



## HLIN302 – Projet noté « Bataille Navale »

**Programmation impérative avancée**  
**Alban MANCHERON et Pascal GIORGI**

---

L'objectif de ce projet est la réalisation du célèbre jeu « Bataille Navale » en C++, permettant à un utilisateur (humain) d'afficher et de jouer à la bataille navale contre un ordinateur dans un terminal Unix.



Le principe du jeu est le suivant, chaque joueur (l'humain et l'ordinateur) dispose d'une flotte de bateaux – les flottes sont identiques pour les deux joueurs. Une fois que chacun a placé sa flotte sur son territoire maritime, le jeu commence et les joueurs envoient à tour de rôle des missiles pour essayer de détruire les bâtiments adverses (à chaque tour un joueur n'envoie qu'un seul missile). Chacun des deux territoires maritimes est représenté par une grille de taille  $10 \times 10$ . Chaque joueur dispose au départ des mêmes bâtiments, chaque bâtiment occupant un nombre de cases spécifique et ne pouvant être placé sur la grille (sans dépassement de la grille, ni chevauchement d'un autre bâtiment) qu'horizontalement ou verticalement. Pour détruire une embarcation adverse, il faut toucher chacune des cases que celle-ci occupe.

Les bâtiments de chaque joueur sont :

- 1 porte-avions (5 cases) ;
- 1 croiseur (4 cases) ;
- 1 contre-torpilleur (3 cases) ;
- 1 sous-marin (3 cases) ;
- 1 torpilleur (2 cases).



Avant de définir les attentes pédagogiques, il convient donc de présenter le...

## 1 Cahier des charges

Afin de pouvoir jouer, il faut donc visualiser les deux territoires maritimes (celui du joueur humain et celui du joueur virtuel) ainsi qu'une zone de dialogue permettant d'afficher des messages et instructions à l'attention du joueur humain.

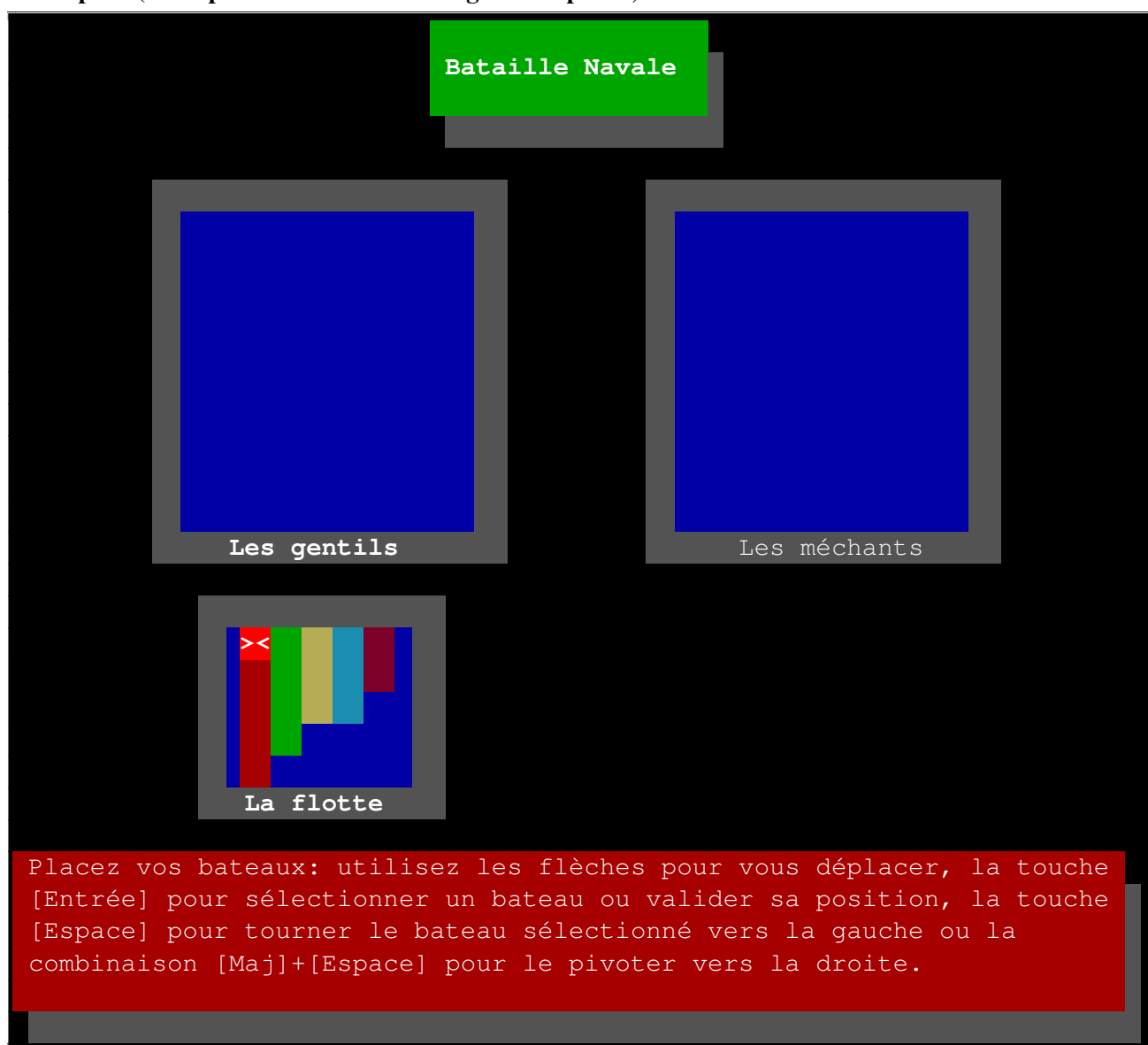
Cela implique donc dans un premier temps, de créer les grilles de chaque joueur, puis de positionner les bâtiments. Concernant le joueur virtuel (ordinateur), le placement se fera au hasard tout en s'assurant que les bâtiments ne se chevauchent pas et sont bien tous inclus dans la grille. Pour le joueur humain, les bâtiments seront tous positionnés au départ les uns à côté des autres (de la gauche vers la droite), dans le sens vertical sur un espace dédié. L'exemple 1 vous donne une idée de ce à quoi peut ressembler votre interface.

L'utilisateur doit pouvoir :

- se déplacer sur la grille ou dans sa flotte en utilisant les flèches de son clavier : , ,  et .

- sélectionner un navire de sa flotte en se plaçant dessus et en appuyant sur la touche **[Entrée]**. Lorsqu'un navire est sélectionné, il est déplacé automatiquement de la flotte vers la grille. Un code couleur – *a minima* – doit permettre de le distinguer des autres navires et la case qui a permis de le sélectionner doit également être mise en emphase ; Le déplacement du navire s'effectuera en déplaçant le point de sélection du navire à l'aide des flèches du clavier (**[↑]**, **[↓]**, **[←]**, **[→]**).
- faire pivoter un navire de 90° vers la droite en appuyant sur la barre d'espace et de 90° vers la gauche en appuyant simultanément sur la touche **[↑]** et la barre d'espace. La rotation doit s'effectuer à partir du point de sélection du navire et ne doit pas entraîner de sortie de la grille.
- Valider le positionnement du navire en appuyant sur la touche **[Entrée]**. Attention, si le positionnement entraîne le chevauchement avec un autre bâtiment la position ne peut être validée et un message d'erreur doit être émis au joueur. Après validation, l'utilisateur recommence le placement des autres navires restant dans sa flotte. La partie commence dès lors que tous les navires ont été placés correctement dans la grille.

### Exemple 1 (Exemple d'écran de démarrage d'une partie)



Une fois le placement réalisé et validé, la partie *stricto sensu* peut débuter. Il s'agit alors de désigner le joueur qui commence (un tirage aléatoire est tout indiqué).

Lorsque c'est à l'humain de jouer, il choisit une case sur le territoire ennemi (toujours avec les flèches du clavier pour se déplacer et la touche **[Entrée]** pour valider). S'il n'y a pas de bateau sur la case visée, alors la case

correspondante doit être marquée par un code visuel et il ne doit plus être possible de la sélectionner pour les tours suivants (typiquement, la case sélectionnée doit « sauter » par dessus, si c'est possible). Si *a contrario* un bateau est présent sur la case visée, l'information doit apparaître visuellement et si le bateau est coulé (*i.e.*, il a été touché sur toutes les cases qu'il occupe), alors cette information doit également apparaître dans le terminal.

Lorsque c'est au joueur virtuel de jouer, alors une case valide (*i.e.*, qui n'a pas déjà fait l'objet d'un tir) est choisie et comme précédemment, l'affichage est mis à jour de sorte de faire figurer l'information.

La partie s'arrête lorsque tous les bâtiments de l'un des joueurs ont été coulés. Le joueur possédant encore des bateaux est déclaré vainqueur.

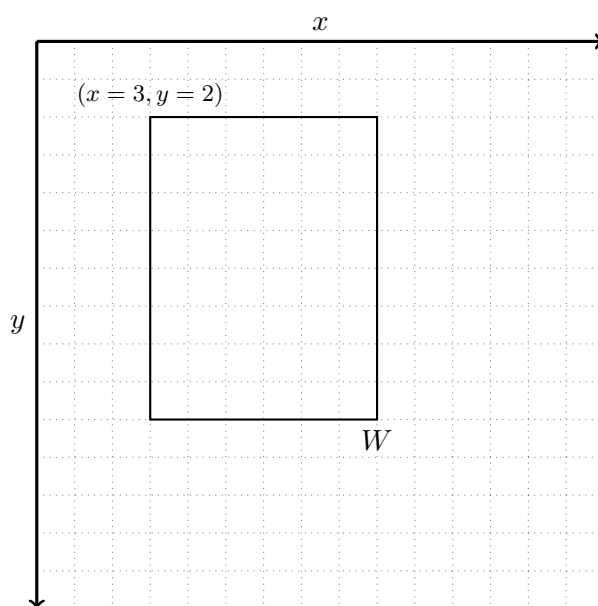
Afin de proposer une option de score permettant d'établir un classement des joueurs, il suffira de compter le nombre de missiles utilisés par le gagnant.

## 2 Mise en œuvre

Afin de réaliser l'affichage graphique ainsi que l'interaction homme-machine, vous devrez vous appuyer sur une bibliothèque permettant la manipulation de votre terminal. Vous devrez également modéliser les différents composants du jeu (grille, navires, joueurs, ...)

### Gestion de l'interface

La bibliothèque permettant de manipuler votre terminal et les interactions avec le clavier se nomme `ncurses`<sup>1</sup> et elle est fournie librement dans toutes les distributions Unix. Pour utiliser cette bibliothèque, il vous suffit d'inclure le fichier `ncurses.h` et compiler votre programme en ajoutant l'option `-lncurses`. Comme la prise en main de cette bibliothèque n'est pas évidente de prime abord, nous vous fournissons une classe C++ permettant une gestion simplifiée. La classe `Window` définit des objet permettant de gérer des fenêtres dans votre terminal. Par exemple, si votre terminal est de dimension  $15 \times 15$ , la déclaration `Window W(3, 2, 6, 8)` permet de définir une fenêtre de taille  $6 \times 8$  dont le coin supérieur gauche est positionné à la coordonnée  $(x = 3, y = 2)$ . Attention, la coordonnée du coin supérieur gauche de votre terminal est à la position  $(x = 0, y = 0)$  et l'axe des  $x$  est orienté vers la droite alors que l'axe des  $y$  est orienté vers le bas. La figure ci-dessous illustre la construction de notre exemple.



À partir d'un objet `Window` il est possible d'écrire du texte ou un caractère à une position donnée dans la fenêtre. Pour cela, il faut utiliser la méthode `print(x, y, s)` où  $(x, y)$  représente la position où l'on souhaite écrire dans la fenêtre (la position  $(0, 0)$  correspond au coin supérieur gauche de la fenêtre) et  $s$  représente le caractère ou la chaîne de caractère que l'on veut afficher.

1. <https://fr.wikipedia.org/wiki/Ncurses>

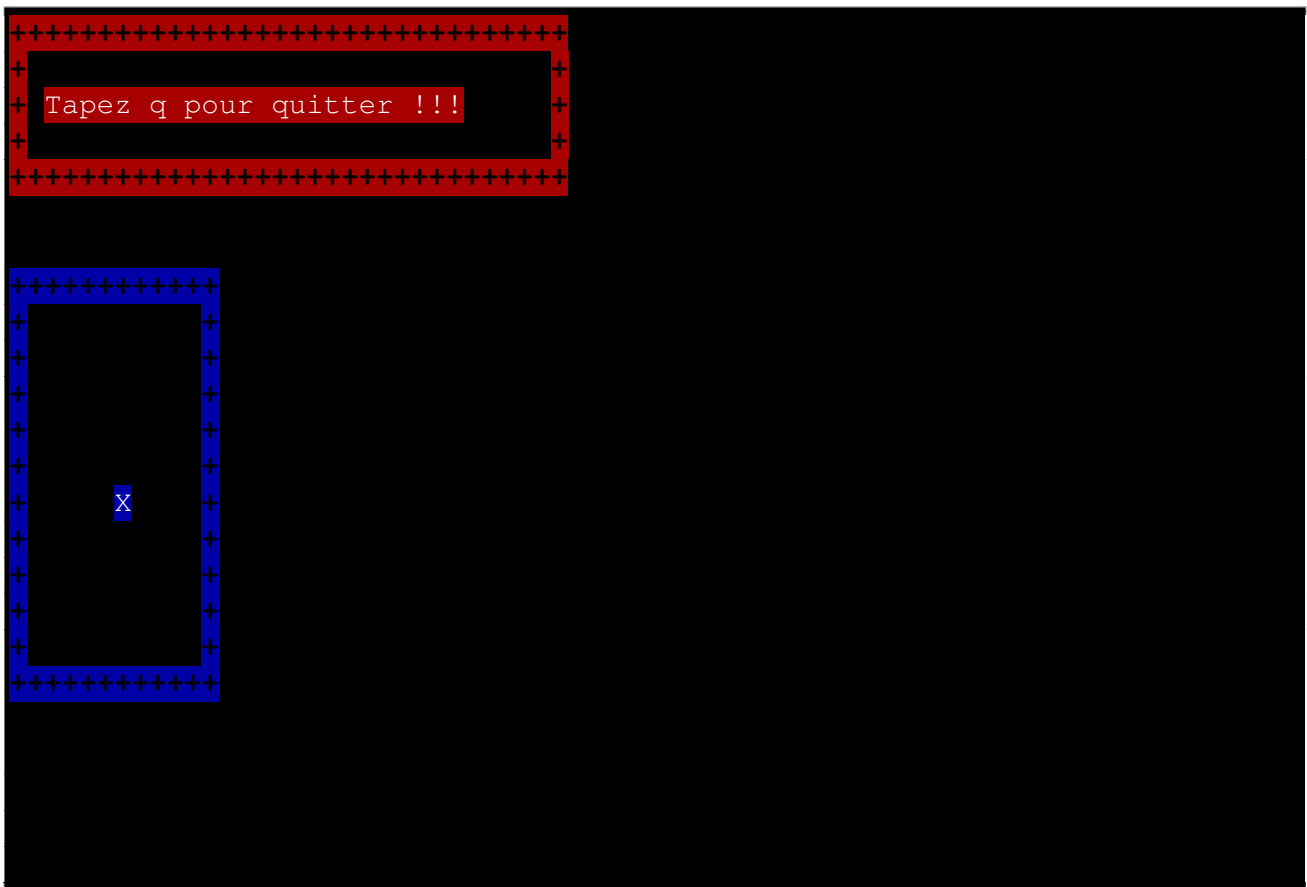
La fenêtre utilise un système d’affichage de couleur : une couleur de texte et une couleur de fond. Il est possible de changer cette couleur via la méthode `setCouleurFenetre`. De même, on peut changer la couleur de la bordure de la fenêtre via la méthode `setCouleurBordure`. On peut également spécifier une couleur `c` à la méthode `print(x, y, s, c)` pour écrire dans une couleur choisie. Enfin, on peut supprimer l’ensemble des textes écrits dans une fenêtre en utilisant la méthode `clear`. Le code de la classe `Window` est disponible dans l’espace projet sur l’ENT/Moodle du cours : fichiers `window.h` et `window.cpp`. Le fichier *header* comporte des commentaires vous permettant d’utiliser correctement cette classe et les couleurs d’affichage. Si vous le souhaitez, vous pouvez améliorer cette classe pour qu’elle gère plus d’options d’affichage de la bibliothèque `ncurses`.



Pour que votre programme fonctionne correctement avec la classe `Window` il faut que votre programme ressemble à celui-ci :

```
1 #include "window.h"
2
3 int main(int argc, char** argv) {
4     startProgramX();
5     myprogram();
6     stopProgramX();
7     return 0;
8 }
```

Vous devrez remplacer l’appel à la fonction `myprogram()` par la fonction principale de votre jeu. Vous trouverez également un exemple de programme utilisant la classe `Window` dans l’espace projet sur l’ENT/Moodle du cours. Rappel, pour compiler il vous faudra ajouter l’option `-lncurses`. Ce programme vous montrera également comment vous pouvez interagir avec les actions au clavier. Voici un rendu graphique du programme :



Pour la réalisation de votre programme, vous devez impérativement utiliser la classe `Window`. Si vous souhaitez utiliser d’autres possibilités de la bibliothèque `ncurses`, il vous faudra les intégrer dans la classe `Window`.

## Modélisation du programme

Vous devez vous appuyer sur les concepts vus en cours, TD et TP. Le langage étant imposé (C++ pour ceux qui auraient oublié), il semble évident qu'il faut mettre en œuvre les paradigmes du langage (modélisation objet, polymorphisme [dont la surcharge d'opérateurs], allocation dynamique, ...).

Il vous faudra donc fournir plusieurs classes permettant de modéliser le jeu : bâtiments, territoires maritimes, menu, ...

## 3 Travail à réaliser

Le projet consiste donc à implémenter un programme complet permettant de jouer à la bataille navale contre l'ordinateur en C++ en utilisant les concepts abordés pendant l'ensemble des cours, TD, TP.

Pour mener ce projet à bien, il vous faudra bien évidemment définir les étapes clés du projet. Mais également appréhender des concepts qui ne relèvent pas directement de cette UE, comme l'utilisation d'une librairie tiers, la gestion des codes d'échappement permettant de modifier les couleurs d'avant-plan et d'arrière-plan d'un terminal<sup>2</sup>.

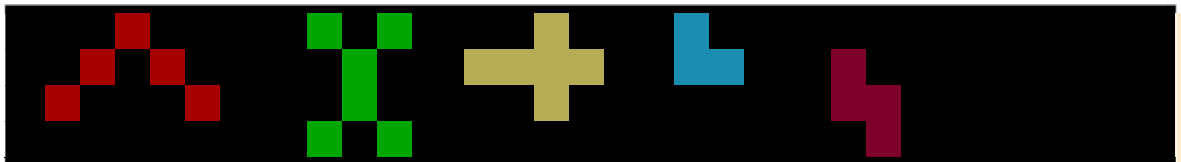
Le travail à réaliser devra correspondre à plusieurs étapes. Chaque étape correspond à une logique de conception, avec des objectifs spécifiques et des niveaux de difficulté variés. Plus les étapes seront réalisées correctement, meilleure sera la note finale du projet.

**(Étape 1)** Le jeu doit être jouable avec les caractéristiques suivantes :

- chaque composant du jeu doit être modélisé et intégré dans une classe : navire, territoire maritime, fenêtre de jeu, message, menu, score, joueur. Aucune variable globale ne doit être utilisée ;
- le jeu permet le passage de paramètres au lancement du programme pour connaître l'aide, la version du jeu et le nom des auteurs ;
- le positionnement des navires doit respecter les règles de non chevauchement et de bordure.
- aucun des joueurs ne doit pouvoir lancer un missile deux fois au même endroit ;
- tous les messages doivent être fonctionnels : invite de positionnement des navires, erreur de positionnement, invite de lancement de missile, cible déjà utilisée, navire touché, navire coulé, fin de partie et annonce gagnant.

**(Étape 2)** Les options suivantes doivent être ajoutées au jeu :

- on doit pouvoir passer un fichier de configuration au programme qui donne l'ensemble des navires avec lesquels on veut jouer ainsi que la taille du territoire maritime. Il faudra proposer un formalisme pour décrire un navire dans un fichier. Cette option doit permettre de jouer avec d'autres navires que ceux utilisés par défaut. Par exemple, on doit pouvoir utiliser les navires décrits ci-dessous dans une grille  $20 \times 20$  ;



- on doit pouvoir sauvegarder les 5 meilleurs scores dans un fichier et les afficher dans le jeu. Il faudra prévoir la saisie des noms des joueurs ;
- lors du choix de position pour un missile, il ne doit pas être possible de se positionner sur une cible déjà visée. Autrement dit, lors du déplacement avec les flèches les positions déjà utilisées seront évitées automatiquement (passage à la position suivante disponible dans l'axe de la flèche ou aucune action si le déplacement est impossible).

2. Vous trouverez les informations nécessaires à l'adresse [http://en.wikipedia.org/wiki/ANSI\\_escape\\_code](http://en.wikipedia.org/wiki/ANSI_escape_code).

**(Étape 3)** Les options avancées suivantes doivent être possibles :

- il est possible de sauvegarder une partie (dans un fichier) et de la reprendre plus tard ;
- l'ensemble des actions du jeu doivent être enregistrées dans un fichier journal (fichier *log*) permettant le suivi exact de l'évolution de la partie. Le nom du fichier journal pourra être spécifié par une option de la ligne de commande ;
- il est possible d'activer un mode où les deux joueurs sont virtuels et un mode où les deux joueurs sont humains.

**(Étape 4)** Les options suivantes seront nécessaires pour obtenir la note maximale :

- il est possible de défaire des actions du jeu. Cela signifie que l'on doit pouvoir annuler le lancement de missiles et les dégâts occasionnés dans l'ordre inverse de leurs exécutions. Cette option doit permettre de revenir en arrière dans le jeu, jusqu'au tout début. Aide : il va falloir sauvegarder l'ensemble des actions ;
- lors du lancement du jeu, on doit pouvoir choisir un nombre arbitraire de joueurs parmi lesquels un certain nombre seront humains. Par exemple, 4 joueurs dont 1 humain. Dans ce mode, chaque joueur choisi de lancer un missile vers le territoire maritime du joueur de son choix. Dans le cas des joueurs non humains, le choix se fera aléatoirement.

## Le bon sens en action

*A minima*, il vous faut implémenter les demandes exprimées dans les quatre étapes précisées dans le cahier des charges<sup>3</sup>.

Vous veillerez à coder proprement (respect des normes ANSI, commentaires utiles, indentation cohérente, fichiers structurés<sup>4</sup>, ...).



Vous veillerez également à réaliser des tests unitaires pour chaque classe que vous aurez développée.

Vous ne devez en aucun cas recopier du code déjà tout prêt. Toute utilisation de concepts non abordés en cours (*e.g.* : héritage, polymorphisme non paramétrique) ou code non C++ (*e.g.* : code C pur) est proscrit et entraînera la nullité du projet : note de 0/20.

Vous avez le droit d'utiliser la bibliothèque STL à l'exception de la classe `vector`. Bien entendu, il est proscrit d'utiliser toute autre structure de données de la STL pour remplacer la classe `vector`. Il vous sera donc indispensable d'utiliser des tableaux dynamiques pour remplacer la classe `vector`. Bien évidemment, vous pourrez encapsuler vos tableaux dynamiques dans une classe pour simplifier leurs utilisations.

## Boni

Le cahier des charges peut toujours être complété si vous en avez le temps, l'envie et la compétence (il va de soi que cela permet de gagner des points en plus, mais ne compensera **en aucune manière** un manquement au cahier des charges).

Par exemple, la librairie `ncurses` permet d'interagir avec la souris. Rien ne vous interdit d'explorer ces aspects afin de rendre l'interaction avec le terminal accessible à vos proches. Il est aussi envisageable de laisser la possibilité à l'utilisateur de modifier les touches du clavier permettant de contrôler l'interface *via* un menu « paramètres ». Ainsi un joueur chevronné pourra définir la touche  pour (dé-)sélectionner une case, la touche  pour déplacer le curseur vers la gauche, ...

Et si vous souhaitez vraiment dépasser le cadre de l'UE, vous pouvez également implémenter un protocole de communication réseau vous permettant de jouer sur des machines distinctes en mode multi-joueurs.

Ceci nous amène le plus naturellement du monde à aborder les aspects liés à votre. ...

3. Cela va sans dire, mais c'est toujours mieux en le disant...

4. La règle une classe = 1 .h + 1 .cpp est raisonnable.

## 4 Organisation



Concernant votre organisation, le travail est à réaliser en groupes de **4 étudiants**. Vous devrez saisir la composition de votre groupe de travail sur l'espace projet de l'ENT/Moodle du site du cours au plus tard le 29 novembre 2017 à 23h59. Les retardataires verront leur note de projet pénalisée de 2 points.

Enfin, vous devrez rendre sur l'ENT – au plus tard le 8 janvier 2018 à 23h59 – une archive compressée (au format **tar.gz** ou **tar.bz2** uniquement) contenant un répertoire portant le nom de votre groupe (e.g. : groupe-12) et dans lequel on trouvera<sup>5</sup> :

- les sources de votre programme ;
- un fichier texte<sup>6</sup> mentionnant vos noms et prénoms ;
- un fichier texte<sup>7</sup> expliquant comment compiler (en ligne de commande) et utiliser votre programme<sup>8</sup> ;
- un rapport<sup>9</sup> au format **pdf** (10 pages maximum, annexes comprises, hors pages liminaires, police 11 points pour le corps, marges à 2cm) mentionnant vos noms et prénoms et décrivant votre analyse, vos choix, vos tests, les remarques et commentaires, ... Vous discuterez notamment dans ce rapport de l'algorithme que vous aurez défini pour choisir la case visée par l'ordinateur, ainsi que du formalisme que vous aurez choisi pour représenter les bateaux aux formes fantaisistes. Vous mettrez également en avant les différentes étapes que vous avez réalisées et leur degré d'achèvement.

Et comme tout jeu, vous vous devez d'y jouer pour comprendre ce que vous aurez à implémenter.



*Bon Courage. . .*

5. Le non-respect de ces consignes sera indubitablement sanctionné.

6. Typiquement, un fichier `Auteurs.txt`.

7. Typiquement, un fichier `LisezMoi.txt`.

8. L'utilisation d'un `Makefile` est fortement conseillée.

9. Le rapport devra être écrit en français et respecter au mieux les règles typographiques françaises.