Bataille Navale S5 2023-2024

Introduction

Pendant le cours d'informatique du premier semestre en Anee 3 de la specialite Robotique à Polytech Sorbonne, l'idee principal est d'apprendre les bases de programmation en c, ainsi que de mettre de bases d'analyses ou encore imaginer des algorithmes qui servirons de repondre à un cahier de charges precis.

Dans ce cadre là, on est demandé de developer un mini prjet qui nous permeta d'exploiter nous competences dans le domaine de la programmation et la collaboration dans un environment complex en ce qui concerne l'organisation des fichiers et leur liason. Le projet consiste à recréer le jeu de la bataille navale en informatique. À partir d'informations essentielles telles que la taille du tableau de jeu et ses règles, il faut placer aléatoirement six navires de tailles variant de 2 à 6 cases sur ce plateau de jeu (version principale).

Dans notre jeu de bataille navale, vous trouverez d'autres modes de jeu que nous avons explorés :

- Multijoueur
- Sauvegarder/Continuer
- IA

Avant commencer il faudrait de se mettre en commun en ce qui concerne les regles du jeu. Comment on peut decider si quelqu'un a gagne ou pas quand il joue en solo ou contre un autre utilisateur ? Existe-t-il d'autres manières de déterminer le vainqueur ?

Règles du jeu

Inspiré par la version simple du jeu que nous avions programmée pour le TP2, l'algorithme qui décide si le joueur a gagné ou non doit être en relation avec le nombre total d'essais. Il faut prendre en compte que la taille du tableau de jeu n'est pas toujours la même. Ainsi, l'algorithme proposé décide en fonction de la taille du plateau.

```
// Vérifie si le nombre maximum de tours a été atteint et si l'utilisateur
n'a pas trouvé tous les navires
if (tour_actuel == tours_max && nombre_de_navires_trouvés <
nombre_de_navires) {
    // L'utilisateur a épuisé ses tours
    continuer = false;  // Arrête la boucle de jeu
    afficher_message_defaite();  // Affiche le message de défaite
}

// Vérifie si l'utilisateur a trouvé tous les navires
if (nombre_de_navires_trouvés == nombre_de_navires) {
    // L'utilisateur a gagné le jeu
    afficher_message_victoire(taille_plateau, proportion, *NbJoue - 1, 1,
"");  // Affiche le message de victoire et termine le jeu
    return 0;  // Indique que le jeu est gagné
}
```

De plus, nous nous sommes inspirés des jeux d'arcade qui sont souvent associés à un temps limité. C'était une excellente occasion d'exploiter la bibliothèque `time.h` tout en intégrant un système qui arrête le jeu s'il n'y a plus de temps. Que faire en cas de mode pause ? Vous trouverez ci-dessous l'algorithme qui répond à toutes ces questions.

Algorithme de gestion du temps :

```
// Vérifie si le temps restant a expiré
if (temps_restant <= 0) {
    // L'utilisateur a épuisé son temps
    continuer = false;  // Arrête la boucle de jeu
    afficher_message_defaite();  // Affiche le message de défaite
}
```

Indications

Pour la création du jeu, il y avait quelques fonctions de base à créer et à respecter, plus précisément :

- `int est_valide(int **plateau, int taille_plateau, struct navire *nav)`
- `Navire creer_navire(int taille, int taille_plateau)`
- `void init_nb_aleatoire()`
- `int nb_aleatoire(int max)`

- `void initialisation_plateau(int **plateau, int taille_plateau)`
- `void proposition_joueur(int **plateau, int **prop, int *NbTouche, int *NbJoue, int *NbToucheNav, int taille_plateau)`
- `void affichage_plateau(int **plateau, int taille_plateau)`

Fonction `est_valide_pro`

La fonction `est_valide` (appelée `est_valide_pro` dans notre programme) est utilisée à l'intérieur de la fonction `proposition_joueur` :

```
bool proposition_joueur(int **prop, int *NbJoue, Liste_Navire L, int
taille_plateau, int *NbNav)
{
    int x, y;
    bool coordonnees_valides = true;  // utilisé pour demander à
l'utilisateur de nouvelles coordonnées pour un navire si les précédentes ne
sont pas dans les limites spécifiées
    bool navire_trouve;
    char input[3];
    int code_statut;
    bool répéteur = true;

    while (coordonnees_valides)
    {
        custom_graphics_on_proposition(-1, prop, taille_plateau, -1, 0, -1);
        scanf("%s", input);

        while (répéteur)
        {
            code_statut = décryptage_point(input, &x, &y);
            if (code_statut == 0)
            {
                répéteur = false;
            }
            else if (code_statut == 8)
            {
                custom_graphics_on_proposition(-1, prop, taille_plateau, -1,
1, -1);
                scanf("%s", input);
            }
            else if (code_statut == 9)
            {
                custom_graphics_on_proposition(-1, prop, taille_plateau, -1,
2, -1);
                scanf("%s", input);
            }
            else if (code_statut == 7)
            {
```

```
                custom_graphics_on_proposition(-1, prop, taille_plateau, -1,
3, -1);
                scanf("%s", input);
            }
            else
            {
                custom_graphics_on_proposition(-1, prop, taille_plateau, -1,
4, 1);
            }
        }

        répéteur = true;

        if (!(x < 1 || x > taille_plateau || y < 1 || y > taille_plateau))
// vérification des coordonnées hors limites
        {
            coordonnees_valides = false;
        }
        else
        {
            custom_graphics_on_proposition(-1, prop, taille_plateau, -1, 5,
-1);
        }
    }

    coordonnees_valides = true;

    x--;  // Ajustement pour les indices.
    y--;

    update_prop(prop, x, y);
    (*NbJoue

)++;  // tour suivant

    navire_trouve = navire_trouve(prop, L);
    if (navire_trouve == true)
    {
        (*NbNav)++;
        return true;
    }

    clearScreen();
    return false;
}
```

Le code fourni représente une partie de la fonction `proposition_joueur` qui gère la saisie de l'utilisateur pour jouer un coup dans un jeu de bataille navale. Elle permet au joueur de saisir un point de coordonnées (x, y) et met à jour le tableau `prop` en conséquence.

- La fonction `décryptage_point` est appelée pour valider le format d'entrée et extraire les composants numériques et alphabétiques.

- Le code d'état `code_statut` renvoyé par `décryptage_point` est vérifié pour détecter les erreurs :
  - `7`: L'entrée n'est pas au bon format (c'est-à-dire qu'elle ne comporte pas deux caractères).
  - `9`: Le premier caractère n'est pas un chiffre.
  - `8`: Le deuxième caractère n'est pas une lettre minuscule.

La fonction vérifie le status du point sur plateau pour les coordonees en question et le met à jour.

- Si `prop[x][y] = NAVIRE`: mettez `prop[x][y] = NAVIRE_TROUVE` pour indiquer un navire touché.
- Si `prop[x][y] = NAVIRE`: mettez `prop[x][y] = NAVIRE_TROUVE_PLUS_1` pour indiquer un navire précédemment touché.
- Sinon (pas de navire): mettez `prop[x][y] = AUCUN_NAVIRE` pour indiquer une cellule vide.

Pourquoi est_valide_pro et pas est_valide ?

`est_valide_pro` est une fonction qui répond au cahier des charges de la fonction demandée `est_valide`, sauf que celle-ci traite chaque cas séparément tout en effectuant la vérification du placement d'un navire à une position dès le moment où les coordonnées x, y aléatoires sont créées.

Fonction `navire_trouve`

La dernière étape est de vérifier si un navire a été coulé selon l'algorithme `navire_trouve` :

```
bool navire_trouve(int x, int y)
{
    bool navire_coule = false;

    if (prop[x][y] == AUCUN_NAVIRE)
    {
        navire_coule = true;
    }
    else if (prop[x][y] == NAVIRE_TROUVE)
    {
        navire_coule = true;
        nombre_touches = nombre_touches + 1;
    }
    else
```

```
    {
        navire_trouve = false;
    }

    return navire_coule;
}
```

1. La fonction initialise une variable `navire_coulé` à `faux`. Cette variable servira à stocker si le navire a coulé.
2. La fonction vérifie si la case aux coordonnées `x, y` est vide (`prop[x][y] = AUCUN_NAVIRE`). Si c'est le cas, le navire a coulé et `navire_coulé` est mis à `vrai`.
3. La fonction vérifie également si la case aux coordonnées `x, y` a déjà été touchée (`prop[x][y] = NAVIRE_TROUVE`). Si c'est le cas, le navire n'a pas coulé, mais le nombre de coups sur le navire est incrémenté (`nombre_touches = nombre_touches + 1`).
4. Si aucune des conditions précédentes est verfié, aucun point des navires été trouvé et `navire_trouve` est mis à `faux`.

Fonction `ajuster_tours`

Comme prévu, dans notre jeu de bataille navale, l'utilisateur peut choisir la taille du plateau de jeu. De plus, le sujet mentionnait qu'il doit toujours y avoir 6 navires. Nous avons donné à l'utilisateur la possibilité de choisir le nombre de navires maximal (pas seulment 6). En fait, selon tous ces paramètres, quelqu'un peut avoir soit un jeu très facile, soit un jeu difficile.

Par exemple: imagine que le nombre maximum de tours reste statique à 55 pour un plateau de atille 4x4. On a lance de trouver tous les navires.

Alors, la fonction `ajuster_tours` décide du nombre maximum de tours en fonction des paramètres du jeu choisis par l'utilisateur. Son algorithme est le suivant :

1. Vérifier le mode de jeu (`mode`) :
    - Si `mode` est égal à 1 (mode solo), ajuster le nombre maximum de tours en fonction de la taille du plateau de jeu (`taille_plateau`) et du nombre de navires (`nb_navires`).
    - Sinon, si `mode` est égal à 2 (mode multijoueur), fixer le nombre maximum de tours à 40.
2. Gérer les cas du mode *solo* :

    - Si la taille du plateau (`taille_plateau`) est inférieure ou égale à 4 :
        - Basculer entre différentes valeurs de `max_tours` en fonction du nombre de navires (`nb_navires`).
    - Si la taille du plateau (`taille_plateau`) est supérieure à 4 et inférieure ou égale à 10 :
        - Basculer entre différentes valeurs de `max_tours` en fonction du nombre de navires (`nb_navires`).
    - Sinon (taille du plateau supérieure à 10) :
        - Fixer `max_tours` à 60.
3. Affecter la valeur calculée de `max_tours` à la variable globale.

Procédure

Chaque projet informatique est exigeant et nécessite la contribution de chaque membre de l'équipe pour éviter des erreurs de logique dans le projet final et sur le code source. En ce qui concerne la communication, les messages directs sont la manière la plus simple pour une équipe composée de deux membres. Mais comment fait-on pour prendre en compte toutes les choses que l'on a discutées en direct ou en messages ?

L'utilisation de Git est essentiel qui nous permettra de reprendre le projet, tout en contribuant chaqu'un independement et avoir un archive de chaque chagrment qui était effectué. Git aussi permet d'analyser le code avant compiler et faire un backup sur une service destiné de maintenir le code et ses changements toujours accesibles par n'importe quel appareil. Comme attendue, Git etait associe pour un repertoire Github avec un Copilot gcc pour l'analyse du code envoyé vers le compte Github.

Programmation

UI

L'UI définit un ensemble de fonctions liées à l'interface graphique du jeu "Bataille Navale". Ces fonctions sont responsables de l'affichage d'informations à l'utilisateur, telles que le plateau de jeu, les instructions et les messages. Très pratique si on veut modfier un message ou addresser un probleme sur la partie de l'affichage par exemple.

Voici une brève description des fonctions declares et construits :

custom_graphics_on_proposition:

Cette fonction permet d'afficher différents messages et invites à l'utilisateur en fonction de l'état actuel du jeu. Elle prend les paramètres suivants :

- `i`: Un entier représentant l'itération actuelle du processus de placement
- `plateau`: Un tableau bidimensionnel représentant le plateau de jeu
- `taille_plateau`: La taille du plateau de jeu
- `colour`: La couleur du joueur
- `mode`: Le mode de jeu (solo ou multijoueur)
- `id`: Un identifiant pour des messages d'erreur spécifiques

Détermine le message qu'il faut afficher selon les combinaisons des paramètres.

game_mode_graphics_congratulations:

Cette fonction permet d'afficher un message de félicitations le moment qye l'utilisateur trouve un navire. Elle prend les paramètres suivants :

- `prop`: Un tableau bidimensionnel représentant le plateau de jeu
- `taille_plateau`: La taille du plateau de jeu
- `nb_navires`: Le nombre total de navires
- `nb_navires_trouvés`: Le nombre de navires trouvés jusqu'à présent
- `id`: Un identifiant pour le type de navire trouvé (navire du joueur ou navire de l'adversaire)
- `buffer`: Une chaîne contenant le type de navire trouvé

error_graphics:

Il afficher des messages d'erreur à l'utilisateur. Elle prend le paramètre suivant :

- `error_code`: Un entier représentant le message qu'il faut afficher par la libraire de base des erreurs.

new_round_graphics:

Cette fonction permet d'afficher un message indiquant le début d'une nouvelle manche. Elle prend les paramètres suivants :

- `round`: Le numéro de la manche actuelle
- `taille_plateau`: La taille du plateau de jeu

- `prop`: Un tableau bidimensionnel représentant le plateau de jeu
- `id`: Un identifiant pour le mode de jeu (tour du joueur ou tour de l'adversaire)
- `buffer`: Le nom du joueur dont c'est le tour
- `temps`: Le temps restant dans la manche (pour le mode solo)

La fonction affiche un message indiquant le numéro de la manche et, si applicable, le temps restant. Elle affiche également le plateau de jeu.

Structures

Selon le sujet, on etait demandé de declarer les structures suivantes:

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

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

De plus de ça, on a aussi declaré les structures qui nous avons permet de traiter des cas dynamique en differents modes de jeu tels que:

- Les structures `Cellule_Liste_Navire` et `Cellule_Liste_Point` sont utilisées pour créer des listes chaînées de navires .
- La structure `Liste_Navire` est utilisée pour gérer une liste de navires. Elle permet d'ajouter des navires dans la liste chainé des navires.
- La structure `Liste_Point` est utilisée pour gérer une liste de points. Elle permet d'ajouter et de supprimer des points (couple x,y) dans la liste chainé des points.
- La structure `Tableau_Point` est utilisée pour transformer une liste chainé des points à un tableau statique.

```
typedef struct Cellule_Liste_Navire_
{
        Navire data;
        struct Cellule_Liste_Navire_* suiv;
```

```
} Cellule_Liste_Navire;

typedef struct Liste_Navire_
{
        unsigned int taille;
        Cellule_Liste_Navire *first;
        Cellule_Liste_Navire *last;
                                /* first = last = NULL et taille = 0 <=>
liste vide */
} Liste_Navire;


/*---- le type cellule de liste de point ----*/
typedef struct Cellule_Liste_Point_
{
        Case data;
        struct Cellule_Liste_Point_* suiv;
} Cellule_Liste_Point;

/*---- le type liste de point ----*/
typedef struct Liste_Point_
{
        unsigned int taille;
        Cellule_Liste_Point *first;
        Cellule_Liste_Point *last;
                                /* first = last = NULL et taille = 0 <=>
liste vide */
} Liste_Point;

/*---- le type tableau de point ----*/
typedef struct Tableau_Point_
{
        unsigned int taille;
        Case *tab;
} Tableau_Point;
```

La structure Cellule_Liste_Navire

Représente un élément d'une liste de navires. Elle contient deux champs :

- `data`: Une structure `Navire` qui représente les données du navire.
- `suiv`: Un pointeur vers l'élément suivant de la liste.

La structure Liste_Navire

Représente une liste de navires. Elle contient trois champs :

- `taille`: Le nombre d'éléments dans la liste.
- `first`: Un pointeur vers le premier élément de la liste.
- `last`: Un pointeur vers le dernier élément de la liste.

La structure Cellule_Liste_Point

Représente un élément d'une liste de points. Elle contient deux champs :

- `data`: Une structure `Case` qui représente les données du point.
- `suiv`: Un pointeur vers l'élément suivant de la liste.

La structure Liste_Point

Représente une liste de points. Elle contient trois champs :

- `taille`: Le nombre d'éléments dans la liste.
- `first`: Un pointeur vers le premier élément de la liste.
- `last`: Un pointeur vers le dernier élément de la liste.

La structure Tableau_Point

Représente un tableau de points. Elle contient deux champs :

- `taille`: Le nombre d'éléments dans le tableau.
- `tab`: Un pointeur vers le tableau des éléments.

Saving API

L'option de sauvegarde était très importante pour nous et était basée sur le "filing codec" que nous avons inventé. La procédure est très simple. Pour sauvegarder un jeu, il suffit d'ouvrir le menu et de sélectionner "Sauvegarder" ou "Save". Notre fonction `api_save_game` va encoder le jeu selon le codec.

Si quelqu'un veut reprendre à partir du jeu sauvegardé, il lui suffit de choisir l'option "Continuer" ou "Load" depuis le menu principal. Ce mode de jeu effectue toutes les initialisations en fonction des données lues par la fonction `api_load_game`. Pour améliorer la logique des différents modes de jeu, d'autres fonctions supplémentaires ont été ajoutées. Par exemple: qu'est qu'il se passe si un utilisateur essaye de lancer un jeu à partir d'une sauvegarde qui n'existe pas, ou imagine qu'il a joué, il a sauvegardé et après il a continué selon le fichier de sauvegrade et il ferme le jeu jeu avant finir. Il faut prevoir de supprimer le fichier le moment que les donnes sont lu.

api_load_game

Algorithme :

1. Ouvrir le fichier `filecodec239012V1` en lecture, sinon afficher un message d'erreur et quitter le programme.
2. Initialiser une liste `liste` vide de navires.
3. Allouer une nouvelle structure `Navire` et initialisez ses champs avec les valeurs lues du fichier.
4. Ajouter le navire à la liste `liste`.
5. Répéter les étapes 4 et 5 jusqu'à ce que le fichier soit lu.
6. Fermer le fichier.
7. Retourner la liste `liste` des navires.

api_table_size

Algorithme :

1. Ouvrir le fichier `filecodec239012V1` en lecture, sinon afficher un message d'erreur et quitter le programme.
2. Lire le nombre de navires du fichier.
3. Fermer le fichier.
4. Retourner le nombre de navires.

api_clearFile

Algorithme :

1. Ouvrir le fichier `filecodec239012V1` en écriture, ce qui supprime le contenu du fichier s'il existe déjà, sinon afficher un message d'erreur et quitter le programme.
2. Fermer le fichier.

api_save_game

Algorithme :

1. Ouvrir le fichier `"filecodec239012V1.txt"`, sinon il affiche un message d'erreur et quitter le programme.
2. Supprimer son contenu si besoin.
3. Écrire le nombre de navires, la taille du plateau, le nombre de navires coulés, le numéro de la manche et les informations sur chaque navire dans le fichier.
4. Écrire "$" pour indiquer la fin des informations sur les navires.

5. Écrire le tableau du plateau de jeu dans le fichier.
6. Fermer le fichier.

api_delete_game_file

Algorithme :

1. Supprimer le fichier `"filecodec239012V1.txt"`.
2. Afficher un message de réussite.
3. Retourner.

Filing codec

```
number_of_navires taille_plateau coulle round sens_nav_1 x_nav_1
y_nav_1 taille_nav_1 id_nav_1 sens_nav_2 ... $ prop_table_codec
```

Notre filing codec représente un format structuré pour encoder et décoder les données du jeu Battleship. Il suit une séquence spécifique d'éléments pour garantir l'intégrité et la lisibilité des fichiers de jeu enregistrés.

Qui est inclus sur le codec ?

1. **Nombre de navires :** Cet entier indique le nombre total de navires présents dans le jeu.
2. **Taille du tableau :** Cet entier représente la dimension du plateau de jeu, généralement représenté comme `x` par `y`.
3. **Navires coulés :** Cet entier signifie le nombre actuel de navires coulés.
4. **Numéro de manche :** Cet entier indique la manche actuelle du jeu.
5. **Informations sur le navire :** Pour chaque navire, les attributs suivants sont codés :

   a. **Orientation (sens_nav) :** Représente l'orientation du navire (horizontale ou verticale).

   b. **Coordonnée X (x_nav) :** Indique la coordonnée X de départ de la tête du navire.

   c. **Coordonnée Y (y_nav) :** Spécifie la coordonnée Y de départ de la tête du navire.

   d. **Taille du navire (taille_nav) :** Représente la longueur du navire.

e. **ID du navire (id_nav) :** Identifie le navire de manière unique parmi tous les navires du jeu.

6. **Marqueur de fin des données de navire ($) :** Ce caractère marque la fin de la section des informations sur le navire.

7. **Données du tableau de jeu :** Cette section encode l'état réel du plateau de jeu, représenté comme une matrice d'entiers. Chaque élément de la matrice correspond à une cellule sur le plateau, avec des valeurs indiquant si la cellule est vide, touchée ou coulée.

Une identification-du-temps ajouté au 'filing codec' fournirait un contexte supplémentaire sur le jeu enregistré.

Mode Solo

Il existe deux sous-modes du mode solo :

- **Mode classique :** Le joueur dispose d'un nombre illimité de tours pour couler les navires

de l'ordinateur.

- **Mode chronométré :** Le joueur a un temps limité pour couler les navires de l'ordinateur.

Dans le mode classique, le joueur peut prendre son temps pour jouer et tenter de couler tous les navires de l'ordinateur. Dans le mode chronométré, le joueur doit jouer plus rapidement et faire attention à ne pas manquer de temps. Les deux variantes du mode solo sont organisées comme suit :

1. **Initialisation :** Le joueur et l'ordinateur placent leurs navires sur le plateau de jeu.
2. **Tour du joueur :** Le joueur joue un coup en sélectionnant une case sur le plateau de jeu.
3. **Tour de l'ordinateur :** L'ordinateur joue un coup en sélectionnant une case sur le plateau de jeu.
4. **Fin de la partie :** La partie se termine lorsque tous les navires d'un joueur sont coulés.

Mode Load

Le mode load dans notre jeu permet à l'utilisateur de continuer un jeu qui etait enregistrée. Le fichier de sauvegarde contient les données suivantes (plus de détails sur la partie Filing codec)) :

- La taille du plateau de jeu
- Le nombre de navires
- L'emplacement des navires
- Le nombre de navires coulés
- Le nombre de tours joués

On pourrait ameliorer ce mode de jeu si:

- on faisait des sauvegardes à chaque lapse du temps (ex: chaque 10 seconds)
- on avait un deuxieme fichier avec l'archive de chaque mouvement, permetant d'affichier les mouveemnts de l'utilisateur sur le terminal.

Mode Multiplayer

Le mode multiplayer dde notre jeu permet à deux utilisateur de joueur l'un contre l'autre. L'utilisateur 1 crée le tableau pour l'utilisateur 2 et vice-versa. L'algorithme qui decide qui gagne prend en compte le nombre de navires coulés, ainsi que le nombre des tours passés.

Algorithme :

1. Initialiser un compteur de navires coulés pour chaque joueur.
2. Pour chaque joueur, itérer sur les navires de son plateau de jeu et vérifier si le navire est coulé.
    - Si un navire est coulé, incrémenter le compteur de navires coulés correspondant.
3. Comparer les compteurs de navires coulés des deux joueurs.
    - Le joueur qui a coulé le nombre le plus grandes des navires gagne au cas ou il ne reste plus des tours. Sinon gagne celui qui trouve tous les navires au premier.

Pseudo-code :

```
compteurCoulesJoueur1 = 0
compteurCoulesJoueur2 = 0

pour chaque navire du plateau du joueur 1 :
    si le navire est coulé :
        incrémenter compteurCoulesJoueur1

pour chaque navire du plateau du joueur 2 :
    si le navire est coulé :
        incrémenter compteurCoulesJoueur2

si compteurCoulesJoueur1 >= nombreNaviresJoueur2 :
    joueur1 gagne
sinon si compteurCoulesJoueur2 >= nombreNaviresJoueur1 :
    joueur2 gagne
sinon :
    le match continue
```

Mode IA

Problèmes - Solutions + Notes

1. La création de six navires sur un plateau de taille inférieure à 6 n'était pas garantie. Nous avons découvert que le problème provenait de la variable booléenne responsable de l'incrémentation du compteur de navires créés. Cette variable n'était pas réinitialisée, ce qui provoquait un problème logique lorsque le programme devait prendre une nouvelle taille, direction ou coordonnées pour un navire.
    - La variable booléenne responsable de l'incrémentation du compteur de navires créés n'était pas réinitialisée, ce qui provoquait un problème logique lors de la création de six navires sur un plateau de petit taille.
2. Another issue that we had was with the detection of existign navires that were placed from the user manually (on multiplayer or AI mode). It was like the function est_valide_pro wasn't returning a response.
    - Eventually, we discovered that we used another identification code for the navires placed manually from the user - allowing us simply to distinhue the different operations - that were retransformed to real navire points once the lacement was completed.

    - The solution was to include the logical 'or' on the `est_valide_pro` function to detect those points as well.

3. A liste chainne with type Liste_Navire is used in order to make all the required verifications. Ideally we would like to use the method that deletes a navire from that liste in irder to minimise the processing time. However we need to fix the function responsible for doing that `Liste_Point supprimer_liste_Navire(Liste_Navire L)`.

4. To be noted that this procedure allows us to have only one plateau where everything is parsed dynamicly. The liste looks like the following:

ID	Data
0	Data of navire 0 -> 1
1	Data of navire 1 -> 2
2	Data of navire 2 -> 3
3	Data of navire 3 -> 4
4	Data of navire 4 -> 5
5	Data of navire 5 -> NULL

5. A new verison of the printing function is available called printing_the_grille_v2 that is more UI friendly and it's the one that is used on the latest version of the code

Exit codes

Below you will find the definition an dthe explanation of ou rpogram exit codes:

Codes	Status	Explanation	Solution
exit(-1)	ERROR	There is an allocation memory error	Check output on terminal that indicates the row, column and table with the error
exit(5)	OUTPUT	The game has been saved succesfully on the server	N/A

Codes	Status	Explanation	Solution
exit(-2)	STATE	No file of saved game found	Play a game at least once and save it, then try again in order to load it and continue from were you stoppe dthe last time.
7	STATE	the coordinates for a point are not wth a length of 2	
8	STATE	Point choosen first carcater is not a letter	
exit(-4);	ERROR	Unidentfied error with the point declaration	Check the logic on the function proposition_joueur
9	STATE	Point choosen first carcater is not a number	
exit(-3)	ERROR	Multiplayer sub mode failed	Check the output of the function and try again
exit(-4)	ERROR	Solo sub mode failed	Check the output of the function and try again

Versioning

Le versioning est une pratique importante en programmation informatique. Il permet de garantir la cohérence des modifications apportées à un programme et de faciliter la collaboration entre nous, ainsi que de récupérer des données perdues ou corrompues.

Voici les differents versions pendant l'evolution du code:

- **V1.1** Code de base
- **V1.2** Code corrigé selon indications de Manu
- **V1.3** Code optimized from the previous source code
- **V1.4** Comments added and cleaned source code
- **V2.1** Bug fixes regarding est_valid_pro
- **V2.2** Makefile updated

- **V3.1** Explained different things and added the function copier_navires . The part 'To be created' was updated. Check the proto on that section of what needs to be done.
- **V3.2** Fixes copier_navire error and added the main game loop including the functions discused on the board. NOTE: Code is not optimised and it can not be runned until teh game loop logic is completed
- **v4.5** Game logic has been completed and improvements were made on the source code
- **V5.0** All the UI has been completed. Introduction to different game modes was included on this version. Needs to be done: create the saving functionality (check to be done section) and comeplete the multiplayer mode.
- **V6.1** load game functionality has been added and fixed some bugs regarding the UI as well as the repsonse of the program n several extreme scenarions
- **V6.2** Loading and saving game functionality was completed, fixed some bugs on the get_user_input ui and we fixed some UI customisation issues as well.
- **V6.3** Added the temps choice on Solo mode and there were some bug fixes as well
- **v7.0** multiplayer added and more functionality was added on the ui with intelligent logic printing modes as well as there were some bug fixes on the source code programmes.c
- **v7.1** Imporved the program to have the exact signatures that the prof asked for and added some functionality for the points that were selected at least twice. Moreover the logic for the end of LOAD mode new logic was added to prevent a user to play again with a game file that was played before.
- **v7.2** Tried to fix the non paused time for solo mode option temps with no sucess
- **v7.3** Added a functionality to accept the format 2B for points coordinates
- **V7.4** AI added and the module to create navires customly
- **V8.0** Fixed issue with initialisatio_plateau_custom which couldn't detect the already existed navires, updated the UI files and transfered the majority of texts to the UI file as well. In addition to that, the code was optimised and there were several small bug fixes. The different game modes were seperated in order to be able to test them without the whole's game menu.

Copyright