INF3135 – Construction et maintenance de logiciels
TP2 – Été 2023

Projet : la bataille navale

Énoncé

L'objectif du projet est de développer un jeu de bataille navale. Le joueur saisit tout d'abord la taille `taille_plateau` du plateau de jeu (min 6; max 100). Le programme ensuite place aléatoirement 5 navires de taille variant de 2 à 5 cases sur ce plateau de jeu. Ce sont :

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

(Voir annexe une courte description des navires).

Il est demandé ensuite au joueur une case sur laquelle il veut lancer une torpille. Le programme ensuite affiche :

- **touché** si sur cette case est placé un navire;
- **à l'eau** si sur cette case n'est pas placé un navire;
- **déjà joué** si cette case a déjà été jouée.
- Une grille de jeu de taille `taille_plateau × taille_plateau` pour laquelle une case est représentée par le signe
    - **X**, si la case a été jouée et si un navire est placé sur cette case;
    - **O**, si la case a été jouée et si aucun navire n'est placé sur cette case;
    - **.**, si la case n'a pas été jouée.

Lorsque toutes les cases d'un navire ont été touchées, un message indique que le navire (en précisant sa taille et son nom) a été coulé. Ce processus est répété tant qu'il reste des navires non coulés.

Lorsque tous les navires ont été coulés, un message indique que vous avez gagné avec le nombre de coup joués. *Exemple :* « `Bravo !! Vous avez gagné en 18 coups.` »

Statistiques

Lorsque le logiciel sera invoqué avec l'option `-S`, il accumulera des statistiques sur les données du jeu et écrira ces statistiques dans un fichier de sortie spécifié à la console.

Exemple d'exécution du logiciel avec l'option `-S` :

```
bash> ./bataille_navale -S stats.txt
```

Si le fichier de sortie n'existe pas, il sera créé; s'il existe, il sera écrasé.

Voici les statistiques à produire :

- le nombre total de coups réalisé pour couler tous les navires;
- le nombre de lettres sans doublon du nom du premier navire touché ;
- le nombre total de coups « à l'eau »;
- le nombre total de coups « déjà joué »;
- le nombre total de coups « touché »;
- le nom du dernier navire coulé.

Modélisation du problème

a. Structure de données

Une case du jeu est modélisée par la structure suivante :

```
typedef struct une_case {
  int x; /* position de la case en x*/
  int y; /* position de la case en y*/
} Case;
```

Un navire est modélisé par la structure suivante :

```
typedef struct navire {
  int sens; /* 0 haut 1 droite 2 bas 3 gauche */
  Case premiere_case;
  int taille;
} Navire;
```

Le plateau du jeu *plateau* de taille *taille_plateau × taille_plateau* sur lequel le programme place les navires est alloué dynamiquement.

Le logiciel doit être découpé en modules. On devrait, au minimum, retrouver un module pour chaque élément suivant :

- le main;
- le jeu (bataille navale);
- la gestion des statistiques.

La grille de jeu *grille* de taille *taille_plateau × taille_plateau* est allouée dynamiquement.

b. Fonctions à utiliser

La fonction **int nb_aleatoire(int max)** qui renvoie un nombre, tiré au hasard, compris entre 1 et max.

```
#include<stdlib.h>
#include<time.h>

/* Initialiser le générateur au début du main par la fonction suivante*/
 void init_nb_aleatoire() {
    srandom(time(NULL));
}

/* Renvoie un nombre, tiré au hasard, compris entre 1 et max*/
int nb_aleatoire(int max) {
    return (random()%max);
}
```

Voici la liste de quelques prototypes des fonctions à utiliser lors de ce projet. Vous avez la liberté d'en écrire de nouvelles selon vos besoins.

- **Navire creer_navire(int taille, int taille_plateau) :** cette fonction permet de créer un navire d'une taille donnée dont la case de départ et le sens sont fixés aléatoirement.

- **int est_valide(int **plateau, int taille_plateau, struct navire * nav):** cette fonction retourne 1 si le navire est bien situé dans les limites du plateau, et qu'il ne se chevauche pas avec un autre navire, sinon elle retourne 0.

    - `plateau` est une matrice représentant le plateau de jeu, dans laquelle les cases occupées par des navires contiennent un 1 et les autres un 0.

- **void initialisation_plateau(int **plateau, int taille_plateau):** cette fonction initialise aléatoirement cinq navires de taille 2 à 5 dans le plateau.

- **void proposition_joueur(int **plateau, int **prop, int *nb_touche, int *nb_joue, int *nb_touche_nav, int taille_plateau) :** cette fonction demande à l'utilisateur de saisir une case (x,y) à jouer et selon la valeur contenue `plateau[x][y]` enregistre dans `prop[x][y]` la valeur :

    - 0 si la case ne contient pas de navire
    - -1 si la case a déjà été jouée
    - 1 si la case contient un navire

  Note :
    - nb_joue est le compteur du nombre de coups
    - nb_touche est le compteur de cases touchées
    - nb_touche_nav est un tableau qui contient le nombre de cases touchées pour chaque navire. nb_touche_nav[i] indique le nombre de cases touchées pour le navire de taille i.

- **void affichage_plateau(int **plateau, int taille_plateau)** :

Améliorations possibles

- Écriture d'une fonction qui affiche différemment les cases coulées et les cases touchées.
- Sauvegarde et chargement de parties en cours.

Exigences du code source

Vous devez appliquer les exigences suivantes à votre code source.

- L'indentation doit être de 3 espaces. Aucune tabulation n'est permise dans l'indentation.
- Votre code doit être découpé en fonctions d'une longueur acceptable. Pas de fonctions de plus de 15 lignes.
- Chaque fonction doit être documentée avec un commentaire expliquant l'objectif de la fonction, les paramètres et la valeur de retour (si applicable). Le nom de la fonction doit être significatif.
- N'utilisez pas de variables globales (sauf errno).
- Les erreurs systèmes doivent être gérées correctement.
- Vous devez adapter une approche de programmation modulaire. Utilisez de fichier d'en-tête (.h) pour représenter vos interfaces et cacher vos implémentations dans les fichiers (.c). Vos modules devraient suivre le standard vu en classe.
- Les identifiants de fonctions et variables doivent être en snake_case.
- Une attention particulière doit être apportée à la lisibilité du code source.
- Vous devrez utiliser **`bats`** et **`Cunit`** (80% de couverture et code de test propre) comme cadres de test pour tester votre logiciel
- Vous devez adopter une approche de développement par branche et initier des *`merge request`* pour toutes vos fusions dans la branche principale. Plus spécifiquement, tout module, documentation et correctifs doivent faire l'objet d'une nouvelle branche.
- Vous devez ajouter une intégration continue (IC) de votre projet. L'IC doit être constituée d'au moins 2 tâches : une tâche de construction et de test.
- Vous devez fournir un Makefile qui exprime les dépendances de façon complète.

Exigences techniques (Pénalité 20%)

- Votre travail doit être rédigé en langage C et doit **compiler sans erreur et sans avertissement** sur le serveur Java de l'UQAM (java.labunix.uqam.ca). Pour vous y connecter, vous devez connaître votre CodeMS (de la forme ab123456) ainsi que votre mot de passe (de la forme ABC12345)
- Votre dépôt doit se nommer exactement `inf3135-ete2023-tp2`
- L'URL de votre dépôt doit être exactement `https://gitlab.info.uqam.ca/<utilisateur>/inf3135-ete2023-tp2` où `<utilisateur>` doit être remplacé par votre identifiant
- Votre dépôt doit être privé
- Les usagers *`@lapointe.gabriel.2`* et *`dogny_g`* doivent avoir accès à votre projet comme *`Developer`*

Remise

Le travail est automatiquement remis à la date de remise prévue. Vous n'avez rien de plus à faire. Assurez-vous d'avoir votre travail disponible sur votre branche ***main*** qui sera considérée pour la correction. Tous les *commit*s **après le 9 juillet 2023 à 23:55** ne seront pas considérés pour la correction.

Barème

Critère		Points
Fonctionnalité		/30
Améliorations possibles		/10
Qualité du code		/10
Utilisation de git		/10
Tests	Cunit	/10
	Bats	/10
GitLab-CI		/10
Makefile		/5
Documentation		/5
Total		/100

Plus précisément, les éléments suivants seront pris en compte:

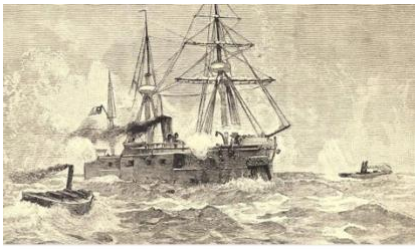
- **Fonctionnalité (30 points):** Tous les programmes compilent et affichent les résultats attendus.
- **Améliorations possibles (10 points):** Implémentation des améliorations suggérées dans l'énoncé.
- **Qualité du code (10 points):** Les identifiants utilisés sont significatifs et ont une syntaxe uniforme, le code est bien indenté, il y a de l'aération autour des opérateurs et des parenthèses, le programme est simple et lisible. Pas de bout de code en commentaire ou de commentaires inutiles. Pas de valeur magique. Le code doit être bien factorisé (pas de redondance) et les erreurs bien gérées. La présentation doit être soignée.
- **Documentation (5 points):** Le fichier `README.md` est complet et respecte le format Markdown. Il explique comment compiler et exécuter vos programmes ainsi que toutes les autres cibles demandées.
- **Utilisation de Git (10 points):** Les modifications sont réparties en *commits* atomiques. Le fichier `.gitignore` est complet. Utilisation des branches. Les messages de *commit* sont significatifs et uniformes. Les demandes d'intégration.
- **Tests (20 points):** Le code de test est propre, avec une couverture de 80%. **Cunit** et Bats.
- Makefile (5 points)**:** Le Makefile doit supporter au moins les cibles ***compile***, ***link***, ***test*** et une cible ***html*** permettant de créer une version html du `README.md`. L'appel à **make** doit compiler et construire l'exécutable.
- **GitLab-CI (10 points):** Votre projet doit supporter le système d'intégration continue de GitLab (GitLab-CI) qui construit et roule tous vos tests à chaque commit sur la branche ***master***.

***Annexe* :**

Un **porte-avions** ou un **porte-aéronefs** est un navire de guerre permettant le lancement et la réception d'aéronefs à partir de son pont. Ces bâtiments sont dotés d'une puissance militaire considérable dont les capacités multiples en font des instruments d'une grande souplesse d'utilisation militaire o

Un **destroyer** ou **contre-torpilleur** est un navire de guerre capable de défendre un groupe de bâtiments contre toute menace, comme d'attaquer un groupe de navires moyennement défendus. Il possède des moyens de lutte antiaérienne, anti-sous-marine et anti-navire. À l'origine, il s'agit

Un **croiseur** est un navire de guerre. Depuis le début des années 1990 et la disparition des cuirassés, c'est le plus puissant et le plus grand des bâtiments de combat, exception faite des porte-avions.



Un **torpilleur** est un petit navire de guerre relativement rapide et manœuvrant, chargé de combattre les grosses unités navales de surface en utilisant, comme arme principale, la torpille. Ce type de navire n'existe plus aujourd'hui car, trop spécialisé, il a été dépassé. Il a été remplacé par les