas, c'est qu'une partie existante porte déjà ce nom.



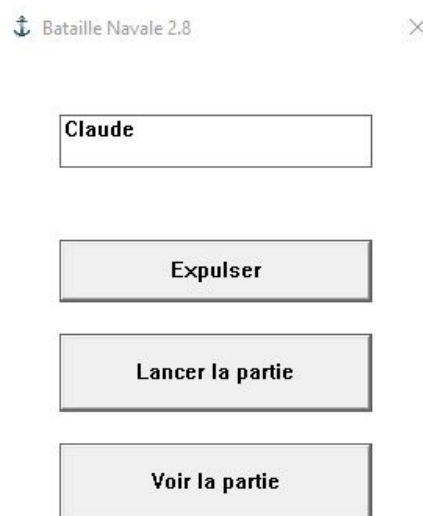
Liste des parties

Pour afficher la liste des parties, on clique sur le bouton "Liste des parties" qui va nous afficher la fenêtre correspondante.

On pourra y voir la liste des parties qui ont été créées et qui sont en attente de commencement ou en cours. Les parties terminées ne sont pas affichées car une fois finies elles n'existent plus.



On peut accéder aux informations de la partie en la sélectionnant et en cliquant sur le bouton “Gérer la partie”. On obtient une nouvelle fenêtre nous présentant la liste des utilisateurs ayant rejoint la partie ainsi que quelques fonctionnalités.



Le bouton “Expulser” permet d’exclure un joueur préalablement sélectionné dans la liste. Une fois l’action effectuée, il n’y pas de message qui apparait mais la fenêtre est actualisée. L’utilisateur, quant à lui, recevra un message le notifiant de son expulsion et il pourra de nouveau sélectionner une partie à rejoindre.



Comme on peut le constater, une fois le joueur exclu la liste est vide.

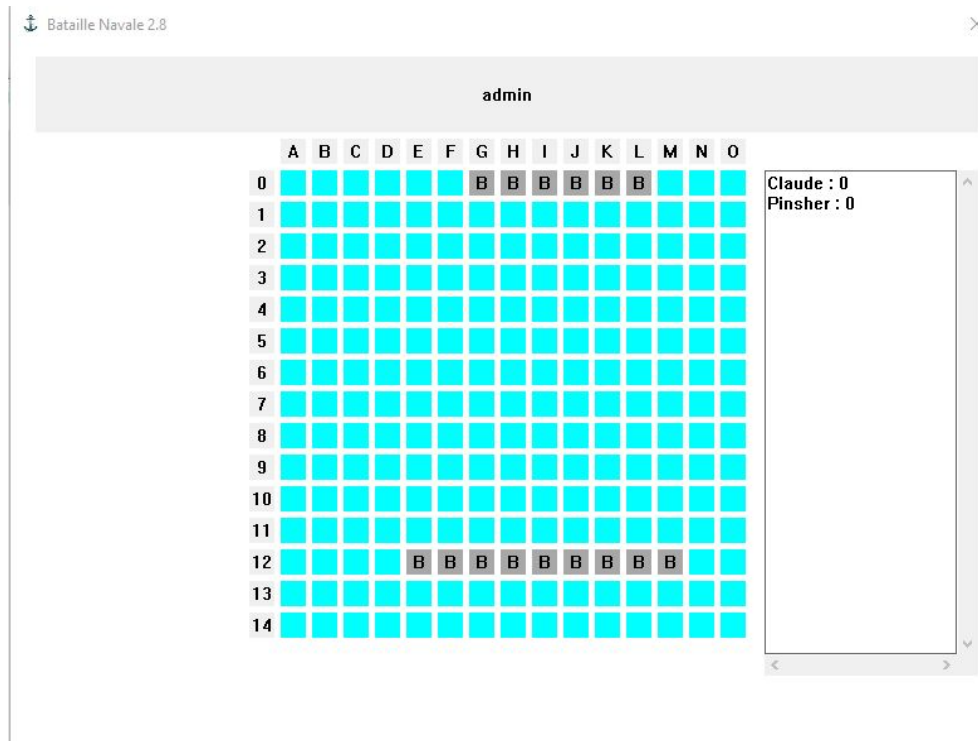
Par ailleurs, on ne peut pas démarrer une partie qui est vide ou pas encore pleine. De plus, tous les joueurs qui n'ont pas été exclu sont considérés comme acceptés par l'administrateur au moment de lancer la partie.

Lorsque le nombre maximum de joueur pouvant rejoindre une partie a été atteint, on peut cliquer sur "Lancer la partie" pour qu'elle démarre.

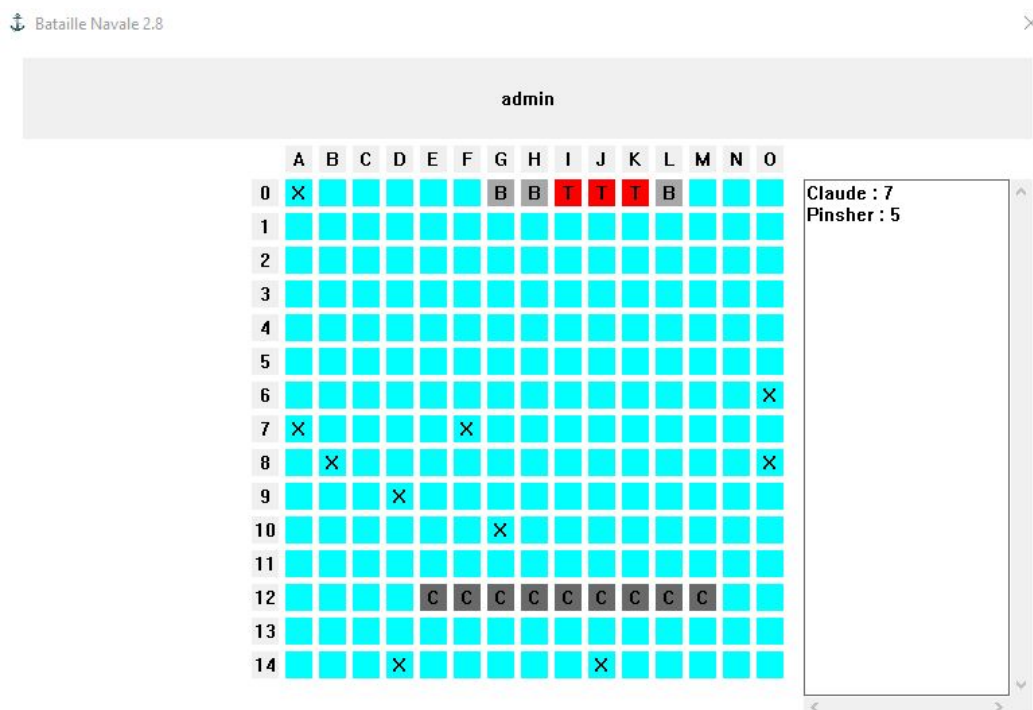
Les joueurs auront alors accès à la grille de jeu affichant les cases sur lesquelles ils peuvent tirer et les scores en cours.



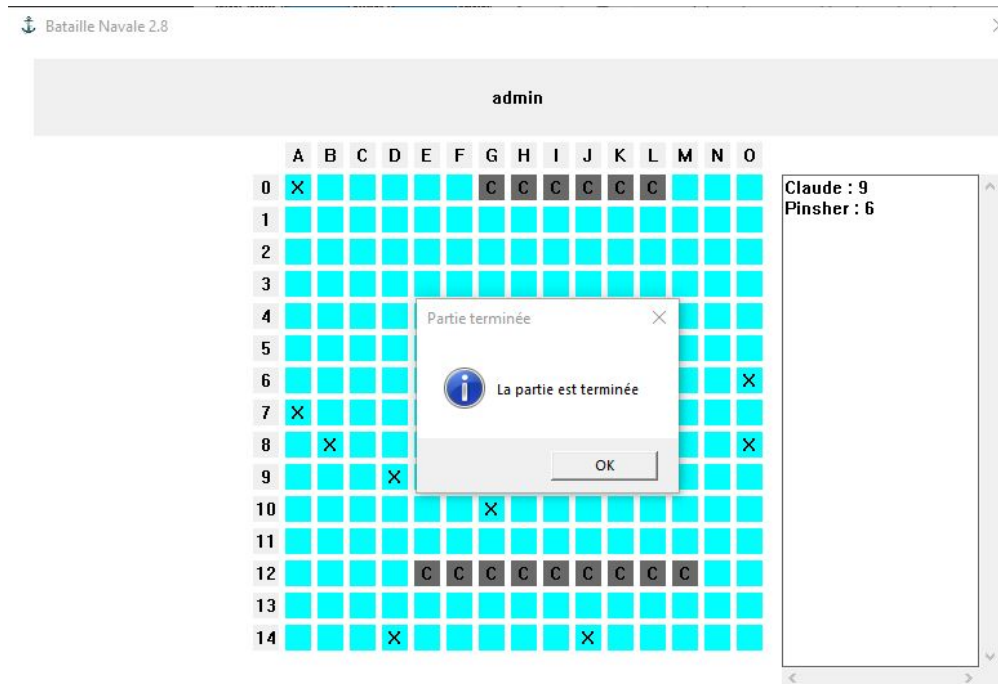
L'administrateur peut, s'il le souhaite, observer la partie en temps réel. Pour cela il lui suffit de cliquer sur le bouton "Voir la partie". Il aura accès au même genre d'interface que les joueurs à la différence près que lui verra où sont placés les bateaux et où on été tirés des coups. Il ne pourra bien évidemment pas interagir avec la grille de jeu. Les scores sont mis à jour après chaque coup porté par un joueur.



Un exemple de la grille après que quelques coups ont été joués.
 Les cases non touchées des bateaux sont marquées par un B, celles touchées sont marquées par un T. Les cases des bateaux coulés sont marquées par C.
 Les coups dans l'eau sont notés avec un X.
 Un code couleur est associé aux différents états des cases.



A la fin d'une partie, lorsque tous les bateaux ont été coulés, une fenêtre popup s'affiche pour le signaler à l'administrateur. Il pourra voir sur le côté l'état final des scores.



Une fois que l'on clique sur "OK", on revient à la fenêtre d'accueil.

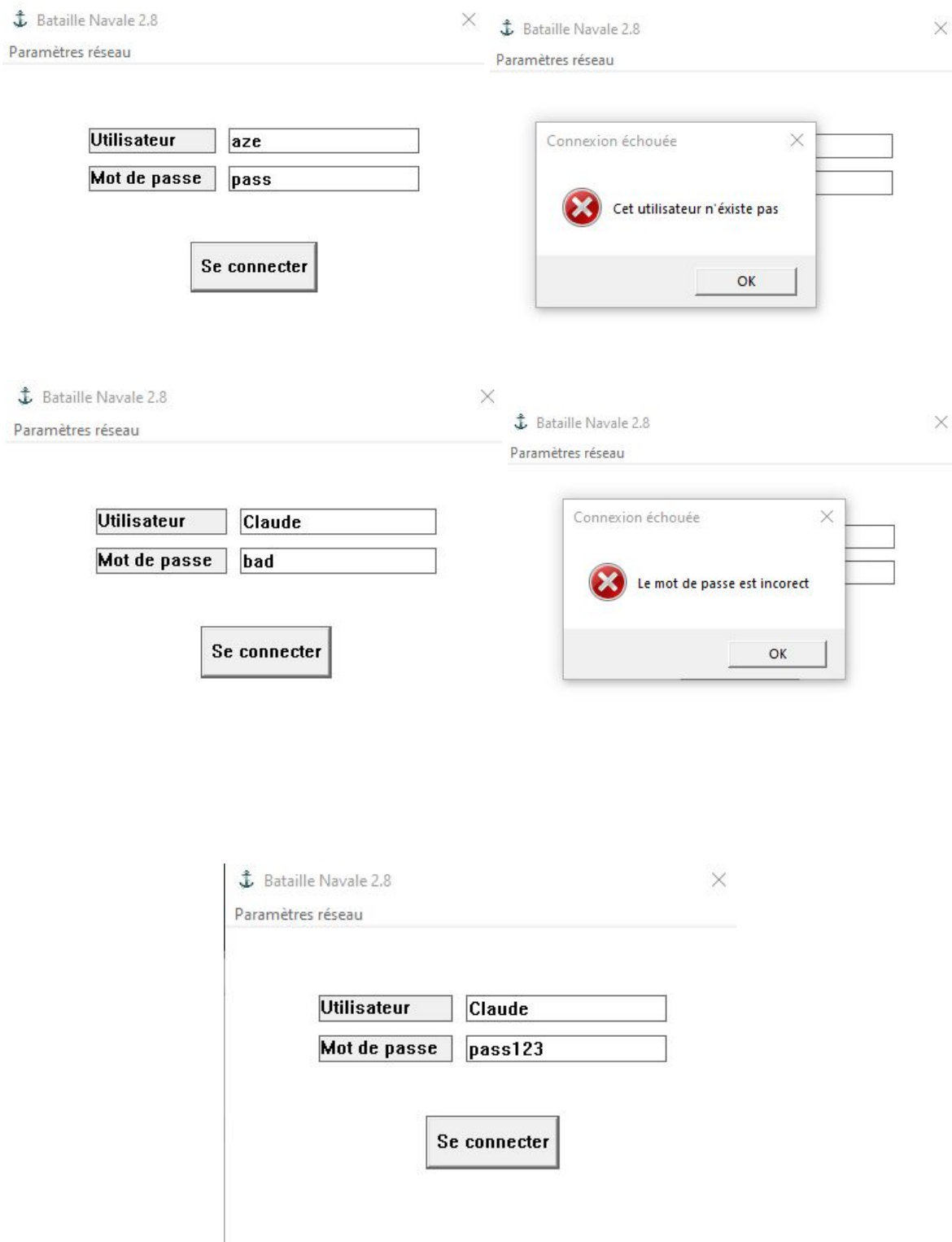
Se déconnecter

Ce bouton sert à la déconnexion, une fois cliqué, l'utilisateur est renvoyé sur la fenêtre d'authentification.

Joueur

Pour se connecter il suffit de rentrer le couple identifiant/mot de passe qui nous a été attribué par l'administrateur.

Comme pour la connexion en tant qu'administrateur, un message d'erreur peut s'afficher si l'identifiant ou le mot de passe est incorrect.



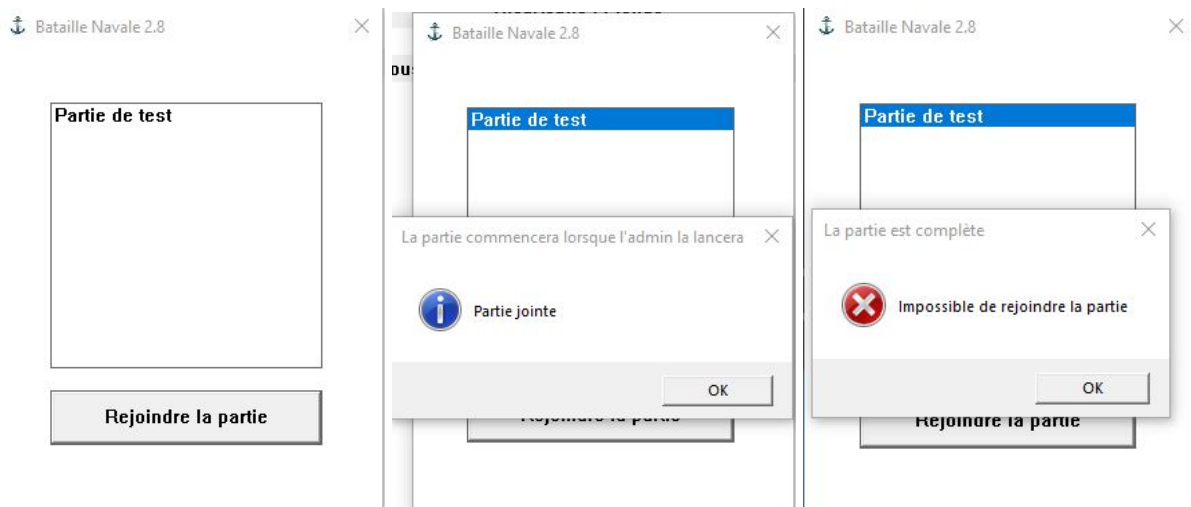
Si on tente de se connecter avec un utilisateur valide, on obtiendra l'interface ci-dessous. Comme sur celle de l'administrateur, on peut voir l'adresse IP du serveur ainsi que le port sur lequel nous sommes connecté.

Bienvenue : Claude**Vous êtes connecté au serveur localhost:7001****Rejoindre une partie****Se déconnecter**

Rejoindre une partie

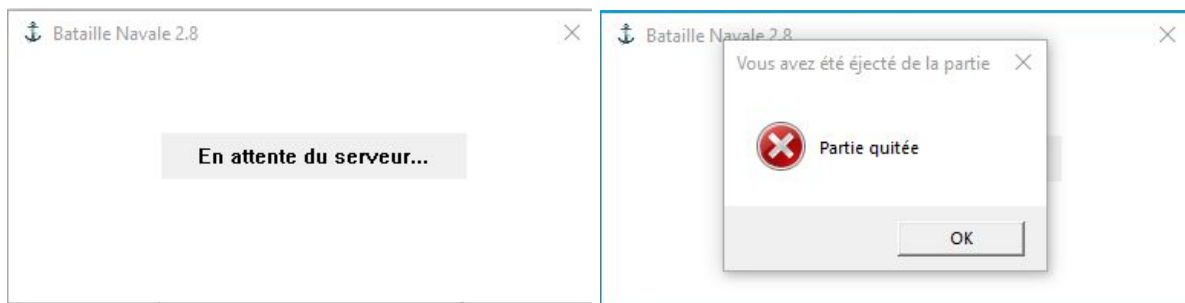
L'interface ne nous propose que de rejoindre une partie, pour cela on clique sur le bouton et une fenêtre s'affiche proposant les parties pas encore commencées.

Il suffit de sélectionner la partie qu'on souhaite rejoindre et de cliquer sur "Rejoindre la partie". Une petite fenêtre popup nous confirmera que nous avons bien rejoint la partie, ou nous informera si c'est impossible.

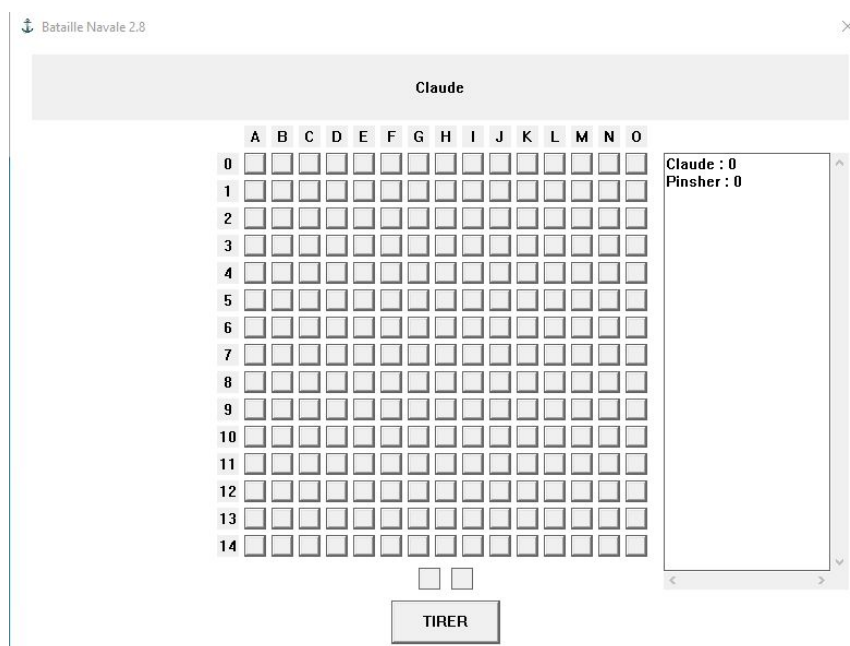


Une fois que nous avons rejoint la partie, la seule fenêtre à disposition devient un message d'attente. Ce message reste tant que l'administrateur n'a pas lancé la partie ou qu'il ne nous a pas exclu.

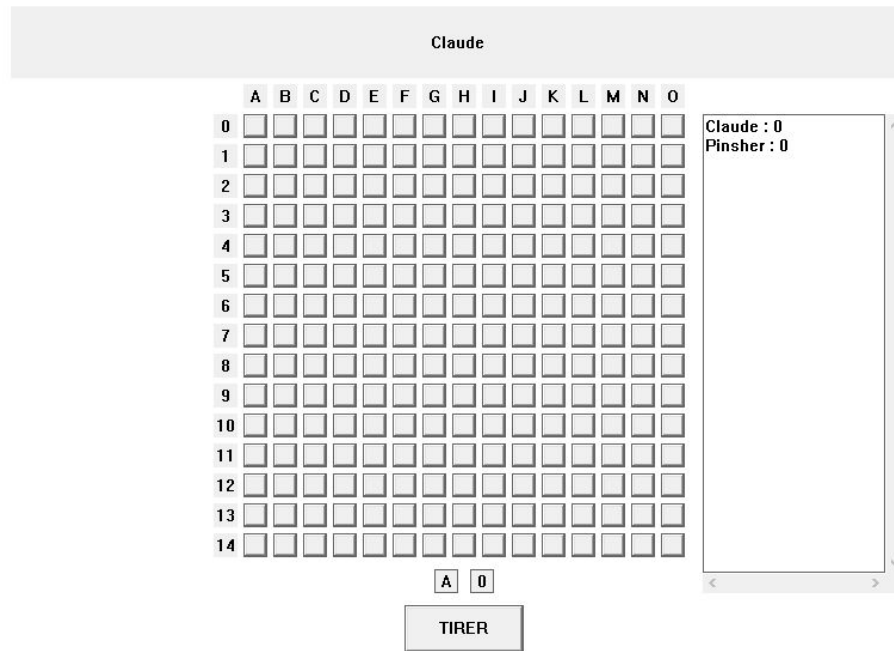
Si nous sommes exclu de la partie, nous sommes renvoyé vers l'interface d'accueil.



Lorsque l'admin lance la partie, la fenêtre du jeu s'affiche automatiquement chez les clients.



Pour sélectionner la case sur laquelle on veut tirer, il suffit de cliquer dessus. Les coordonnées seront mises à jour dans les 2 cellules au-dessus du bouton "TIRER", il ne reste qu'à cliquer dessus pour envoyer au serveur les coordonnées de la cible.



Le serveur nous répond ensuite :

- Si c'est un coup dans l'eau alors il ne se passe rien, la case reste sous forme de bouton, et les scores ne bougent pas ;
- Si c'est un coup qui touche un bateau, alors la case n'est plus cliquable et on peut voir apparaître un T sur fond rouge, les scores sont mis à jour ;
- Si le coup fait couler un bateau, on verra apparaître un C sur fond gris, les scores sont mis à jour.

Le bouton TIRER est alors grisé et désactivé (on ne peut plus cliquer dessus) car ce n'est plus notre tour de joueur et le restera jusqu'à ce que ce soit de nouveau à nous de tirer.



A la fin de la partie, lorsque tous les bateaux ont été coulés, les scores sont mis à jour une dernière fois et une petite fenêtre popup apparaît pour indiquer au joueur s'il a gagné ou non.

Claude

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0							C	C	C	C	C				
1															
2															
3															
4															
5															
6															
7															
8															
9															
10															
11															
12							C	C	C	C	C	C	C		
13															
14															

Bravo

Bravo, tu as gagné

Claude : 9
 Pinsher : 6

Pinsher

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0							C	C	C	C	C				
1															
2															
3															
4															
5															
6															
7															
8															
9															
10															
11															
12							C	C	C	C	C	C	C		
13															
14															

Perdu

Domage tu as perdu

Claude : 9
 Pinsher : 6

Démonstration

Les environnements liés à la démonstration ci-contre sont les suivants :

- Pour le serveur, nous avons utilisé une machine virtuelle sur lequel nous avons installé un système d'exploitation **Debian 10.4** ;
 - *N.B. Le serveur est cross-plateforme, et nous avons aussi utilisé lors de nos tests de développement une machine Windows 10.*
- Pour le client, nous avons utilisé une machine **Windows 10**.
 - *N.B. Le programme client ayant été développé avec Visual Studio Community, il est nécessaire d'avoir vcruntime140.dll (il est théoriquement déjà installé sur Windows).*
 - *N.B.2. Le programme peut être bloqué par Windows car il ne connaît pas la source.*

On vérifie l'adresse IP du serveur, car il faudra la communiquer aux joueurs pour qu'ils puissent s'y connecter.

```
[daniel@SERVEUR]-(~/projetc/Server_2.8_Full_Comments)
$ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:18:bb:73 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.70/24 brd 192.168.0.255 scope global ens33
        valid_lft forever preferred_lft forever
    inet6 2a01:e0a:5b0:220:20c:29ff:fe18:bb73/64 scope global dynamic mngtmpaddr
        valid_lft 85996sec preferred_lft 85996sec
    inet6 fe80::20c:29ff:fe18:bb73/64 scope link
        valid_lft forever preferred_lft forever
```

On démarre le serveur.

```
[daniel@SERVEUR]-(~/projetc/Server_2.8_Full_Comments)
$./server
#####
##### Bataille navale #####
#####
# Daniel ZEITOUN -- Damien MICHAUDEL #
#####
SERVEUR - LINUX

Attente d'un client ...
```

Chaque client doit configurer les paramètres réseaux liés au serveur.



On se connecte en tant qu'administrateur pour créer les clients à la partie.



On a accès à l'interface d'administration, on va commencer par créer les comptes.

Bienvenue sur la console d'administration

Vous êtes connecté au serveur 192.168.0.70:7001

Créer un compte

Supprimer un compte

Créer une partie

Liste des parties

Se déconnecter

On clique sur le bouton “Créer un compte” et on crée le compte du premier joueur : Claude.

⚓ Bataille Navale 2.8

×

Création de compte

Utilisateur	Claude
Mot de passe	pass123
Confirmer le mot de passe	pass123

Créer le compte

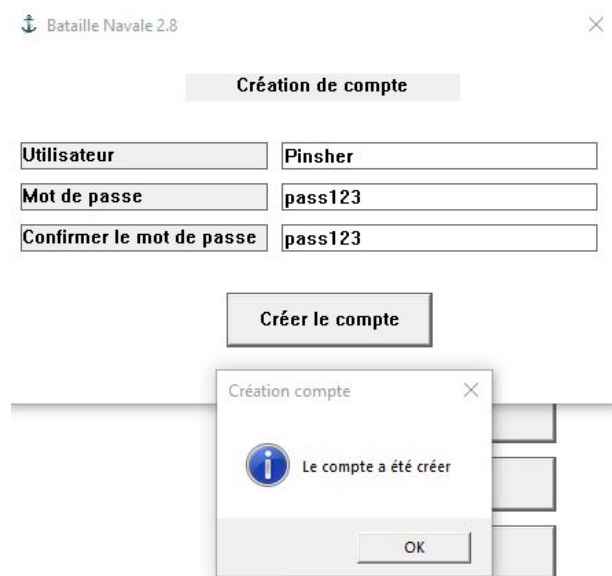
Création compte

×

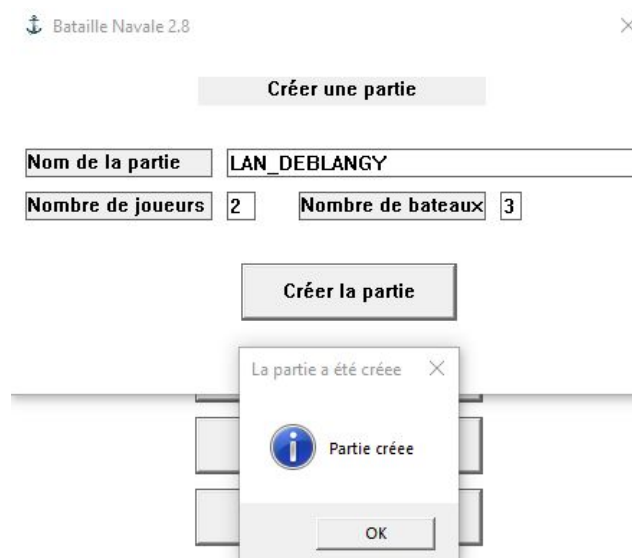
Le compte a été créé

OK

On clique sur le bouton “Créer un compte” et on crée le compte de notre second joueur : Pinsher.



On clique sur le bouton “Créer une partie”, on la nomme comme on le souhaite, et on configure le nombre de joueurs et de bateaux souhaité.
Dans le cas de notre test, il y a 2 joueurs qui pourront rejoindre la partie (les deux joueurs créés précédemment) et il y aura 3 bateaux à couler pour la terminer.
Le placement et la taille des bateaux seront aléatoires.



Maintenant que la partie est créée, les joueurs peuvent se connecter et rejoindre la partie.
On commence par le joueur Claude, il se connecte avec son couple identifiant/mot de passe.

Bataille Navale 2.8

Paramètres réseau

Utilisateur	Claude
Mot de passe	pass123

Se connecter

Il a ensuite accès à son interface.

Bataille Navale

Bienvenue : Claude

Vous êtes connecté au serveur 192.168.0.70:7001

Rejoindre une partie

Se déconnecter

On clique sur le bouton “Rejoindre une partie”, et on obtient la liste des parties.
On rejoint la partie créée pour le test en la sélectionnant et en cliquant sur le bouton en bas de la fenêtre.

Bataille Navale 2.8

LAN_DEBLANGY

La partie commencera lorsque l'admin la lancera

Partie jointe

OK

Rejoindre la partie

Un fois ceci fait et qu'on a cliqué sur "OK" dans la popup, la fenêtre ci-dessous apparaît. On doit attendre que l'administrateur lance la partie.



En attente du serveur...

On se connecte avec notre autre joueur, Pinsher, pour effectuer les mêmes étapes.



Utilisateur	Pinsher
Mot de passe	pass123

Se connecter

Une fois connecté on a accès à l'interface du joueur.



Bienvenue : Pinsher

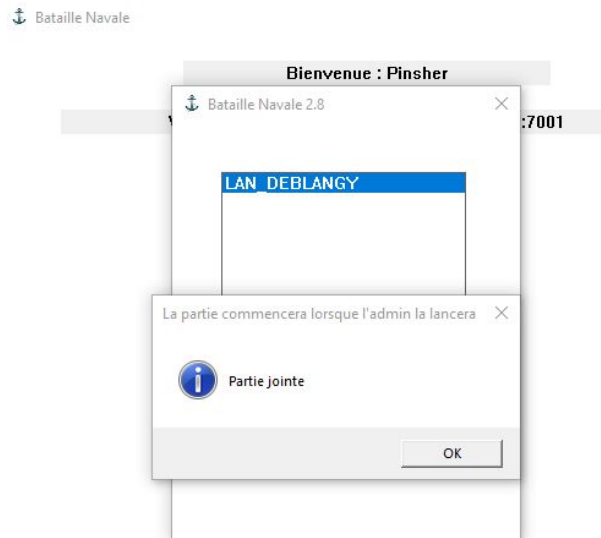
Vous êtes connecté au serveur 192.168.0.70:7001

Rejoindre une partie

Se déconnecter

On clique sur le bouton "Rejoindre une partie", et on obtient la liste des parties.

On rejoint la partie créée pour le test en la sélectionnant et en cliquant sur le bouton en bas de la fenêtre.



Un fois ceci fait et qu'on a cliqué sur "OK" dans la popup, la fenêtre ci-dessous apparaît. On doit attendre que l'administrateur lance la partie.



On retourne ensuite du côté de l'administrateur pour lancer la partie. Dans son interface d'administration, on clique sur le bouton "Liste des parties". On sélectionne dans la liste des parties celle liée à notre test : "LAN_DEBLANGY", et on clique sur le bouton permettant la gestion.



On obtient une petite fenêtre avec la liste des joueurs présents dans la partie, on peut voir que nos deux joueurs sont dedans.
 Pour les accepter, il suffit de lancer la partie. Si on ne veut pas d'un joueur on peut l'exclure avec le bouton prévu à cet effet.
 On clique donc sur le bouton "Lancer la partie".



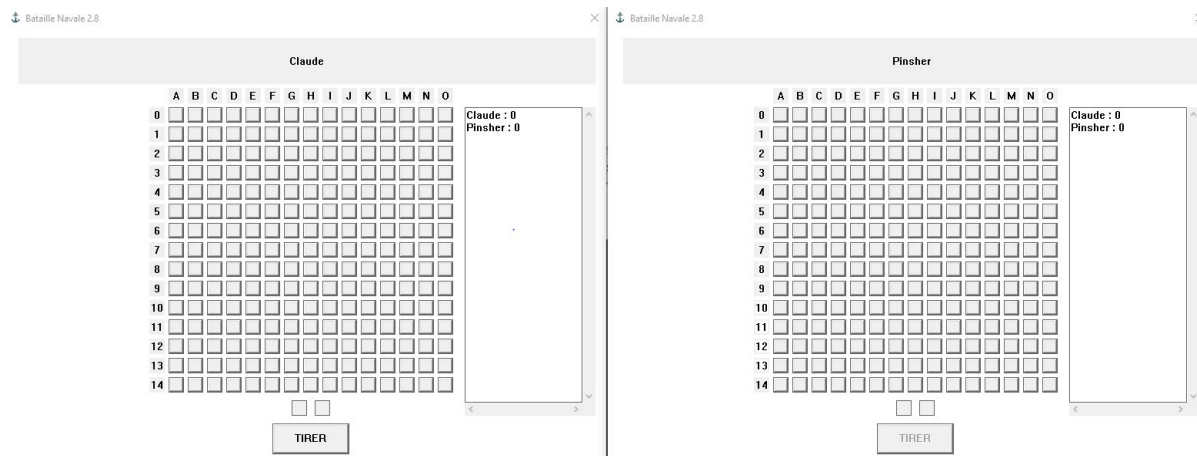
Du côté des joueurs, on peut voir que les fenêtres d'attente se sont transformées pour afficher la grille de jeu (et les scores).
 On peut donc commencer à jouer !

Tour 1 :

Dans ce premier tour on peut voir que c'est à Claude (écran de gauche) de commencer à jouer. Et on constate que Pinsher ne peut pas tirer pour l'instant.

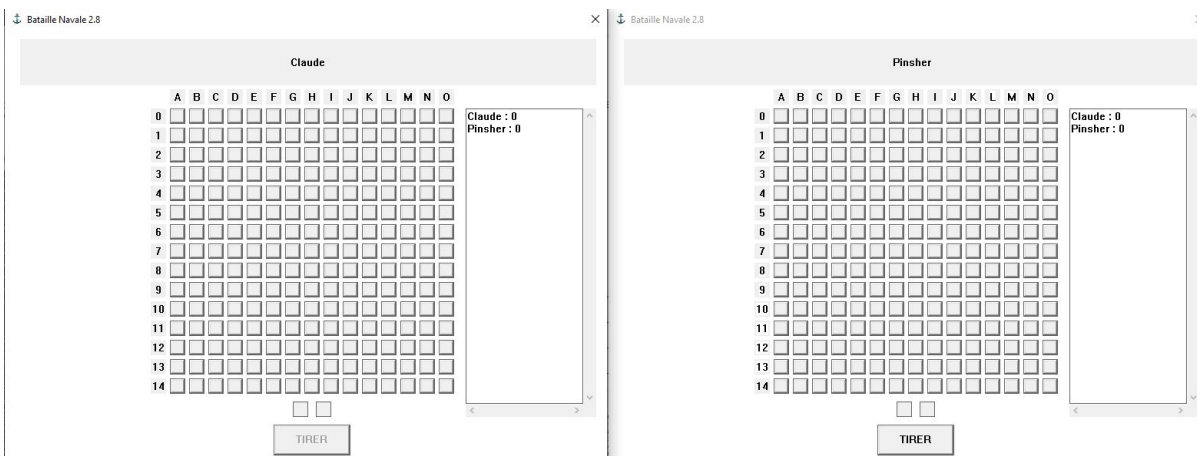
On fait tirer Claude dans l'eau en cliquant directement sur la cellule que l'on vise. Les coordonnées de celle-ci s'inscriront automatiquement dans les deux cases au-dessus du bouton "Tirer".

On clique ensuite sur le bouton "Tirer" pour envoyer les coordonnées au serveur.



Comme on peut le voir ci-dessous, Claude vient de tirer dans l'eau donc aucun des boutons ne s'est transformé en case touchée. De plus, Claude ne peut plus tirer contrairement à Pinsher qui ne connaît donc pas la cellule que Claude a visé.

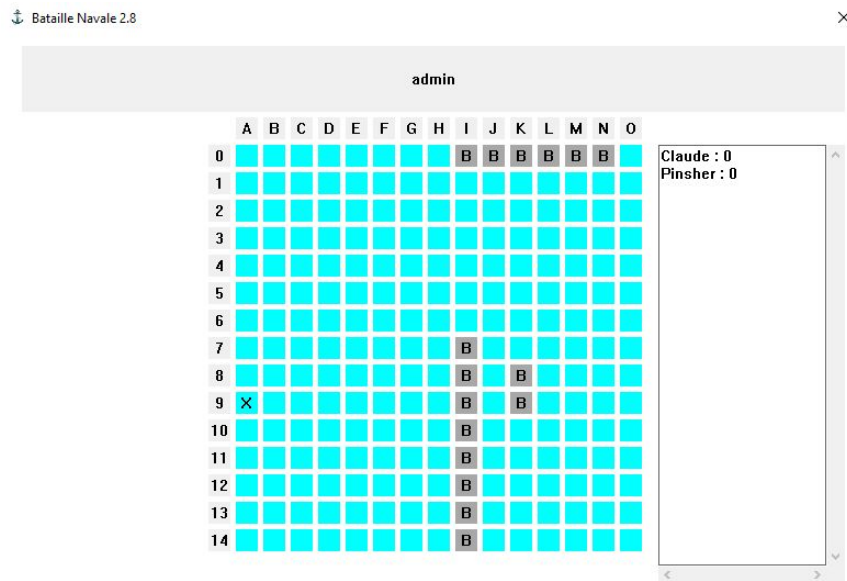
Pour ce tour, on fait aussi tirer Pinsher dans l'eau.



On peut d'ailleurs voir sur l'écran de l'administrateur, que nous avons placé en tant que spectateur en cliquant sur "Voir la partie" après l'avoir lancée, l'état du jeu.

On peut ainsi remarquer qu'aucun des bateaux n'a été touché, et on peut voir les coups dans l'eau marqués par un X.

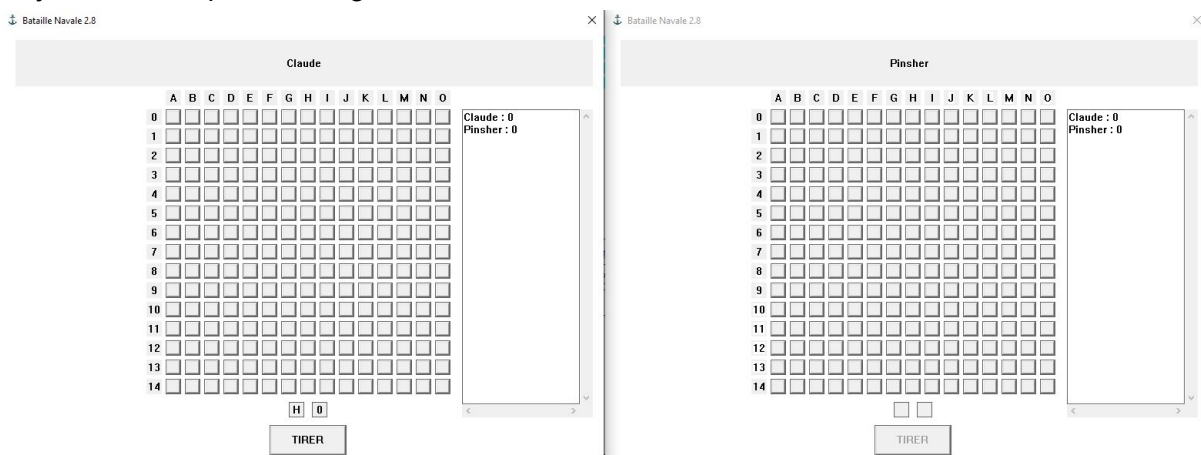
Ici il n'y a qu'un X car cette capture a été prise juste avant de faire jouer Pinsher.



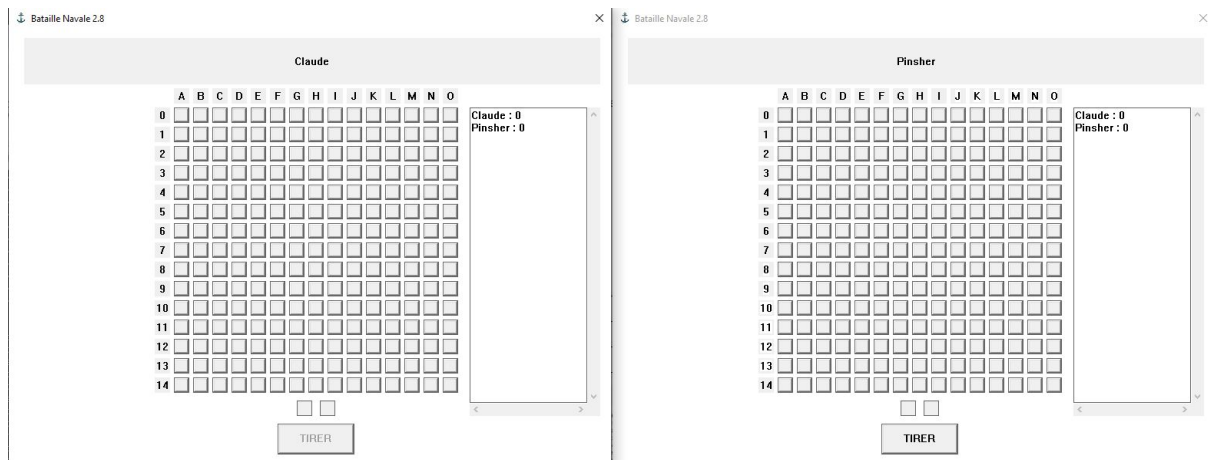
N.B. Si plusieurs coups (dans l'eau) sont tirés sur la même case, il y aura toujours le même X de marqué.

Tour 2 :

Au tour suivant nous avons fait tirer Pinsher dans l'eau, et on peut constater ici qu'il n'y a toujours rien de plus sur la grille.



C'est donc à Claude de jouer, il tire encore dans l'eau et rien ne se passe.

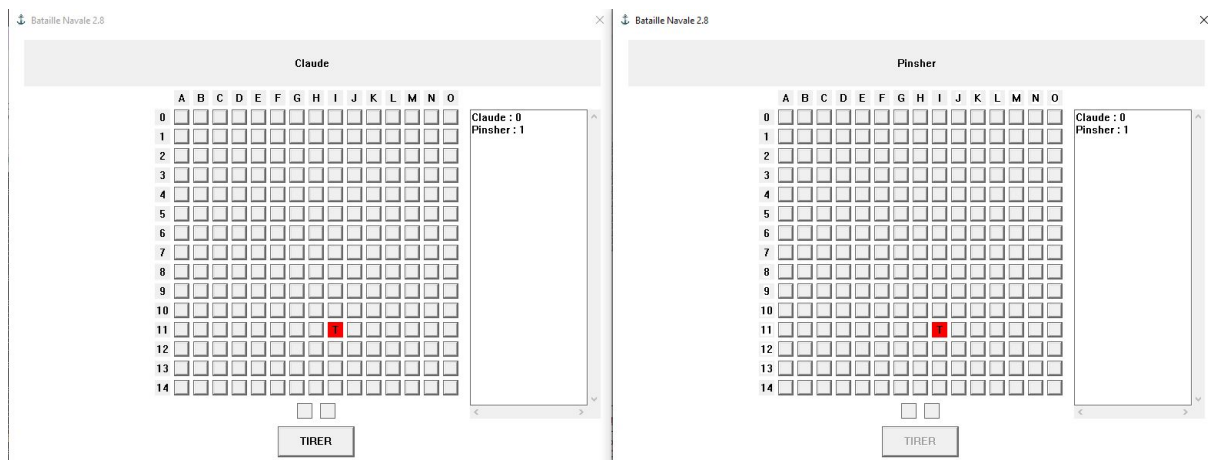


On va donc faire tirer Pinsher sur un bateau.

On peut ainsi voir apparaître un T sur fond rouge signifiant qu'un bateau a été touché.

La cellule n'est plus cliquable, ça n'aurait bien entendu aucun intérêt de retirer sur une case déjà touchée.

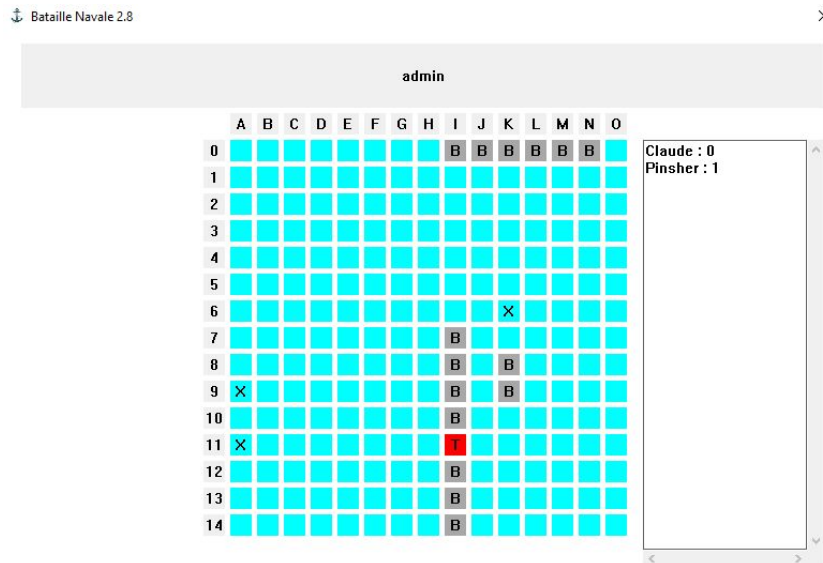
On peut aussi voir que la cellule a changé sur les grilles de tous les joueurs et que les scores se sont mis à jour.



On peut même visualiser l'état de la partie sur l'écran de l'administrateur.

On peut donc voir à quel endroit précisément le bateau a été touché.

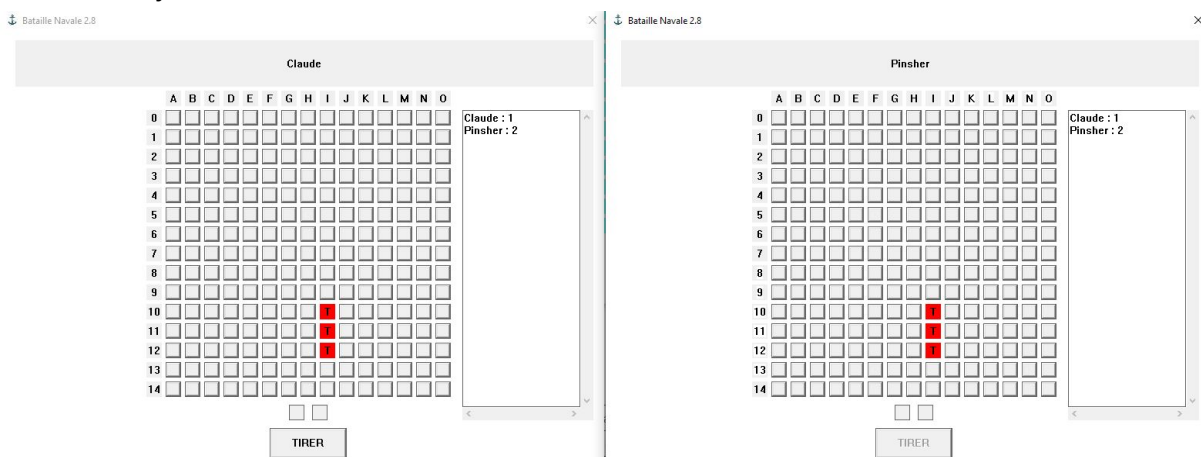
On peut aussi voir qu'un coup dans l'eau n'est pas passé très loin d'un bateau, et on remarquera que les scores sont bien à jour.



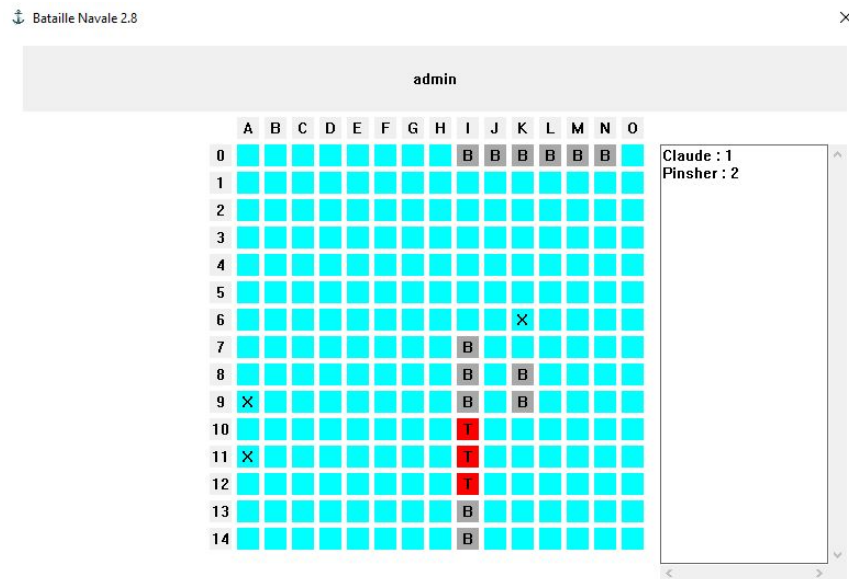
Tour 3

Comme on l'a vu précédemment, le joueur Pinsher a touché un bateau, Claude a donc un indice pour tirer, il vise et touche. Il a trouvé dans quel sens est aligné le bateau. Pinsher peut donc tirer en suivant cet alignement, elle vise et touche.

A la fin de ce troisième tour, on a donc les deux grilles actualisées pour nos joueurs ainsi que leur score à jour.



L'administrateur aussi a son interface de visualisation à jour.



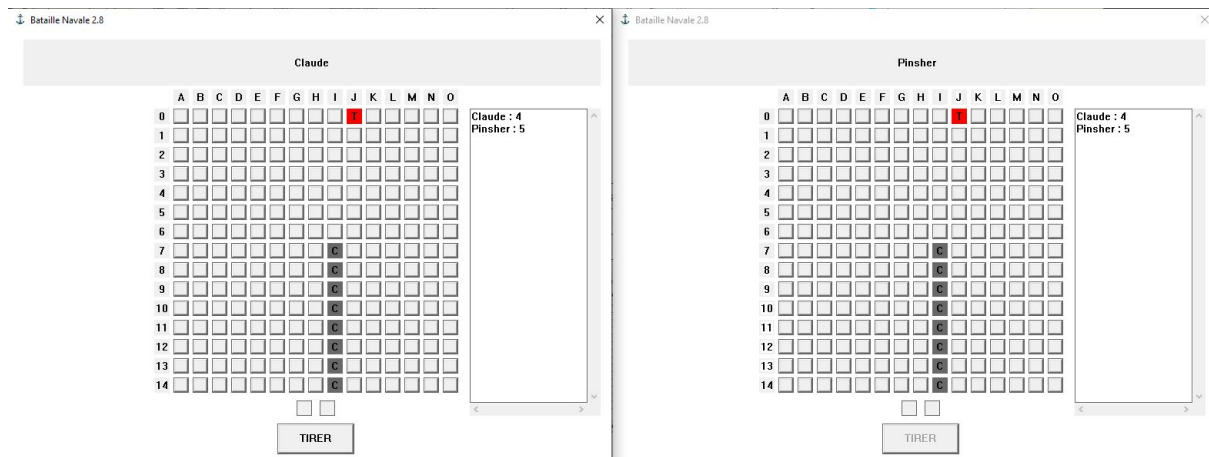
Le principe pour viser une case étant toujours le même (cliquer sur la case souhaitée) et les bateau étant assez grand, nous allons passer quelques tours.

Tour 6

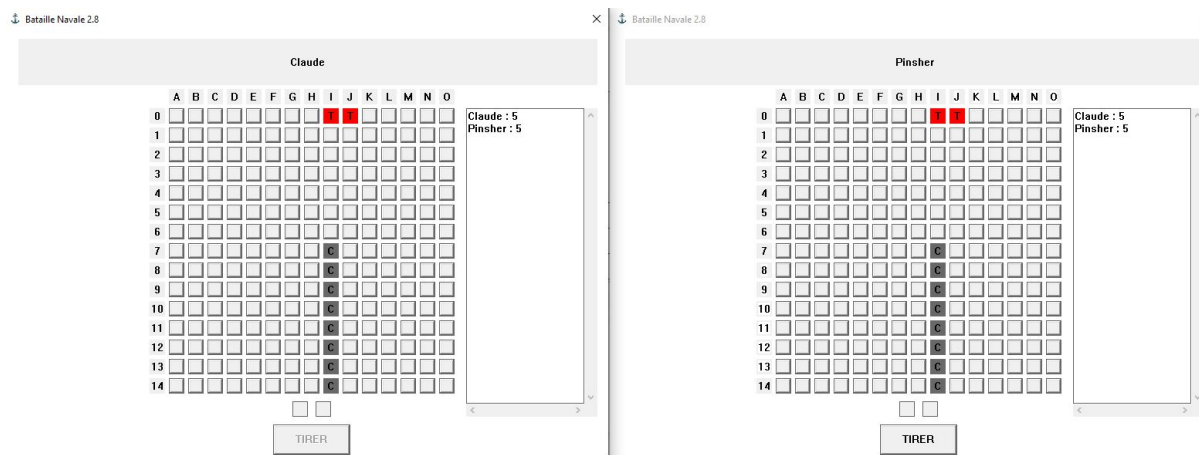
On vient d'avancer un peu dans la partie. Claude et Pinsher se sont disputés la destruction totale du bateau et c'est finalement Claude qui a réussi à le couler.

On peut voir que les cellules qui étaient un T sur fond rouge se sont transformées en un C sur fond gris. Claude a simplement pris 1 point car il a touché un bateau, il n'y pas de bonus attribué si quelqu'un coule un navire.

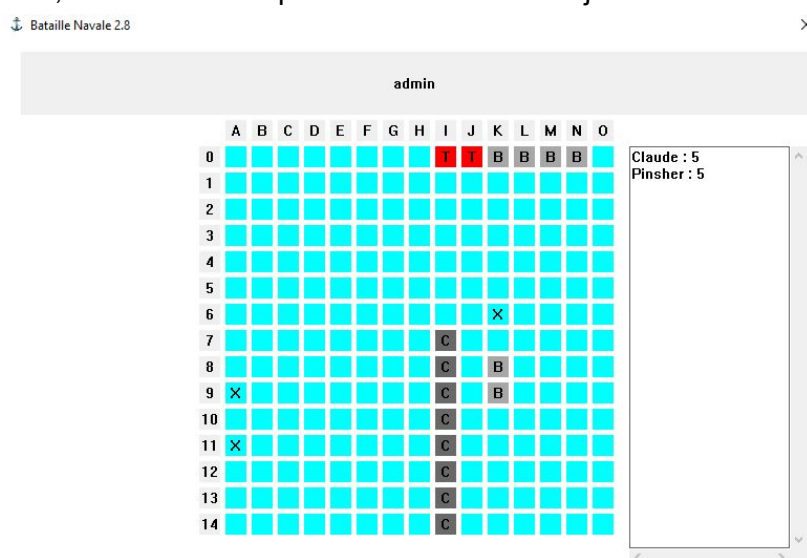
Pinsher a ensuite répondu en trouvant directement un bateau, et a donc aussi gagné 1 point. Ce joueur garde donc la tête dans les scores.



Claude tire donc sur la case juste à droite de celle touchée et comme on peut le voir ci-dessous il touche.



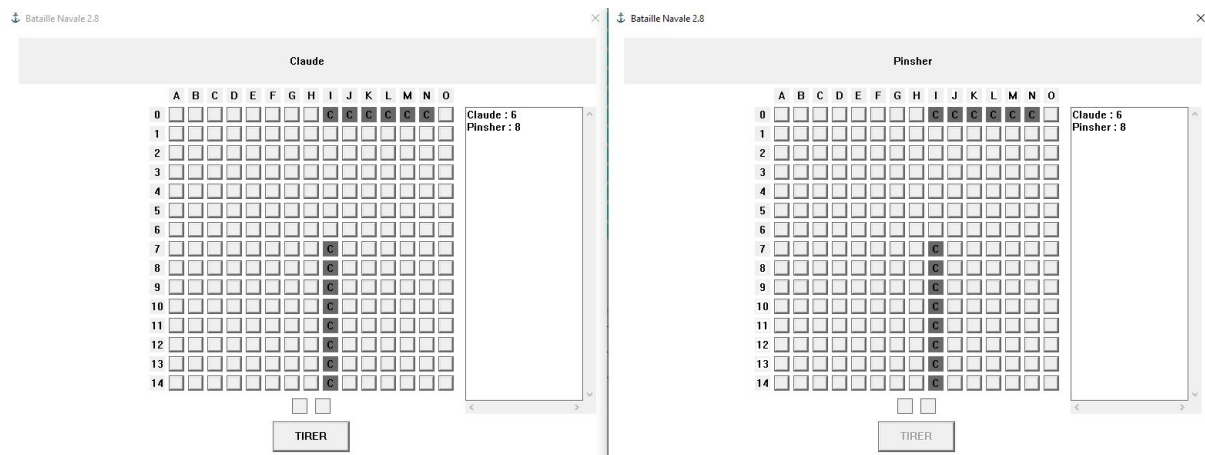
Comme d'habitude, l'administrateur peut visualiser l'état du jeu.



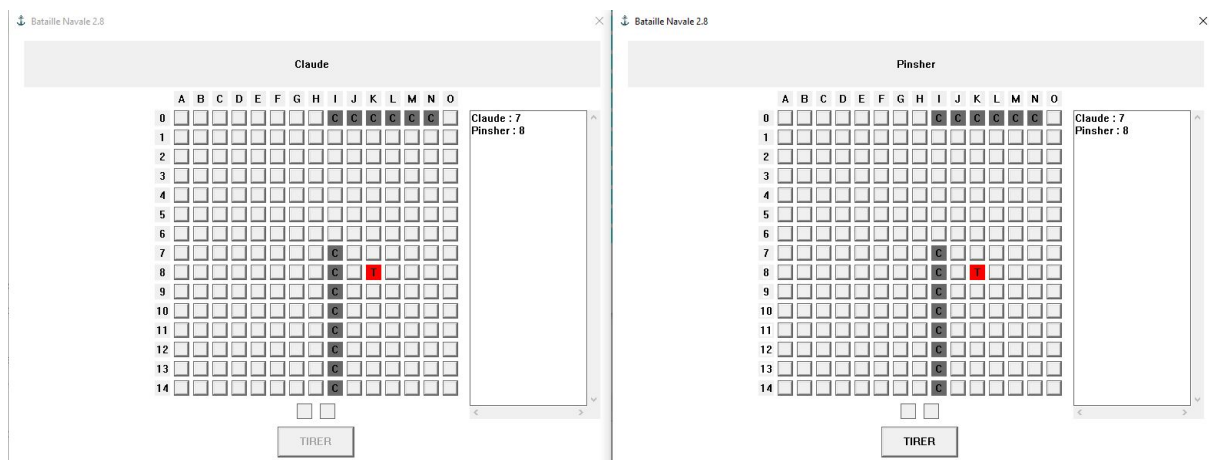
C'est maintenant au tour de Pinsher de jouer. Mais encore une fois le principe reste le même et, comme on peut le voir au-dessus, le bateau est un peu grand. On va donc avancer de quelques tours une fois de plus.

Tour 10

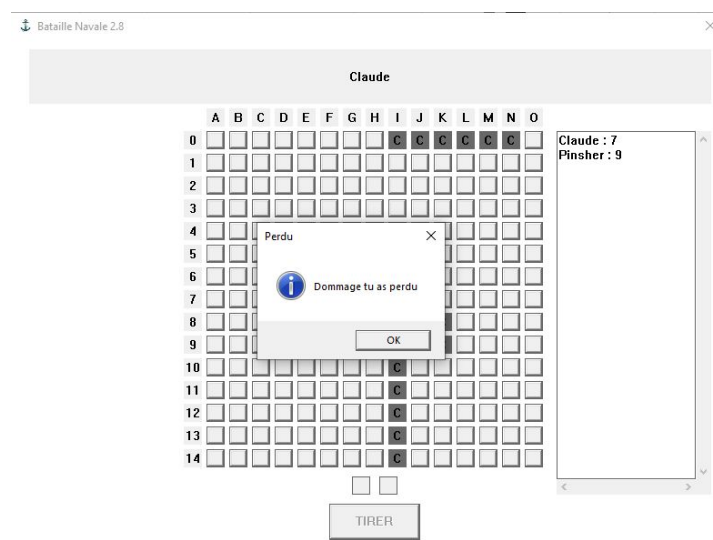
Nous avons bien avancé dans la partie, il ne reste qu'un bateau de taille 2 à couler. Entre temps, quelques coups dans l'eau ont été tirés à la recherche du dernier bateau. Voilà donc l'état du jeu à cet instant.

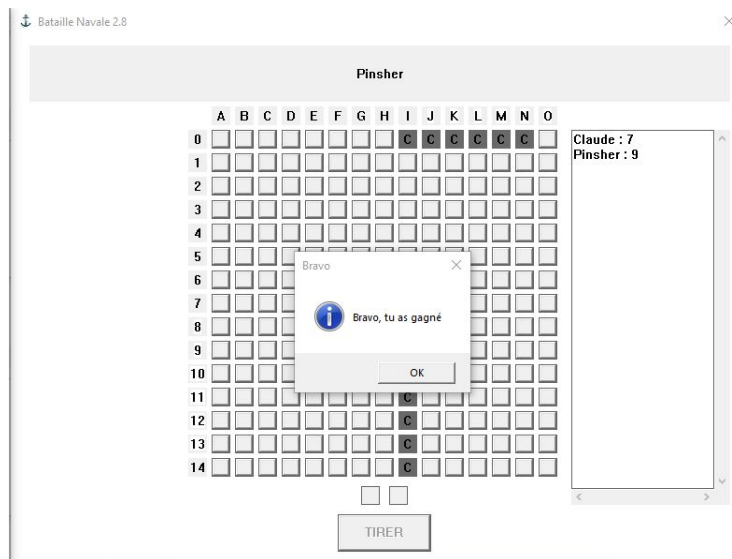


Comme on peut le constater ci-dessus, c'est donc au tour de Claude de jouer.
Il sélectionne une case, tire et touche le dernier bateau.



Au tour de Pinsher de tirer, le joueur vise une case annexe à celle touchée, tire et touche.
Le dernier bateau est donc coulé.
La partie est finie.



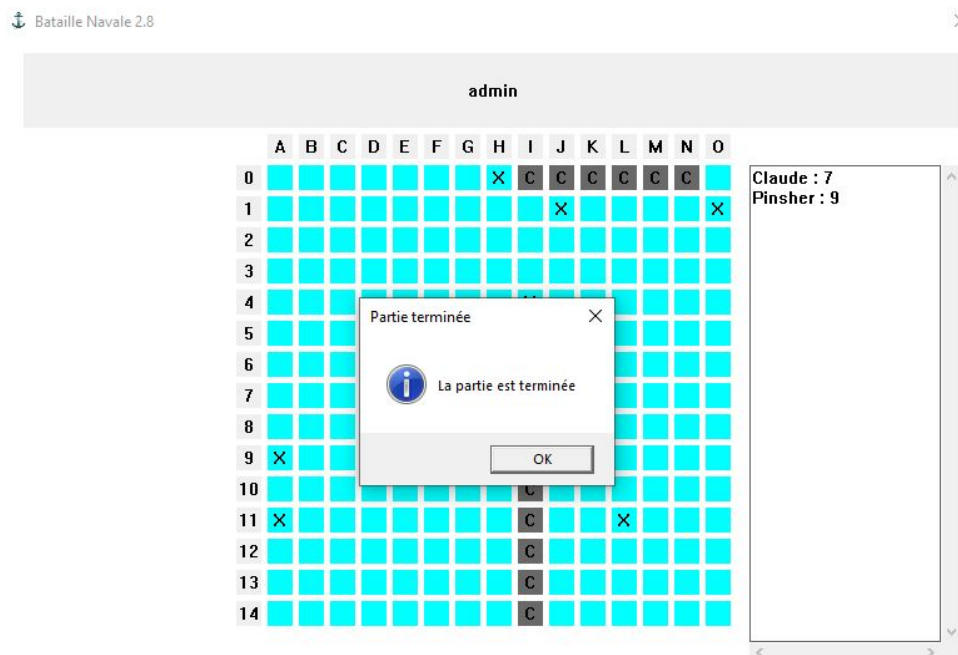


Les scores sont mis à jour une dernière fois, on remarque que Pinsher domine avec 9 points et que Claude est derrière avec 7 points.

Ils sont informés par une petite fenêtre popup que la partie est finie en leur indiquant s'ils ont gagné ou perdu.

L'admin, quant à lui, voit une dernière fois l'état du jeu s'actualiser et les scores se mettre à jour avant d'être informé que la partie est terminée.

On peut visualiser les scores finaux sur le bandeau de droite.



Après avoir cliquer sur le bouton “OK” de leur fenêtre popup, les joueurs comme l’administrateur se voient redirigés vers leur écran d’accueil respectif.

