ÉCOLE NATIONALE DES SCIENCES DE L'INFORMATIQUE

Rapport du projet de programmation « Bataille Navale »

Travail élaboré par :

- Ben Selem Oussema
- Chaieb Koussay
- Aoun Yassine

Sommaire :

I-Présentation générale du jeu « Bataille Navale »

1) Règles générales du jeu

Le jeu de la « bataille navale » est un jeu de société qui se joue à deux joueurs dans lequel les joueurs doivent placer leurs navires sur une grille « secrète » et doivent ensuite couler tous les navires adverses en touchant toutes les cases des navires à tour de rôle.

Le gagnant est celui qui parvient à couler tous les navires de l'adversaire avant que tous les siens ne le soient.

On dit qu'un navire est coulé si chacune de ses cases a été touchées par un coup de l'adversaire.

Chaque joueur possède une grille de 10 cases par 10 cases ou ces navires sont placés sans se superposer ou même se toucher.

Il en existe deux dispositions possibles :

Disposition standard :

- 1 Portes-avions (5 cases)
- 1 Croiseur (4 cases)
- 2 Contre-torpilleurs (3 cases)
- 1 Torpilleur (2 cases)

Disposition en Belgique :

- 1 Cuirassé (4 cases)
- 2 Croiseurs (3 cases)
- 3 Torpilleurs (2 cases)
- 4 Sous-marin (1 cases)

Pour la phase de tirs, chaque joueur doit l'un après l'autre choisir les coordonnées d'un tir constitué d'un indice de ligne et d'un indice de colonne.

Les effets du tir peuvent soit etre « à l'eau » si le tir n'a touché aucun navire. Soit touché si le tir a touché une case du navire.

Lorsque toutes les cases d'un navire sont touchées, le navire est « coulé ».

Version Console

2) Mécanisme du jeu :

Au lancement du jeu, le programme vous donnera une liste de choix :

```
1) 1 VS 1
2)Contre un Robot
Tapez 1 ou 2 :
```

Si vous voulez joueur avec un autre joueur humain, tapez 1. Si vous tapez 2 vous allez jouer contre l'ordinateur.

#Mécanisme de la partie 1 VS 1 :

Tout d'abord les noms des deux joueurs seront demandés.

```
1) 1 VS 1
2)Contre un Robot
Tapez 1 ou 2 :
1
donner le nom du premier joueur
Koussay
donner le nom du deuxieme joueur
Yassine_Oussema
```

Puis, le programme vous demandera quelle disposition vous voulez adopter pour cette partie :

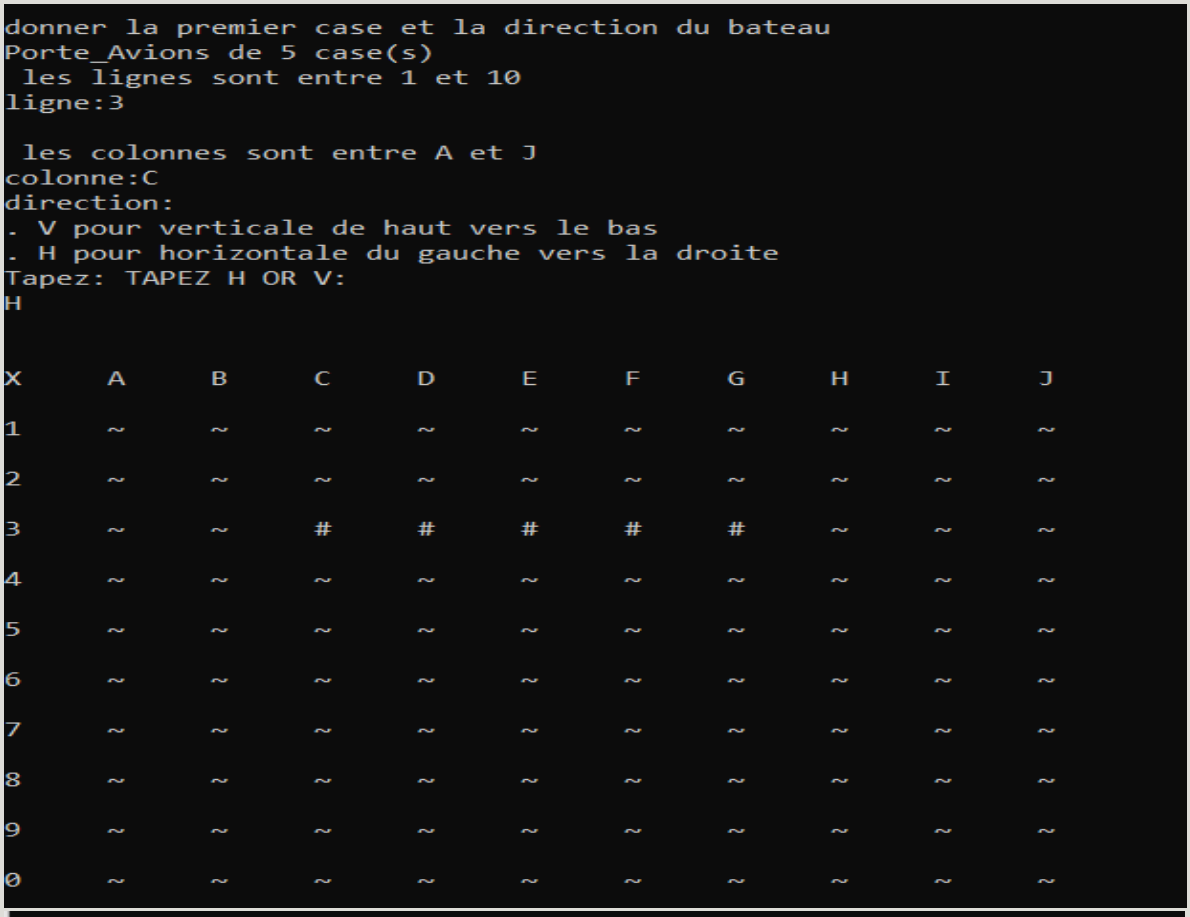
```
choisir la disposition:
1-premier disposition:
. 1 PORTE_AVIONS:5 cases
. 1 CROISEUR:4 cases
. 2 CONTRE_TORPILLEUR:3 cases
. 1 TORPILLEUR:2 cases
2-deuxieme disposition en belgique:
. 1 CUIRASSE:4 cases
. 2 CROISEUR:3 cases
. 3 TORPILLEUR:2 cases
. 4 SOUS_MARIN:1 case
 TAPEZ 1 OU 2
```

Pour la suite de cette explication, nous continuons notre discours avec la disposition 1.

Après avoir choisi la disposition, le jeu demandera au premier joueur de commencer à remplir le plateau.

A chaque fois, le jeu précisera quel bateau à placer et combien de cases ce navire occupera-t-il.

Le joueur est censé à préciser la première ligne de la case (entre 0 et 9) et la colonne de la case (lettre entre A et J) au placement et la direction du bateau : horizontale ou verticale.



Dans cet exemple, le navire se trouvera dans la ligne 3 commençant par la colonne C.

Le navire a des « # » comme figuration du plateau.

Ces actions seront répétées pour chaque bateau pour chacun des deux joueurs.

Les placements effectués, la phase de tir peut commencer.

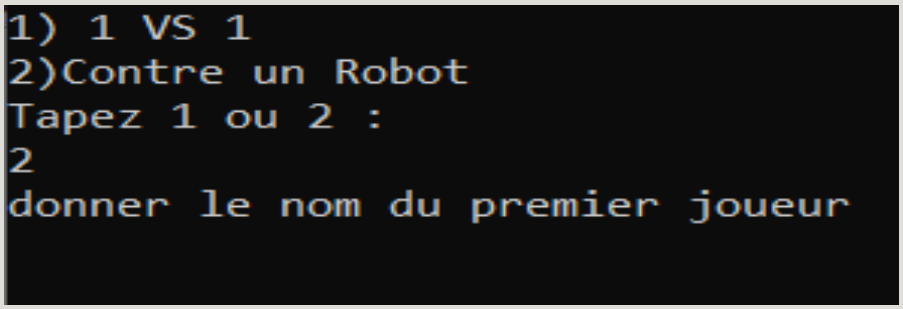
A chaque tour, chaque joueur devra entrer les coordonnées d'un tir.

Lorsque tous les navires d'un joueur seront coulés…

La fin du jeu est donc annoncée ainsi que le nom du gagnant.

#Mécanisme de la partie Robot :

Exécutons maintenant une partie contre un robot.



Le programme vous demandera le nom du joueur et quelle disposition vous voulez adopter pour cette partie :

```
donner le nom du premier joueur
Koussay
choisir la disposition:
1-premier disposition:
. 1 PORTE_AVIONS:5 cases
. 1 CROISEUR:4 cases
. 2 CONTRE_TORPILLEUR:3 cases
. 1 TORPILLEUR:2 cases
2-deuxieme disposition en belgique:
. 1 CUIRASSE:4 cases
. 2 CROISEUR:3 cases
. 3 TORPILLEUR:2 cases
. 4 SOUS_MARIN:1 case
 TAPEZ 1 OU 2
```

Pour cette exemple, nous allons essayer la 2ème disposition (Disposition en Belgique) :

```
 TAPEZ 1 OU 2
2
Koussay commencer a remplir  ton plateau


X    A    B    C    D    E    F    G    H    I    J

1    ~    ~    ~    ~    ~    ~    ~    ~    ~    ~

2    ~    ~    ~    ~    ~    ~    ~    ~    ~    ~

3    ~    ~    ~    ~    ~    ~    ~    ~    ~    ~

4    ~    ~    ~    ~    ~    ~    ~    ~    ~    ~

5    ~    ~    ~    ~    ~    ~    ~    ~    ~    ~

6    ~    ~    ~    ~    ~    ~    ~    ~    ~    ~

7    ~    ~    ~    ~    ~    ~    ~    ~    ~    ~

8    ~    ~    ~    ~    ~    ~    ~    ~    ~    ~

9    ~    ~    ~    ~    ~    ~    ~    ~    ~    ~

0    ~    ~    ~    ~    ~    ~    ~    ~    ~    ~
donner la premier case et la direction du bateau
CUIRASSEde4 case(s)
 les lignes sont entre 1 et 10
ligne:
```

Le robot remplie son plateau d'une manière automatique et aléatoire grâce à un script que nous avions implémentée.

Puis la phase attaque est la même comme on a expliqué précédemment pour le joueur. Mais la fonction attaque est une version intelligente. Le robot attaquera aléatoirement mais lorsque son attaque croise une case d'un navire adversaire il tentera de le détruire en attaquant les cases proches de celui attaqué.

II- Notre démarche :

Nous avons importé les bibliothèques suivantes :

#include <iostream>

#include<cstring>

#include<cstdlib>

#include<ctime>

Les classes utilisées :

-classe Bateau :

-Attributs :

Attributs	Rôles
Int nb	Les cases occupés par le bateau
String nom	Nom du bateau

-méthodes :

Méthodes	Rôles
Bateau()	Initialisation
Bateau (int, string)	Constructeur
Int getnb()	Getter de la classe bateau pour avoir accès à l'attribut nb.
String getnom()	Getter de la classe bateau pour avoir accès à l'attribut nom.

-Classe Joueur :

-Attributs :

Attributs	Rôles
Int nbc_rest	Cases restantes à détruire
Char plateau_bataille[11][11]	Plateau utilisée pour placer les navires à la phase «remplissage».
Char plateau_info[11][11]	Plateau affiché après l'attaque lors de la phase « attaque».
String nick_name	Un pseudo pour identifier le joueur.

-Méthodes :

Méthodes	Rôles
Joueur(String)	Constructeur de la classe « Joueur ». Initialiser les deux plateaux : -Plateau Bataille -Plateau info
Int get_nbc_rest()	Donner le nombre de cases restantes après attaque
String get_nick_name()	Getter de la classe joueur pour avoir accès à l'attribut « nick_name »
Void affiche1()	Affichage du « plateau bataille »

Void affiche2()	Affichage du « plateau info »
Void remplir(l1[5], bateau l2[10], int k)	Remplissage du plateau bataille
Void attack (joueur&)	Action d'attaque du joueur
Void robot_remplir(bateau l1[5], bateau l2[10], int k)	Remplissage du « plateau bataille » du robot
Void robot_attack(joueur& , int& , int&, int &, int&, int& , int&, int &, int&, int&)	Action d'attaque du robot

Autres fonctions utilisées :

Fonctions	Rôles
Vérifier(int, char, char, char l[11][11], bateau)	Vérification des placements des bateaux du joueur.
Verifier1(int, int, char l[11][11], bateau)	Vérification des placement des bateaux du robot.

III- Difficultés rencontrés et solutions adoptées :

-Méthode remplir : pour la partie test, il y'avait un mal fonctionnement au début manque d'ordonnance donc on a eu recours à la fonction vérifier. La méthodologie adaptée est de subdivisée cette fonction en plusieurs sous-fonctions pour faciliter le code et le rendre plus lisible.

-Méthode attaquer : nous avions le problème de l'affichage sur le terminal pour le cas ou deux joueurs jouent. En effet, un joueur pouvait voir ce qu'il y avait affiché pendant le tour de l'autre joueur, et donc de voir, en l'occurrence, sa grille personnelle.

Pour résoudre ce problème, on a eu recours à la commande system ('cls').

Cette commande permet de nettoyer le terminal, et permet donc, à chaque changement de tour d'empêcher le joueur de voir les données de son adversaire.

Comme Notre objectif est de donner une bonne expérience utilisateur ce qui a causé plus de défis.

-Robot Remplir : cette fonction avait quelques lignes de commande que la fonction remplir lors d'un joueur humain. Mais elle a exigé un autre raisonnement. Ce qui a engendré l'utilisation d'une autre fonction remplir personnelle au robot.

-Robot attaque : c'est le plus grand défi qu'on a rencontré. On a dû implémenter un script intelligence artificielle à ce robot. On a dû répéter ce script, ce raisonnement à plusieurs tentatives pour arriver au résultat souhaité.

Version interface graphique :

Outils utilisés :

-Langage : C++

-Environnement : Visual Studio

-Bibliothèque : wxWidgets

wxWidgets (anciennement wxWindows) est une boîte à outils de widgets et une bibliothèque d'outils pour la création d'interfaces utilisateur graphiques (GUI) pour des applications multiplateformes. wxWidgets permet au code GUI d'un programme de se compiler et de s'exécuter sur plusieurs plates-formes informatiques avec un minimum ou aucune modification du code.

Les classes utilisées
wxBackgroundBitmap
Cfill
Cmain
CMenu
CApp

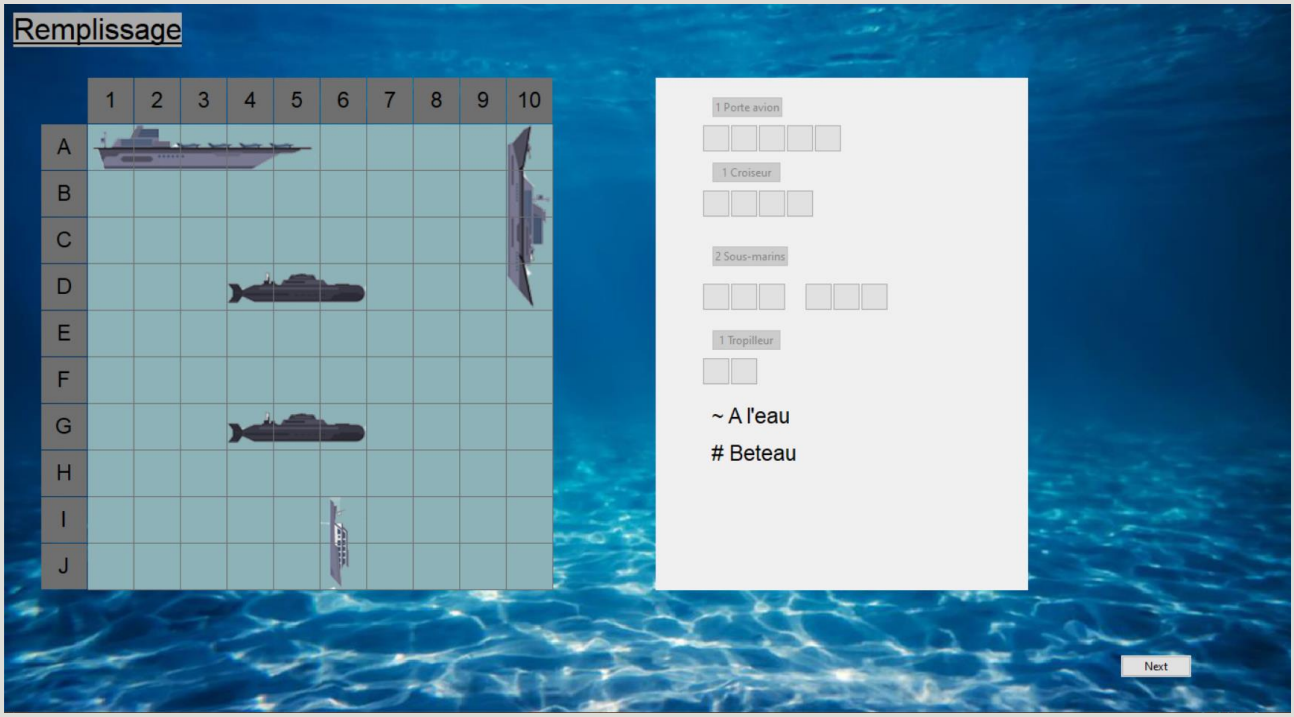
Autres bibliothèques utilisées :

Wx/bitmap.h : insérer une image.

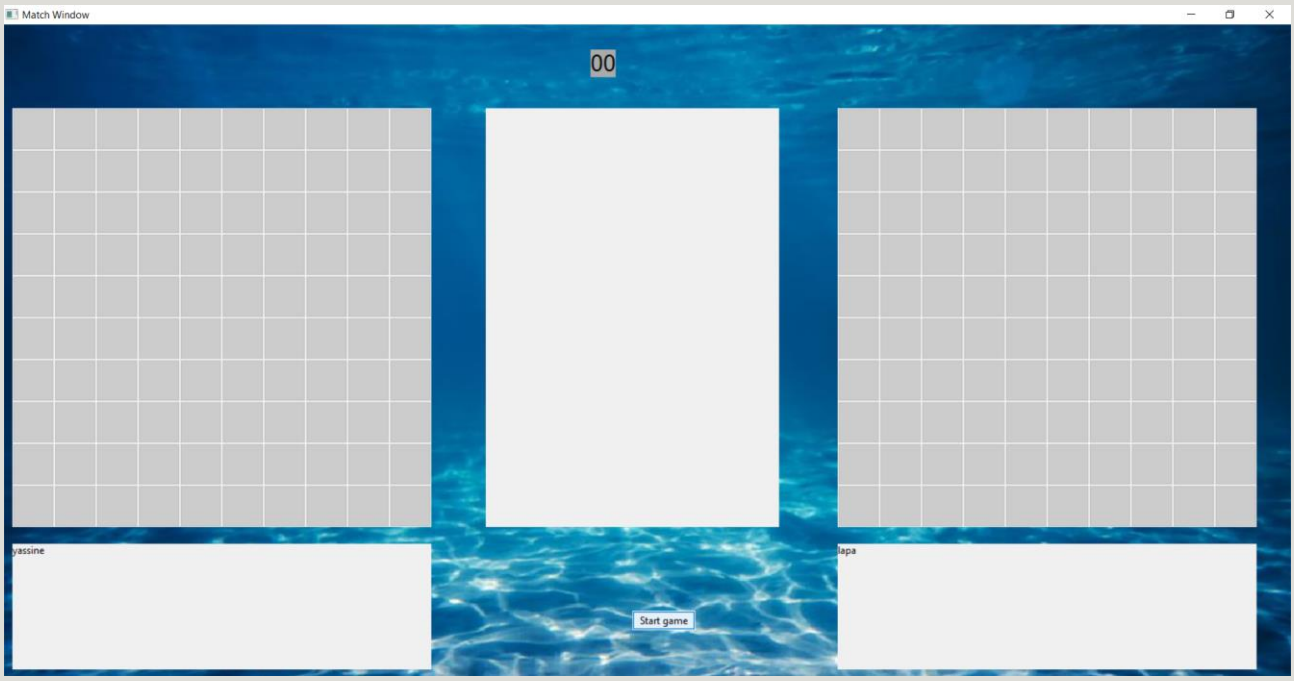
Wx/event.h : déclencher des actions. Par exemple, pour passer d’une fenêtre à un autre.

Ecran d’accueil :

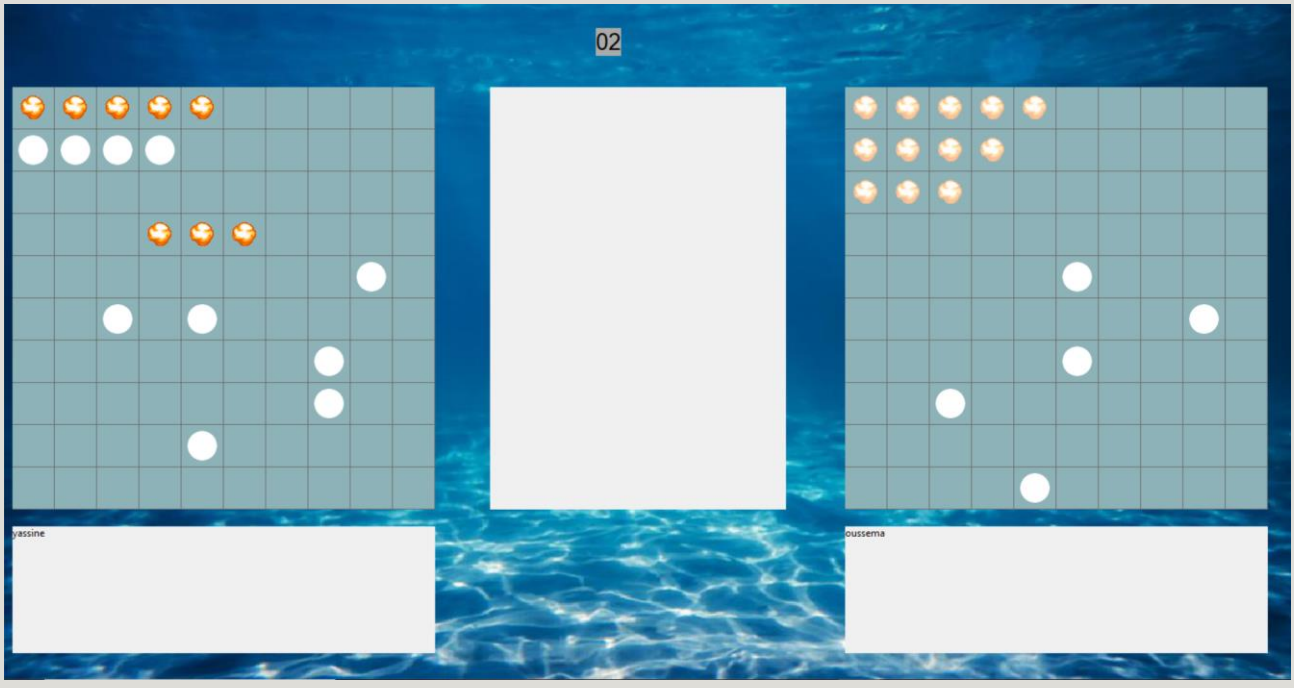
Ecran remplissage :



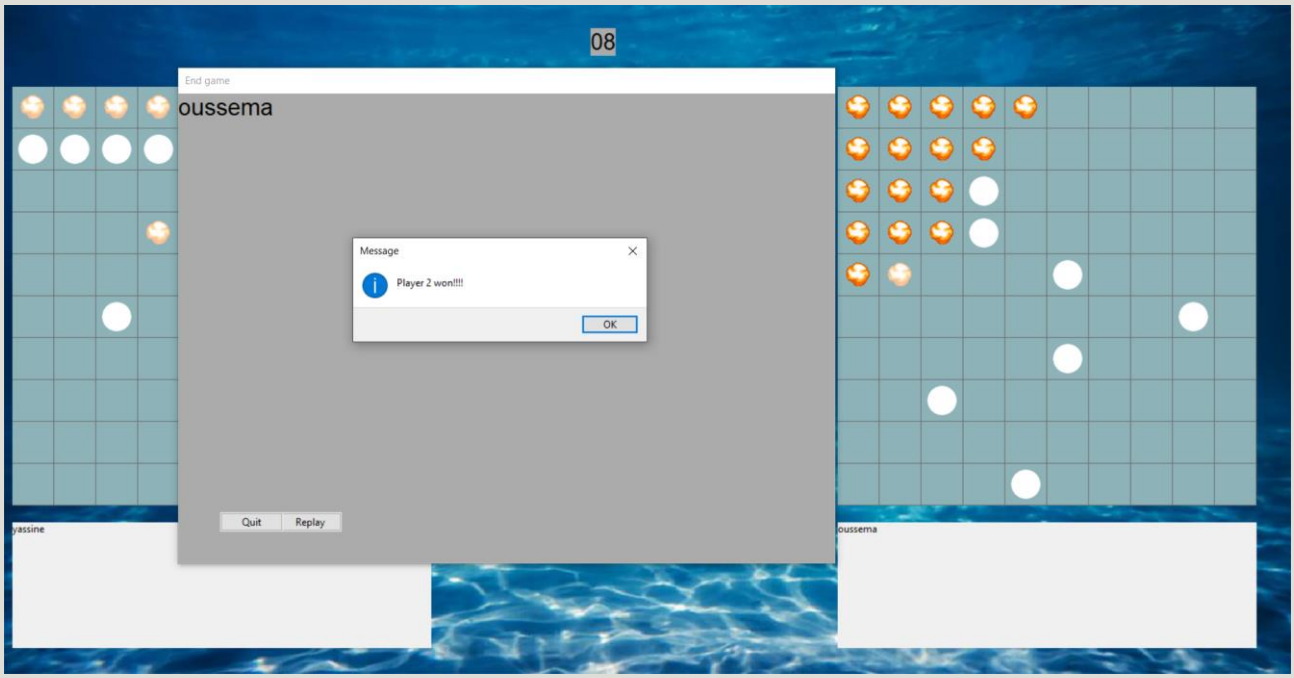
Ecran bataille avant attaque :



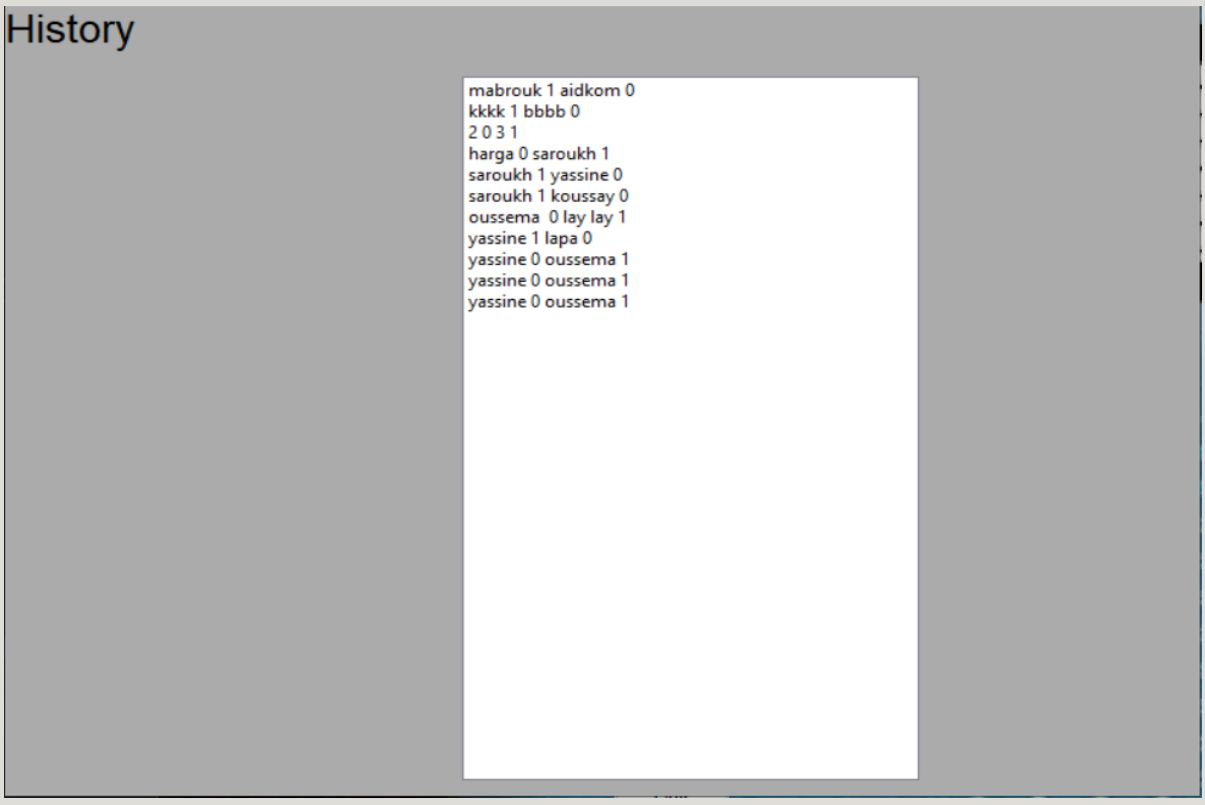
Ecran bataille après attaque :



Affichage du Vainqueur :



Affichage de l'Historique :



Problèmes rencontrés :

-L’installation et la configuration de la bibliothèque wxWidgets.

-Manque de documentation.

-Le choix de la bibliothèque à utiliser. Nous étions hésités entre la bibliothèque QT, wxWidgets, SFML et SDL.