Bataille Navale en C

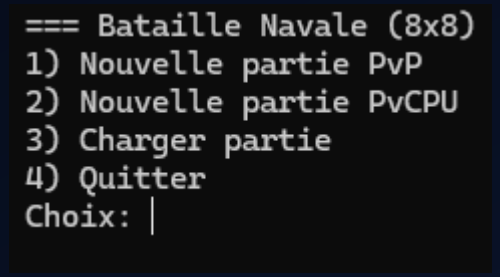
Conception et développement d’application

Ben Ahmed

RAYAN BENNACEUR , MAURANE COMOE, THOMAS COSTELET, JASSIM HAMDOUNE, ADRIEN RUSAOUEN

Cahier des charges

- Jeu avec trois types de bateaux 2, 3, 4 sur une grille de 8x8. Dès qu'un bateau est touché, il coule.
- On doit pouvoir jouer avec l'ordi ou avec deux joueurs.
- Programmer la possibilité de sauvegarder une partie dans un fichier et pouvoir la reprendre.
- Utiliser des matrices pour stocker la bataille



Point de départ

```c
/* bataille_navale_10x10.c
   Version "classique" (10x10) - valeurs numériques dans les matrices
   - Grille: 10x10
   - 5 bateaux: tailles 5, 4, 3, 3, 2
   - Modes: Joueur vs Joueur, Joueur vs IA
   - Placement: manuel ou aléatoire
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#define N 10
#define MAX_SHIPS 5
// Taille maximale d'un bateau (le plus grand est 5)
#define MAX_SHIP_SIZE 5

const int SHIP_SIZES[MAX_SHIPS] = {5,4,3,3,2};

/* Cell values (stored in matrices):
   0 -> EMPTY
   1 -> SHIP (intact)
   2 -> HIT
   3 -> MISS
*/

typedef struct {
   int size;
   // La taille est ajustée à la plus grande taille possible (5)
   int cells_row[MAX_SHIP_SIZE];
   int cells_col[MAX_SHIP_SIZE];
   int placed; /* 0 ou 1 */
   int hits;   /* nombre de cases touchées */
} Ship;

typedef struct {
   int board[N][N]; /* owner board: 0=empty,1=ship,2=hit,3=miss */
   int view[N][N];  /* what this player has discovered on opponent: 0=unknown/empty,2=hit,3=miss */
   Ship ships[MAX_SHIPS];
} Player;
```

- Nous avons trouvé un code en ligne correspondant à nos attentes :
  Un code du jeu classique en C facilement modifiable.

- Une fois ce code trouvé, nous n’avons plus qu’a le comprendre et modifier/ajouter le code afin que le programme corresponde au cahier des charges

Modélisation des données

1. La Structure Ship (Le Bateau)

- Le bateau est la plus petite entité que nous suivons. L'ancienne structure a été **simplifiée** car la règle est : **un coup = un bateau coulé**.
- **int size** : La taille du bateau (4, 3 ou 2).
- **int cells_row[4] / cells_col[4]** : Tableaux stockant les coordonnées exactes (ligne, colonne) de chaque segment du bateau sur la grille. Essentiel pour la fonction ship_at.
- **int placed** : Indique si le bateau a été positionné ou non (utilisé lors du placement initial).

```c
typedef struct {
  int size;
  // cells_row/col est ajusté à 4, la nouvelle taille maximale
  int cells_row[MAX_SHIP_SIZE];
  int cells_col[MAX_SHIP_SIZE];
  int placed; /* 0 ou 1 */
  int hits;   /* Conservé pour la simplicité du code existant, mais sera forcé */
} Ship;
```

2. La Structure Player (Le Joueur)

- La structure Player est le conteneur principal qui représente l'état complet du jeu pour chaque participant.
- **int board[8][8]** : **Grille de propriété (secrète)**. C'est le plateau du joueur où ses propres bateaux sont placés.
  - *Valeurs :* 0 (Vide) ou 1 (Bateau intact), puis mis à jour en 2 (Touché/Coulé) ou 3 (Raté).
- **int view[8][8]** : **Grille de visée (publique)**. C'est le tableau de suivi du joueur pour enregistrer ses tirs sur l'adversaire.
  - *Valeurs :* 0 (Inconnu), 2 (Touché), 3 (Raté).
- **Ship ships[MAX_SHIPS]** : Tableau des 3 bateaux (Ship) que possède le joueur.

```c
typedef struct {
  // Les matrices sont maintenant de taille N x N (8x8)
  int board[N][N];
  int view[N][N];
  // Tableau de 3 bateaux (MAX_SHIPS)
  Ship ships[MAX_SHIPS];
} Player;
```

Modification 1 : Adaptation 8x8

L'adaptation au cahier des charges a commencé par l'ajustement des constantes. Nous sommes passés de l'environnement initial 10 x 10 et 5 bateaux à l'environnement demandé. Ces changements ont eu un impact direct sur la taille des matrices (board et view) et sur les boucles de placement.

```
// 1. CONSTANTES ADAPTÉES

#define N 8                 // Nouvelle taille de la grille (8x8)
#define MAX_SHIPS 3           // Nouveau nombre de bateaux
#define MAX_SHIP_SIZE 4       // Nouvelle taille maximale du bateau (le plus grand est 4)

// Bateaux : 1 de 4, 1 de 3, 1 de 2
const int SHIP_SIZES[MAX_SHIPS] = {4,3,2};
```

Modification 2 : Règle « Coulé au premier tir"

Code initial

```
int process_shot(Player *attacker, Player *defender, int r, int c) {
  if (r<0||r>=N||c<0||c>=N) return -1; /* invalid */
  if (attacker->view[r][c]==2 || attacker->view[r][c]==3) return -2; /* already targeted */

  if (defender->board[r][c]==1) {
    /* hit */
    defender->board[r][c] = 2; /* mark as hit on defender board */
    attacker->view[r][c] = 2;  /* mark hit on attacker view */
    int idx = ship_at(defender, r, c);
    if (idx>=0) {
      defender->ships[idx].hits++;
      if (defender->ships[idx].hits == defender->ships[idx].size) {
        /* ship sunk */
        sink_ship(defender, idx);
        /* also mark all cells on attacker's int __unnamed_struct_0401_1::size hit cell, but ensure all ship cells set to 2) */
        for (int j=0;j<defender->ships[idx].size;j++) {
          int rr = defender->ships[idx].cells_row[j];
          int cc = defender->ships[idx].cells_col[j];
          attacker->view[rr][cc] = 2;
        }
        return 2; /* sunk */
      }
      return 1; /* hit but not sunk */
    }
    return 1; /* hit (fallback) */
  } else {
    /* miss */
    if (defender->board[r][c] == 0) defender->board[r][c] = 3;
    attacker->view[r][c] = 3;
    return 0; /* miss */
  }
}
```

Pour implémenter la règle 'Dès qu'un bateau est touché, il coule', nous avons dû modifier la fonction process_shot. Le code initial vérifiait si hits == size. Nous avons supprimé cette vérification et forcé l'appel à la fonction sink_ship et le code de retour 2 dès qu'un bateau intact (1) est touché. La variable hits n'est plus utilisée pour la décision, simplifiant la logique.

Nouveau code :

```c
int process_shot(Player *attacker, Player *defender, int r, int c) {
    // 1. VÉRIFICATION DES LIMITES ET DES COORDONNÉES DÉJÀ CIBLÉES
    if (r<0||r>=N||c<0||c>=N) return -1; /* invalide */
    if (attacker->view[r][c]==2 || attacker->view[r][c]==3) return -2; /* déjà ciblé */

    // 2. VÉRIFICATION DU COUP : TOUCHÉ (VALEUR 1)
    if (defender->board[r][c]==1) {
        /* hit */
        defender->board[r][c] = 2; /* Marque la case comme touchée/coulée sur la grille du défenseur */
        attacker->view[r][c] = 2;  /* Marque le coup sur la vue de l'attaquant */
        int idx = ship_at(defender, r, c);

        if (idx>=0) {
            // RÈGLE : COULÉ AU PREMIER TIR
            defender->ships[idx].hits = defender->ships[idx].size; // Force le statut "coulé"

            /* ship sunk */
            sink_ship(defender, idx);

            /* Marque toutes les cellules du bateau coulé sur la vue de l'attaquant */
            for (int j=0;j<defender->ships[idx].size;j++) {
                int rr = defender->ships[idx].cells_row[j];
                int cc = defender->ships[idx].cells_col[j];
                attacker->view[rr][cc] = 2;
            }
            return 2; /* sunk */
        }
        return 2; /* sunk (fallback, pour garantir que tout coup sur '1' renvoie 2) */
    } else {
        // 3. VÉRIFICATION DU COUP : MANQUÉ
        /* miss */
        if (defender->board[r][c] == 0) defender->board[r][c] = 3; /* Marque comme manqué sur la grille du défenseur */
        attacker->view[r][c] = 3; /* Marque comme manqué sur la vue de l'attaquant */
        return 0; /* miss */
    }
}
```

Affichage du Jeu

L'affichage des données brutes (matrices de nombres 0, 1, 2, 3) est difficile à lire pour l'utilisateur.

Nous avons amélioré l'expérience utilisateur en convertissant ces codes numériques en symboles ASCII clairs via un bloc switch dans la fonction print_board :

- ~ pour l'eau / case inconnue (0)
- # pour le bateau intact (1)
- X pour le coup touché ou coulé (2)
- O pour le tir manqué (3)"

```
void print_board(const int mat[N][N]) {
  printf("   ");
  for (int c=0;c<N;c++) printf(" %d", c);
  printf("\n");
  for (int r=0;r<N;r++) {
    printf("%2d ", r);
    for (int c=0;c<N;c++) {
      char symbol = '?'; // Symbole par défaut si erreur ou inconnu

      // Logique de conversion du nombre en symbole
      switch (mat[r][c]) {
        case 0: // 0 -> EMPTY/UNKNOWN
          symbol = '~'; // eau ou Inconnu
          break;
        case 1: // 1 -> SHIP (uniquement sur la propre grille du joueur)
          symbol = '#'; // Bateau intact
          break;
        case 2: // 2 -> HIT (Touché)
          symbol = 'X'; // Case touchée
          break;
        case 3: // 3 -> MISS (Raté)
          symbol = 'O'; // Tir raté
          break;
      }
      printf(" %c", symbol); // Afficher le symbole
    }
    printf("\n");
```

Placement des Bateaux

La logique de placement est assurée par deux fonctions : can_place (vérification) et place_ship (exécution). Suite à l'adaptation à la grille 8 x 8 (N=8), la validation des coordonnées est cruciale.

- Respect des Limites : La fonction can_place vérifie que le bateau ne déborde jamais de la nouvelle grille 8 x 8 (c + size > N).
- Évitement des Chevauchements : Elle garantit que le placement se fait uniquement sur des cases vides.
- Correction d'Initialisation : La fonction init_player a été corrigée pour s'assurer que les tableaux de coordonnées du bateau sont initialisés jusqu'à la nouvelle taille maximale de 4 (le plus grand bateau).

```c
int can_place(Player *p, int r, int c, int dir, int size) {
  if (dir==0) { /* horizontal to right */
    if (c + size > N) return 0;
    for (int i=0;i<size;i++) if (p->board[r][c+i] != 0) return 0;
  } else { /* vertical down */
    if (r + size > N) return 0;
    for (int i=0;i<size;i++) if (p->board[r+i][c] != 0) return 0;
  }
  return 1;
}

void place_ship(Player *p, int idx, int r, int c, int dir) {
  int size = p->ships[idx].size;
  p->ships[idx].placed = 1;
  p->ships[idx].hits = 0;
  if (dir==0) {
    for (int i=0;i<size;i++) {
      p->board[r][c+i] = 1;
      p->ships[idx].cells_row[i] = r;
      p->ships[idx].cells_col[i] = c+i;
    }
  } else {
    for (int i=0;i<size;i++) {
      p->board[r+i][c] = 1;
      p->ships[idx].cells_row[i] = r+i;
      p->ships[idx].cells_col[i] = c;
    }
  }
}
```

Fonctionnement de l’IA

1. Stratégie de l'IA (Fonction ai_choose_shot) : L'IA implémentée est basée sur une stratégie simple de tir aléatoire. Elle choisit des coordonnées (r, c) au hasard dans la grille 8 x 8 et vérifie uniquement si la case est inconnue (view[r][c] == 0) avant de tirer. Elle ne possède pas de logique de "chasse" ou de "visée".

2. Correction cruciale de confidentialité : Pour respecter les règles du jeu, nous avons modifié la boucle de jeu (game_loop). Pendant le tour de l'IA (current == 2), nous masquons désormais sa grille secrète (p2->board) et affichons uniquement la grille du joueur humain (p1->board).

```c
void ai_choose_shot(Player *ai_view, int *out_r, int *out_c) {
  // IA simple: tir aléatoire
  int tries=0;
  while (tries<10000) {
    int r = rand()%N;
    int c = rand()%N;
    if (ai_view->view[r][c] == 0) { *out_r = r; *out_c = c; return; }
    tries++;
  }
  /* fallback linear search */
  for (int r=0;r<N;r++) for (int c=0;c<N;c++) if (ai_view->view[r][c]==0) { *out_r=r; *out_c=c; return; }
  *out_r = 0; *out_c = 0;
}
```

```c
// CORRECTION DE L'AFFICHAGE EN MODE PVCPU
if (mode == 2 && current == 2) {
    printf("Grille de vos bateaux (pendant le tour de l'IA):\n");
    print_board(p1->board);
    printf("\nAttente du tir de l'ordinateur...\n");
} else {
    printf("Votre propre grille (symboles):\n");
    print_board(cur->board);
    printf("\nCe que vous avez marque sur l'adversaire (view):\n");
    print_board(cur->view);
}
```

Sauvegarde

Principe : Pour garantir une reprise de partie parfaite, nous avons implémenté une sauvegarde binaire. Cette méthode consiste à lire et écrire les structures C directement sur le disque.

Avantages : C'est la méthode la plus efficace pour enregistrer l'état exact de la mémoire : la position des bateaux, les tirs manqués et touchés, et le mode de jeu sont tous préservés sans perte.

Fonction save_game : Nous utilisons la fonction standard fwrite pour sérialiser les structures Player (p1 et p2) ainsi que le mode de jeu (mode) dans un fichier unique, nommé savegame.dat.

```c
// Sauvegarde : Écrit les structures Player p1 et p2 + le mode de jeu dans un fichier binaire
void save_game(Player *p1, Player *p2, int mode) {
    FILE *f = fopen("savegame.dat", "wb");
    if (f == NULL) {
        perror("Erreur: Impossible d'ouvrir le fichier de sauvegarde");
        return;
    }
    // Stocker le mode de jeu en premier
    fwrite(&mode, sizeof(int), 1, f);
    // Sauvegarder les structures Player en un seul bloc binaire
    fwrite(p1, sizeof(Player), 1, f);
    fwrite(p2, sizeof(Player), 1, f);
    fclose(f);
    printf("Partie sauvegardee avec succes dans 'savegame.dat'.\n");
}
```

Intégration : Sauvegarder et Quitter

Fonctionnalité : L'option de sauvegarde a été intégrée directement dans la boucle de jeu (game_loop) comme une action possible du joueur. Cela permet au joueur d'enregistrer l'état exact de la partie à tout moment.

```
printf("Options: 1) Tirer  2) Quitter  3) Sauvegarder et Quitter\nChoix: "); // AJOUT OPTION 3
int opt = read_int();
flush_stdin();

if (opt==2) { printf("Quitter...\n"); break; }
else if (opt==3) { // Sauvegarder et Quitter
    save_game(p1, p2, mode); // Sauvegarde
    break;
}
else if (opt==1) {
```

Mécanisme : Lorsque le joueur choisit l'option "3) Sauvegarder et Quitter", la fonction save_game est immédiatement appelée. Elle sérialise les structures de données (les deux joueurs) et met fin à la boucle de jeu (break).

```
Options: 1) Tirer  2) Quitter  3) Sauvegarder et Quitter
Choix: 3

Partie sauvegardee avec succes dans 'savegame.dat'.
Appuyez sur Entree pour revenir au menu...
```

Validation : L'affichage du message de confirmation valide que l'opération d'écriture sur le fichier savegame.dat a réussi.

Chargement de la partie

- **Principe de Désérialisation :** La fonction load_game réalise l'opération inverse de la sauvegarde. Elle utilise la fonction standard **fread** pour lire le fichier binaire savegame.dat et **reconstruire** les structures de données dans la mémoire.
- **Restauration de l'État :** Nous lisons les données dans le même ordre exact où elles ont été écrites : d'abord le mode de jeu, puis les structures complètes des joueurs p1 et p2.
- **Sécurité :** Une vérification est faite pour s'assurer que l'opération fread a réussi. Si oui, le jeu est relancé dans le mode de jeu qui avait été sauvegardé (PvP ou PvCPU).

```c
// Chargement : Lit les structures Player + le mode depuis le fichier binaire
int load_game(Player *p1, Player *p2, int *loaded_mode) {
    FILE *f = fopen("savegame.dat", "rb");
    if (f == NULL) {
        printf("Aucune partie sauvegardee trouvee.\n");
        return 0; // Échec du chargement
    }

    // Lire le mode de jeu
    fread(loaded_mode, sizeof(int), 1, f);

    size_t read_p1 = fread(p1, sizeof(Player), 1, f);
    size_t read_p2 = fread(p2, sizeof(Player), 1, f);
    fclose(f);

    // Vérifier si la lecture a réussi
    if (read_p1 != 1 || read_p2 != 1) {
        printf("Erreur lors de la lecture du fichier de sauvegarde (fichier corrompu).\n");
        return 0;
    }

    printf("Partie chargee avec succes.\n");
    return 1; // Succès du chargement
}
```

Démonstration de reprise

But : Montrer que la sérialisation binaire a correctement préservé l'état complet du jeu, permettant de reprendre une partie exactement là où elle a été interrompue.

Preuve de la Reprise :

- Le **Mode de Jeu** est restauré (PvP ou PvCPU).
- Les **deux grilles** du joueur (board et view) montrent l'état exact (tirs ratés, bateaux touchés, bateaux intacts).
- Le **tour de jeu** reprend avec le joueur qui devait tirer.

Cette fonctionnalité valide l'utilisation efficace de la gestion de fichiers bas niveau (fread / fwrite) sur les structures C complexes.

```
=== Bataille Navale (8x8)
1) Nouvelle partie PvP
2) Nouvelle partie PvCPU
3) Charger partie
4) Quitter
Choix:
```

En sélectionnant l'option 3, nous retrouvons bien la partie que nous avions sauvegardé

```
Tour du joueur 1
Votre propre grille (symboles):
   0 1 2 3 4 5 6 7
 0  ~ O ~ O ~ ~ ~ ~
 1  X X X O ~ ~ ~ ~
 2  ~ ~ ~ ~ ~ ~ ~ ~
 3  ~ ~ ~ ~ ~ ~ ~ ~
 4  ~ ~ ~ # ~ ~ ~ ~
 5  ~ # ~ # ~ ~ ~ ~
 6  ~ # ~ # ~ ~ ~ ~
 7  ~ ~ ~ # ~ ~ O ~

Ce que vous avez marque sur l'adversaire (view):
   0 1 2 3 4 5 6 7
 0  ~ ~ ~ ~ ~ ~ ~ ~
 1  ~ X ~ ~ ~ ~ ~ ~
 2  ~ X ~ ~ ~ ~ ~ ~
 3  ~ X ~ O ~ ~ ~ ~
 4  O X ~ ~ ~ ~ ~ ~
 5  ~ ~ ~ ~ ~ ~ ~ ~
 6  ~ O ~ ~ ~ O ~ ~
 7  ~ ~ ~ ~ ~ ~ ~ ~
Options: 1) Tirer  2) Quitter  3) Sauvegarder et Quitter
Choix:
```

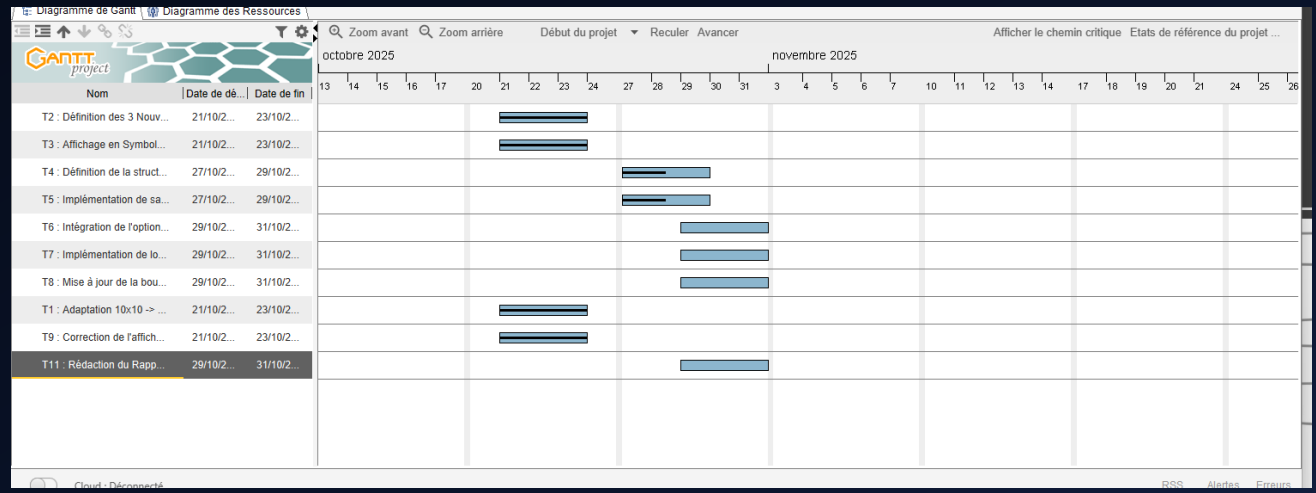
Organisation et planification du projet

Méthode : Nous avons utilisé une approche de gestion de projet structurée pour suivre l'avancement, définir les priorités et respecter les délais.

Planification des Tâches : Le projet a été divisé en phases claires :

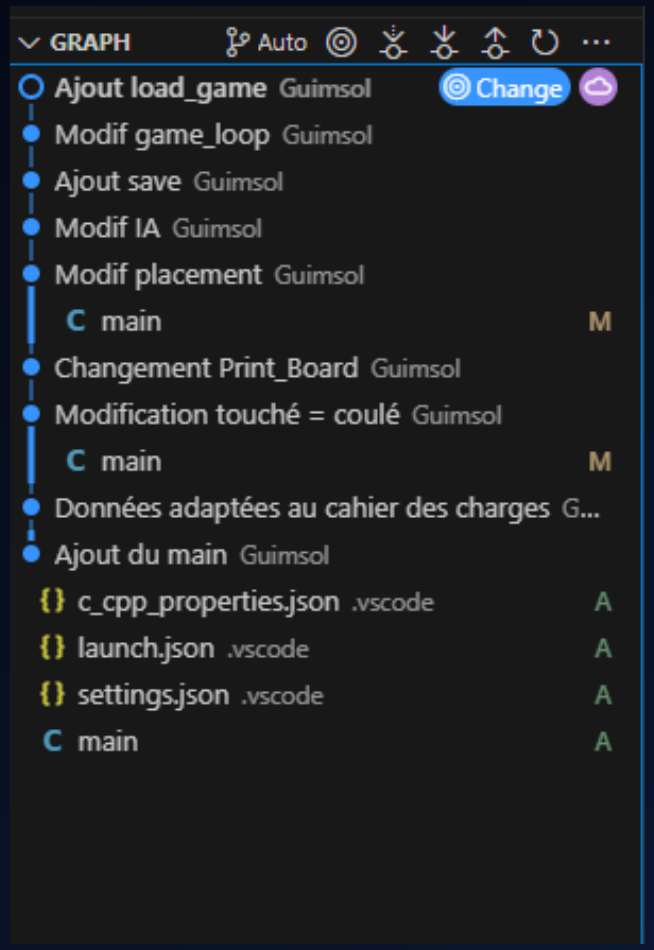
- **Phase d'Initialisation :** Adaptation de la structure (8x8) et de la règle du jeu.
- **Phase de Développement :** Implémentation de la fonctionnalité majeure de sauvegarde/reprise.
- **Phase de Finalisation :** Tests, documentation et préparation du rapport/PowerPoint.

L'utilisation d'outils de planification (comme Trello ou un diagramme de Gantt) a permis d'assurer une couverture complète des exigences du cahier des charges.



Versionning

- **Objectif :** Utiliser un système de contrôle de version pour suivre l'historique de toutes les modifications, collaborer efficacement et assurer la stabilité du code.
- **Stratégie de Branches (Git Flow Simplifié) :**
- **main :** Branche stable, réservée au code final et fonctionnel.
- **Branches de Fonctionnalités :** Nous avons créé des branches isolées (ex: feat/sauvegarde-partie, refactor/grille-8x8) pour développer et tester les modifications sans perturber la branche principale.
- **Processus :** Les modifications sont enregistrées via des **commits**, puis envoyées sur GitHub (Push), et finalement intégrées via des **Pull Requests (PR)** pour garantir la qualité du code avant la fusio



Bilan des compétences

- Ce projet a permis de renforcer la maîtrise de plusieurs concepts fondamentaux et avancés du langage C :
- **Gestion des Pointeur de Structures :** Utilisation efficace des pointeurs (Player *p1, Player *attacker) pour passer des structures complexes aux fonctions et les modifier.
- **Manipulation des Matrice (Tableaux 2D) :** Utilisation des matrices board[N][N] et view[N][N] pour modéliser le plateau de jeu et gérer l'état des cases.
- **Gestion des Flux d'Entrée/Sortie (I/O) :** Diagnostic et résolution du bug du *buffer* d'entrée avec la fonction flush_stdin, garantissant la robustesse de l'interface console.
- **Sérialisation Binaire :** Maîtrise des fonctions bas niveau (fwrite et fread) pour effectuer une **sauvegarde et une reprise de partie parfaite** en lisant et écrivant des structures C complètes en mémoire.
- **Logique Conditionnelle Complexe :** Adaptation de la fonction process_shot pour intégrer la règle spécifique "Coulé au premier tir" tout en conservant les vérifications d'erreurs (-1, -2).

Conclusion et amélioration

Bilan du Projet :

Nous avons réussi à adapter le code source aux exigences techniques et fonctionnelles :

1.Format Finalisé : Grille 8 x8 et règle "Coulé au premier tir" implémentées avec succès.

2.Objectif Majeur Atteint : La fonctionnalité de **sauvegarde et de reprise de partie** est entièrement fonctionnelle grâce à la sérialisation binaire.

3.Code Robuste : Résolution du bug critique du *buffer* d'entrée pour garantir une interface utilisateur stable.

Perspectives d'Évolution :

Pour améliorer l'expérience et la complexité du jeu, les prochaines étapes seraient :

- **Intelligence Artificielle Avancée :** Développer une logique de "chasse" (cibler autour d'un coup réussi) au lieu du tir purement aléatoire.

- **Améliorations de l'Interface :** Intégrer des couleurs en console (via des bibliothèques comme ncurses ou des codes ANSI) pour mieux différencier l'eau, les ratés, et les coups.

- **Mode 1 Joueur (Campagne) :** Sauvegarder les statistiques du joueur contre l’IA.

Démonstration du jeu

```
=== Bataille Navale (8x8) - Projet Final
1) Nouvelle partie PvP
2) Nouvelle partie PvCPU
3) Charger partie
4) Quitter
Choix: 2
```

```
=== Bataille Navale (8x8) - Projet Final ===
1) Nouvelle partie PvP
2) Nouvelle partie PvCPU
3) Charger partie
4) Quitter
Choix: 2

Vous: placement manuel(1) ou aleatoire(2)? 1
```

```
Placement manuel des bateaux (grille 0..7). Entrez ligne puis colonne.
Grille actuelle (symboles):
    0 1 2 3 4 5 6 7
 0  ~ ~ ~ ~ ~ ~ ~ ~
 1  ~ ~ ~ ~ ~ ~ ~ ~
 2  ~ ~ ~ ~ ~ ~ ~ ~
 3  ~ ~ ~ ~ ~ ~ ~ ~
 4  ~ ~ ~ ~ ~ ~ ~ ~
 5  ~ ~ ~ ~ ~ ~ ~ ~
 6  ~ ~ ~ ~ ~ ~ ~ ~
 7  ~ ~ ~ ~ ~ ~ ~ ~
Placer bateau taille 4
Entrez ligne (0-7): 2
Entrez colonne (0-7): 4
Direction 0=horizontal, 1=vertical: 0
```

Démonstration du jeu

```
Grille actuelle (symboles):
    0 1 2 3 4 5 6 7
 0  ~ ~ ~ ~ ~ ~ ~ ~
 1  ~ ~ ~ ~ ~ ~ ~ ~
 2  ~ ~ ~ ~ # # # #
 3  ~ ~ ~ ~ ~ ~ ~ ~
 4  ~ ~ ~ ~ ~ ~ ~ ~
 5  ~ ~ ~ ~ ~ ~ ~ ~
 6  ~ ~ ~ ~ ~ ~ ~ ~
 7  ~ ~ ~ ~ ~ ~ ~ ~
Placer bateau taille 3
Entrez ligne (0-7): 1
Entrez colonne (0-7): 1
Direction 0=horizontal, 1=vertical: 1
```

```
Tour du joueur 1
Votre propre grille (symboles):
    0 1 2 3 4 5 6 7
 0  ~ ~ ~ ~ ~ ~ ~ ~
 1  ~ ~ ~ ~ ~ ~ ~ ~
 2  ~ ~ ~ ~ ~ ~ # ~
 3  ~ ~ ~ ~ ~ ~ # ~
 4  ~ ~ ~ ~ ~ ~ # ~
 5  ~ ~ ~ ~ # ~ # ~
 6  ~ ~ ~ ~ # ~ ~ ~
 7  ~ ~ ~ ~ # ~ # #

Ce que vous avez marque sur l'adversaire (view):
    0 1 2 3 4 5 6 7
 0  ~ ~ ~ ~ ~ ~ ~ ~
 1  ~ ~ ~ ~ ~ ~ ~ ~
 2  ~ ~ ~ ~ ~ ~ ~ ~
 3  ~ ~ ~ ~ ~ ~ ~ ~
 4  ~ ~ ~ ~ ~ ~ ~ ~
 5  ~ ~ ~ ~ ~ ~ ~ ~
 6  ~ ~ ~ ~ ~ ~ ~ ~
 7  ~ ~ ~ ~ ~ ~ ~ ~
Options: 1) Tirer  2) Quitter  3) Sauvegarder et Quitter
Choix: 1

Entrez ligne cible (0-7): 3

Entrez colonne cible (0-7): 3

Vous avez coule un bateau!
Appuyez sur Entree pour continuer...
```

Démonstration du jeu

```
Tour du joueur 2
Grille de vos bateaux (pendant le tour de l'IA):
    0 1 2 3 4 5 6 7
 0  ~ ~ ~ ~ ~ ~ ~ ~
 1  ~ ~ ~ ~ ~ ~ ~ ~
 2  ~ ~ ~ ~ ~ ~ # ~
 3  ~ ~ ~ ~ ~ ~ # ~
 4  ~ ~ ~ ~ ~ ~ # ~
 5  ~ ~ ~ ~ # ~ # ~
 6  ~ ~ ~ ~ # ~ ~ ~
 7  ~ ~ ~ ~ # ~ # #

Attente du tir de l'ordinateur...
Ordinateur tire en 6,6
Ordinateur a rate.
Appuyez sur Entree pour continuer...
```

```
Tour du joueur 1
Votre propre grille (symboles):
    0 1 2 3 4 5 6 7
 0  ~ ~ ~ ~ ~ ~ ~ ~
 1  ~ ~ ~ ~ ~ ~ ~ ~
 2  ~ ~ ~ ~ ~ ~ # ~
 3  ~ ~ ~ ~ ~ ~ # ~
 4  ~ ~ ~ ~ ~ ~ # ~
 5  ~ ~ ~ ~ # ~ # ~
 6  ~ ~ ~ ~ # ~ O ~
 7  ~ ~ ~ ~ # ~ # #

Ce que vous avez marque sur l'adversaire (view):
    0 1 2 3 4 5 6 7
 0  ~ ~ ~ ~ ~ ~ ~ ~
 1  ~ ~ ~ ~ ~ ~ ~ ~
 2  ~ ~ ~ ~ ~ ~ ~ ~
 3  ~ X X X ~ ~ ~ ~
 4  ~ ~ ~ ~ ~ ~ ~ ~
 5  ~ ~ ~ ~ ~ ~ ~ ~
 6  ~ ~ ~ ~ ~ ~ ~ ~
 7  ~ ~ ~ ~ ~ ~ ~ ~
Options: 1) Tirer  2) Quitter  3) Sauvegarder et Quitter
Choix: 1

Entrez ligne cible (0-7): 5

Entrez colonne cible (0-7): 5

A l'eau (rate).
Appuyez sur Entree pour continuer...
```

Démonstration du jeu

```
Tour du joueur 1
Votre propre grille (symboles):
    0 1 2 3 4 5 6 7
 0  ~ O ~ O ~ O ~ ~
 1  ~ ~ ~ ~ O ~ ~ ~
 2  ~ ~ ~ O ~ ~ # ~
 3  ~ ~ ~ ~ ~ O # ~
 4  ~ ~ O O ~ ~ # ~
 5  ~ ~ O O X ~ # O
 6  ~ ~ ~ ~ X ~ O ~
 7  ~ ~ ~ ~ X ~ X X

Ce que vous avez marque sur l'adversaire (view):
    0 1 2 3 4 5 6 7
 0  ~ ~ O ~ ~ ~ ~ ~
 1  X O ~ ~ O ~ O ~
 2  X ~ ~ ~ ~ ~ ~ O
 3  X X X X ~ O ~ ~
 4  X ~ ~ ~ ~ ~ ~ O
 5  ~ ~ O ~ ~ O ~ ~
 6  O ~ ~ ~ ~ ~ O ~
 7  ~ ~ ~ O ~ ~ ~ ~
Options: 1) Tirer  2) Quitter  3) Sauvegarder et Quitter
Choix: 1

Entrez ligne cible (0-7): 7

Entrez colonne cible (0-7): 1

A l'eau (rate).
Appuyez sur Entree pour continuer...
```

```
Tour du joueur 1
Votre propre grille (symboles):
    0 1 2 3 4 5 6 7
 0  O O ~ O ~ O ~ ~
 1  O ~ ~ ~ O ~ ~ ~
 2  ~ ~ ~ O ~ O # ~
 3  ~ ~ ~ ~ ~ ~ O # ~
 4  ~ ~ O O ~ ~ # ~
 5  ~ ~ O O X ~ # O
 6  ~ ~ ~ ~ X ~ O ~
 7  ~ ~ ~ ~ X ~ X X

Ce que vous avez marque sur l'adversaire (view):
    0 1 2 3 4 5 6 7
 0  ~ ~ O ~ ~ ~ ~ ~
 1  X O ~ ~ O ~ O ~
 2  X ~ ~ ~ ~ ~ ~ O
 3  X X X X ~ O ~ ~
 4  X ~ ~ O ~ ~ ~ O
 5  ~ ~ O ~ ~ O ~ ~
 6  O ~ ~ ~ O ~ O ~
 7  ~ O ~ O ~ ~ ~ ~
Options: 1) Tirer  2) Quitter  3) Sauvegarder et Quitter
Choix: 1

Entrez ligne cible (0-7): 3

Entrez colonne cible (0-7): 3

Case deja ciblee. Reessayer.
Entrez ligne cible (0-7):
```

Démonstration du jeu

```
Tour du joueur 1
Votre propre grille (symboles):
    0 1 2 3 4 5 6 7
 0  O O ~ O ~ O ~ ~
 1  O ~ ~ ~ O ~ ~ ~
 2  ~ ~ ~ O ~ O # ~
 3  ~ ~ O O O O # ~
 4  ~ O O O ~ ~ # ~
 5  ~ ~ O O X ~ # O
 6  ~ ~ ~ O X ~ O ~
 7  ~ ~ ~ ~ X ~ X X

Ce que vous avez marque sur l'adversaire (view):
    0 1 2 3 4 5 6 7
 0  ~ ~ O ~ ~ O ~ ~
 1  X O ~ ~ O ~ O ~
 2  X ~ ~ O ~ ~ ~ O
 3  X X X X ~ O ~ ~
 4  X ~ ~ O ~ ~ O O
 5  ~ ~ O ~ ~ O ~ ~
 6  O ~ O ~ O ~ O ~
 7  ~ O ~ O ~ ~ O ~
Options: 1) Tirer  2) Quitter  3) Sauvegarder et Quitter
Choix: 1

Entrez ligne cible (0-7): 2

Entrez colonne cible (0-7): 5

Vous avez coule un bateau!

Joueur 1 gagne!
Partie terminee. Appuyez sur Entree pour revenir au menu...
```