ROB3 – Mini Projet

INF Bataille Navale
Compte Rendu

SADOUN Yanis, SKARLEAS Vasileios Filippos

Bataille Navale S5 2023-2024

Instructions

Pour lancer le jeu de la version 2 (version la plus avancée et développée ci-dessous), il suffit de donner accès au programme d'installation automatique. Voici les instructions :

1. Ouvrez une fenêtre du terminal et assurez-vous que vous êtes dans le même répertoire que le code source du projet

```
vasilisskarleas@Vasiliss-MacBook-Pro MiniProjet % 
```

2. Tappez `chmod 777 play.sh` et exécutez ensuite en faisant `bash play.sh`. Maintenant, il suffit de suivre les instructions sur le terminal et vous pouvez jouer.

```
vasilisskarleas@Vasiliss-MacBook-Pro MiniProjet % chmod 777 play.sh
vasilisskarleas@Vasiliss-MacBook-Pro MiniProjet % bash play.sh
Welcome to the Battle Ship game from Vasileios Filippos Skarleas and Yanis Sadoun. Once you are ready to start playing simply press ENTER
Bienvenue dans le jeu Battle Ship de Vasileios Filippos Skarleas et Yanis Sadoun. Une fois que vous êtes prêt à commencer à jouer, appuyez simplement sur ENTER
```

3. Si vous êtes sur un Mac, le jeu est développé de telle manière qu'il permet d'avoir de la musique lorsque vous jouez.
4. Amusez-vous bien! Bonne lecture...

Introduction

Dans le cadre de notre cours d'informatique en troisième année de Robotique à Polytech Sorbonne, notre défi a été de développer un projet collaboratif : une version informatisée du célèbre jeu de bataille navale.Ce dernier a non seulement mis à l'épreuve nos compétences en programmation, mais a aussi exigé une coordination d'équipe efficace, avec une gestion complexe des fichiers et de leurs interrelations. Notre objectif principal étant de creer un espace où le joueur arrangeait six navires de différentes tailles sur un

plateau dont les dimensions étaient déterminées par lui-même, ajoutant ainsi une touche personnelle et stratégique au jeu.

Ce projet nous a également permis d'explorer et d'intégrer des fonctionnalités avancées , telles que :

- Un mode multijoueur,
- La possibilité de sauvegarder et de continuer des parties en cours,
- Et l'introduction d'une intelligence artificielle pour enrichir l'expérience de jeu.

Pour structurer ce rapport, nous commencerons par expliquer les règles du jeu et les differentes fonctions de bases imposées par le sujet . Nous décrirons ensuite les principales fonctions que nous avons programmées et les contributions personnelles apportées ; pour poursuivre avec les problèmes rencontrés et les solutions trouvées. Ce plan permettra de présenter de manière claire et détaillée notre processus de travail et les résultats obtenus.

Règles du jeu

Avant de commencer la programmation de notre version du jeu de bataille navale, il était essentiel de définir clairement les règles, notamment les critères de victoire, que ce soit en solo ou contre un autre joueur. Inspirés par la version simplifiée que nous avions réalisée lors du TP2, nous avons décidé que le critère de victoire serait basé sur le nombre total de tentatives du joueur, en tenant compte de la taille variable du plateau de jeu.
L'alogithme ci-desosus rend compte des cette regle du jeu :

```
// 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
}
```

Deplus, pour améliorer l'expérience de jeu et ajouter un élément de défi, nous nous sommes inspirés des jeux d'arcade classiques en introduisant une limite de temps. Cette fonctionnalité a été mise en œuvre grâce à l'exploitation de la bibliothèque 'time.h'. L'idée était de créer un système qui arrête le jeu lorsque le temps imparti est écoulé, ajoutant ainsi une pression temporelle qui pousse le joueur à prendre des décisions plus rapidement.

En ce qui concerne le mode pause, nous avons développé un algorithme spécifique qui gère cette situation. Lorsque le jeu est mis en pause, le compteur de temps est suspendu, permettant au joueur de reprendre la partie sans pénalité temporelle.L'algorithme suivant détaille comment ces différents éléments sont intégrés et gérés au sein du jeu :

```
// 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
}
```

Ce système ajoute une couche de complexité et d'engagement, rendant le jeu plus captivant et compétitif.

Indications

Dans le développement de notre jeu de bataille navale, plusieurs fonctions de base étaient nécessaires, chacune jouant un rôle crucial dans la dynamique du jeu :

- `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 utilisé 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 statut du point sur le plateau pour les coordonées en question et le met à jour.

- Si `prop[x][y] = NAVIRE`: Alors `prop[x][y] = NAVIRE_TROUVE` pour indiquer un navire touché.
- Si `prop[x][y] = NAVIRE`: Alors `prop[x][y] = NAVIRE_TROUVE_PLUS_1` pour indiquer un navire précédemment touché.
- Sinon (pas de navire): Alors `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;

    // Vérifie si la case ne contient pas de navire
    if (prop[x][y] == AUCUN_NAVIRE)
    {
        navire_coule = true;
    }
    // Vérifie si la case contient un navire qui n'a pas encore été touché
    else if (prop[x][y] == NAVIRE_TROUVE)
    {
        navire_coule = true;
        nombre_touches = nombre_touches + 1; // Incrémente le nombre de
navires touchés
    }
    else
    {
        navire_coule = false; // La case contient un navire déjà touché
    }
```

```
    return navire_coule; // Renvoie l'état du navire (coulé ou non)
}
```

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 verifiée, aucun point des navires à été trouvé et `navire_trouve` est mis à `faux`.

Fonction `ajuster_tours`

La fonction ajuster_tours est une composante importante de notre jeu de bataille navale. Elle adapte dynamiquement le nombre de tours disponibles en fonction de la taille du plateau de jeu et du nombre de navires choisis par l'utilisateur. Cette flexibilité permet d'équilibrer la difficulté du jeu et offre une expérience personnalisée.

L'idée derrière cette fonction est de rendre le jeu juste et équitable, quelle que soit la configuration choisie. Par exemple, sur un petit plateau de 4x4 avec un nombre fixe de tours, disons 55, le jeu pourrait devenir trop facile ou trop difficile selon le nombre de navires présents. Cela pourrait nuire à l'expérience de jeu.

L'algorithme de ajuster_tours calcule le nombre maximum de tours en tenant compte de ces variables. Notre approche etait simple. Il faudrait trouver une fonction mathematique qui gurantira le bon `max_tours` du jeu. Effectivement, si on prend le numero des cases disponibles pour une configuration, tout en suposant que les propabilites d'avoir des points des navires partour dans le tableau, il suffit de multiplier ce nombre avec un pourcentage pour eliminer l'option pour quel'qu'un d'etre capable de choisir tous les cases du tableau. Cette approche garantit que chaque partie reste stimulante et engageante, peu importe les choix de l'utilisateur. Son algorithme est le suivant :

1. Vérifier le mode de jeu (`mode`) est solo avec l'option "rounds" ou multiplayer:

- Si `mode` est égal à 1 (mode solo [load fait partie de la même catégorie du mode de jeu]), ajuster le nombre maximum de tours en fonction de: `taille_plateau * taille_plateau * 0.8`.
- Si `mode` est égal à 2 (mode mutliplayer), ajuster le nombre maximum de tours en fonction de: `taille_plateau * taille_plateau * 0.9`.

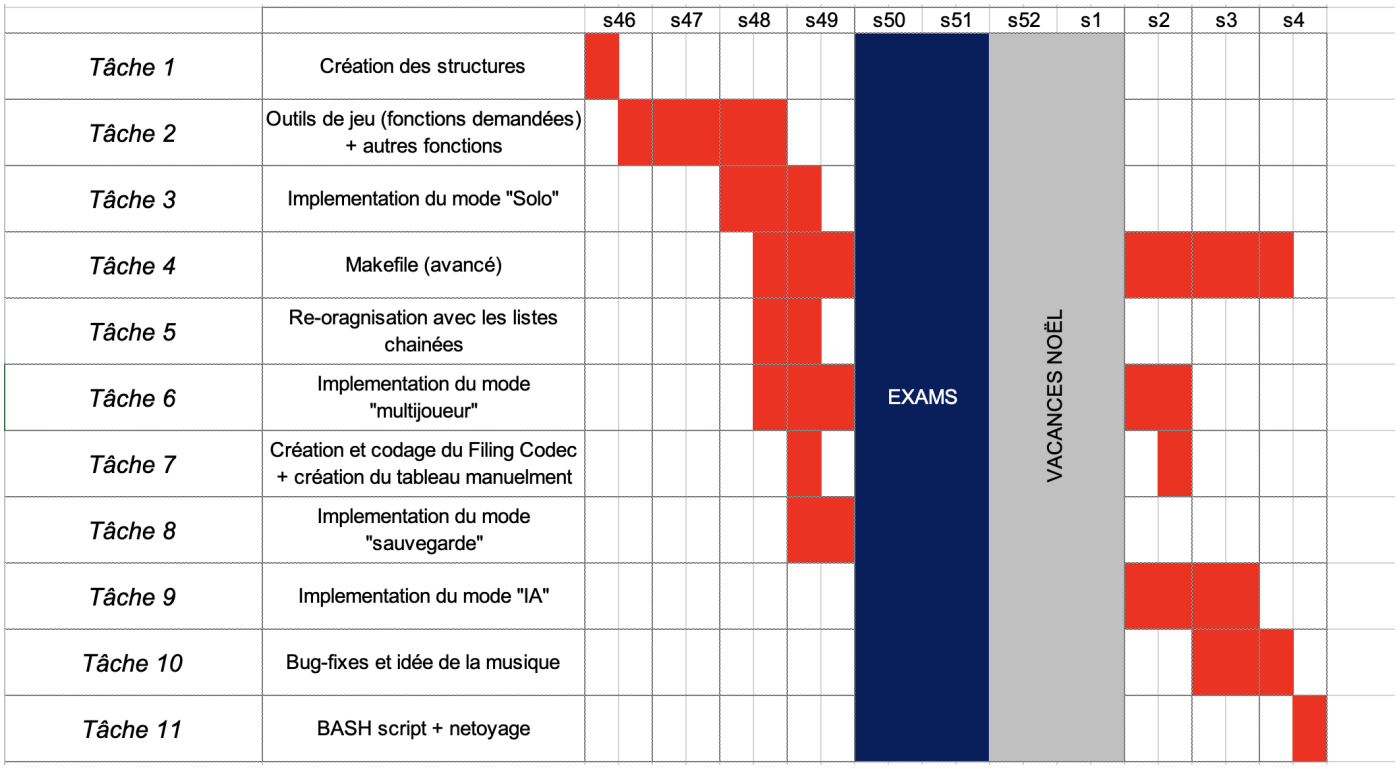
2. Sinon on fait rien (car en fait on n'a pas besoin de ce variable pour les autres mode de jeu [IA])

3. Affecter la valeur calculée de `max_tours` à la variable globale.

Colaboration et gestion de projet en Binôme

Comment garantir que les contributions de chaque membre s'alignent parfaitement dans un projet informatique complexe ? Pour notre projet, les communications directes étaient primordiales, mais pour documenter et suivre les discussions, nous avons utilisé Git. Cette approche nous a permis de contribuer indépendamment, tout en conservant un historique complet des changements. Git a également facilité l'analyse pré-compilation du code et sa sauvegarde sur un service en ligne, accessible depuis n'importe quel appareil. Enfin, notre projet était hébergé sur un répertoire GitHub, intégrant GitHub Copilot pour l'analyse de code.

Pour mener à bien notre projet, nous avons adopté une stratégie d'organisation et de progression graduelle. Nous avons commencé par des étapes simples, améliorant et développant notre jeu au fur et à mesure. Cette approche méthodique nous a permis de rester concentrés et efficaces. De plus, pour une répartition claire des tâches, nous avons utilisé un tableau de Gantt. Celui-ci a été important pour suivre notre progression, affecter des responsabilités et respecter les délais, assurant ainsi une gestion de projet structurée et cohérente tout le long.



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 modifier un message ou addresser un probleme sur la partie de l'affichage par exemple.

Voici une brève description des fonctions declarées et construites :

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

Elle permet ainsi de déterminer le message qu'il faut afficher selon les combinaisons des paramètres.

game_mode_graphics_congratulations:

Cette fonction permet d'afficher un message de félicitations au moment ou 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:

Cette fonction permet d'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 librairie 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

Dans le cadre de notre projet, les consignes exigeaient la déclaration de structures spécifiques pour représenter les éléments clés du jeu:

```
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;
```

Pour une gestion optimale des divers modes de jeu et scénarios dynamiques dans notre projet, nous avons mis en place des structures supplémentaires. Chacunes d'entres elles a été conçue pour remplir un rôle spécifique, contribuant à la flexibilité et à l'efficacité du jeu:

- 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 chainée 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 chainée des points.
- La structure `Tableau_Point` est utilisée pour transformer une liste chainée de 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;
```

```c
/*---- 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'implémentation de la fonction de sauvegarde était un aspect primordial de notre jeu. Nous avons conçu un système de codage de fichiers, ou "filing codec", pour cette fonctionnalité. La sauvegarde d'une partie se fait simplement : l'utilisateur ouvre le menu et sélectionne "Sauvegarder". La fonction `api_save_game` se charge alors d'encoder l'état actuel du jeu en utilisant notre codec spécifique.

Pour reprendre une partie sauvegardé, les joueurs sélectionnent "Continuer" ou "Load" dans le menu principal. Cette action active `api_load_game`, qui réinitialise le jeu avec les données sauvegardées. Nous avons également anticipé des situations complexes, telles qu'une tentative de reprise à partir d'une sauvegarde inexistante. Qu'advient-il dans ce cas ? Ou si un joueur sauvegarde, continue de jouer, puis ferme le jeu sans terminer ? Nous avons inclus des fonctions pour traiter ces scénarios, y compris la suppression automatique du fichier de sauvegarde une fois les données chargées, assurant ainsi une gestion de jeu fluide et cohérente.

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" est une méthode structurée pour encoder et décoder les données du jeu de bataille navale. Il suit une séquence spécifique d'éléments pour garantir l'intégrité et la lisibilité des fichiers de jeu enregistrés.

Qu'est ce qui est inclus dans 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 indique 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.

Les differents modes de jeu

Dans notre projet de jeu de bataille navale, nous avons introduit plusieurs modes de jeu pour enrichir l'expérience utilisateur. Ces modes variés offrent aux joueurs des approches diversifiées pour profiter pleinement du jeu. Voici un aperçu de ces différents modes et de leurs caractéristiques respectives :

Mode Solo

Dans notre jeu, le mode solo se décline en deux sous-modes distincts pour varier l'expérience de jeu :

- **Mode classique :** Ici, le joueur a un nombre illimité de tours pour essayer de couler les navires contrôlés par l'ordinateur. Ce mode permet au joueur de prendre son temps et de stratégiser chaque coup sans pression temporelle.
- **Mode chronométré :** Dans ce sous-mode, le joueur est confronté à une contrainte de temps, ajoutant un élément de rapidité et d'urgence au jeu. Le joueur doit couler tous les navires de l'ordinateur avant la fin du temps imparti, ce qui demande des décisions rapides et stratégiques.

Les deux variantes du mode solo sont organisées comme suit :

1. **Initialisation :** : L'ordinateur place aléatoirement ses navires sur le plateau de jeu. Le joueur ne voit pas la disposition des navires ennemis.
2. **Jeu du joueur :** Le joueur choisit une case pour tenter de trouver et couler les navires ennemis.
3. **Vérification du résultat :** Après chaque coup, le jeu vérifie si le joueur a touché, manqué, ou coulé un navire ennemi.
4. **Fin de la partie :** La partie se termine soit lorsque tous les navires ennemis sont coulés, soit lorsque le joueur atteint la limite de tours ou de temps imparti.

Mode Load

Dans notre jeu, le mode "Load" permet aux joueurs de continuer une partie sauvegardée précédemment. Le fichier de sauvegarde contient des informations essentielles (plus de détails sur la partie Filing codec)) telles que :

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

Nota bene : Pour améliorer ce mode, nous pourrions envisager:

- des sauvegardes automatiques à intervalles réguliers (par exemple toutes les 10 secondes)
- de créer un second fichier archivant chaque mouvement. Ce fichier permettrait de revoir les actions du joueur sur le terminal, offrant ainsi une rétrospective détaillée de la partie.

Mode Multiplayer

Le mode multijoueur permet à deux joueurs de s'affronter directement. Chaque joueur crée son propre plateau de jeu, en y positionnant ses navires de manière stratégique. Les joueurs se relaient ensuite pour attaquer les navires adverses sur le plateau de l'autre joueur. L'algorithme de determination du gagant prend en compte le nombre de navires coulés, ainsi que le nombre de tours joués .

Algorithme :

1. Installation des Compteurs :Initialiser un compteur de navires coulés pour chaque joueur.
2. Progression du Jeu :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. Critères de Victoire:
    - La partie prend fin de deux manières :Si un joueur parvient à couler tous les navires de son adversaire, il remporte immédiatement la partie. Si la limite de

tours est atteinte sans qu'un joueur ait coulé tous les navires adverses, le joueur avec le plus grand nombre de navires ennemis détruits est déclaré vainqueur.

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

Le mode Intelligence Artificielle (AI) dans notre jeu de bataille navale propose une expérience solo captivante, où le joueur se mesure à un ordinateur simulant des stratégies humaines. Développer une IA crédible et réactive a été un enjeu majeur de ce projet, apportant une dimension supplémentaire et enrichissante au mode solo du jeu.

Déroulement du Jeu :

Sélection du Mode AI : Le joueur a le choix entre deux sous-modes AI, "Spark" et "Fireball", chacun avec une stratégie d'attaque distincte.

Initialisation du Plateau : Le joueur crée son propre plateau en plaçant ses navires de manière personnalisée.Parallèlement, l'AI place ses navires de manière automatique sur son plateau.

En mode "Spark", l'AI choisit aléatoirement ses cibles sur le plateau du joueur. Elle attaque sans suivre de stratégie particulière, ce qui rend ce mode plus adapté aux débutants ou pour une expérience de jeu moins complexe. En mode "Fireball", l'AI

adopte une approche plus stratégique. Après avoir touché un navire, elle suit un algorithme pour déterminer la meilleure façon de couler ce navire en attaquant les cases adjacentes de manière systématique.

Interaction Joueur-AI :Le joueur et l'AI se relaient pour attaquer les navires adverses. Le jeu fournit des retours après chaque attaque, indiquant si un tir a touché, manqué, ou coulé un navire.

Fin de la Partie : La partie se termine lorsque l'un des joueurs (le joueur humain ou l'AI) a réussi à couler tous les navires de l'adversaire.Le gagnant est celui qui parvient le premier à éliminer tous les navires de son adversaire.

Voici l'algorithme de la fonction next_point de notre programme, fonction qui joue un role important dans le mode AI,particulierement dans la stratégie plus avancée du mode "Fireball":

Algorithme

```
  // Exemple d'algorithme IA

// Grille de jeu
//   A B C D
// 1  0 1 0 0
// 2  0 1 0 0
// 3  0 1 0 0
// 4  0 0 0 0

// États :
#define VIDE 0
#define NAVIRE 1                // Navire placé sur la position
#define AUCUN_NAVIRE -1         // Déjà joué et aucun point de navire trouvé
#define NAVIRE_TROUVE 2         // Un point du navire a été trouvé
#define NAVIRE_TROUVE_PLUS_1 3 // Un point du navire déjà trouvé est
retrouvé
#define CUSTOM_NAVIRE 8
#define COULE 10                // Félicitations, le navire entier a été
trouvé

// Cas :
// N°1
// 1,A = 1,1 => 0,0 (c'est aussi l'étape d'initialisation – première fois
que l'IA joue)
//    A  B  C  D
// 1 -1  1  0  0
// 2  0  1  0  0
// 3  0  1  0  0
// 4  0  0  0  0
```

```
// Action -> no_skip_action = true
// ===================
// N°2
// 2,B = 2,2 => 1,1
//    A  B  C  D
// 1 -1  1  0  0
// 2  0  2  0  0
// 3  0  1  0  0
// 4  0  0  0  0

// Action -> no_skip_action = false => appelle l'algorithme next_point
// ===================
// N°3
// x_prev = 1
// y_prev = 1

// Action -> Recherche du point courant pour ce tour
// 1. Accéder à un switch qui a les modes suivants (sens_mode) :
//    1. Par défaut
//    2. Horizontal
//        1. Gauche
//        2. Droite
//        3. Par défaut
//    3. Vertical
//        1. Haut
//        2. Bas
//        3. Par défaut
// 2. Dans l'état par défaut :
//    1. Vérifier si un des points autour de lui (x_prev, y_prev) est déjà
trouvé, donc dans l'état NAVIRE_TROUVE défini comme 2.
//    2. Si ce n'est PAS le cas, sélectionner aléatoirement un des
voisinages (Haut, Bas, Gauche, Droite) et les définir comme x_now, y_now,
x_backup, y_backup. Une fois next_point joué (signifie que la fonction
next_point est terminée et nous avons appelé la fonction random_point avec
x_now, y_now dans le programme principal) et selon la réponse de la fonction
random_point, faire ce qui suit :
//        1. no_skip_action = TRUE && state == 1
//           Signifie que le point joué (x_now, y_now) n'était pas une
partie du navire pour lequel nous connaissions déjà un point. Cela signifie
qu'un point est peut-être dans l'autre direction.
//           Définir no_skip_action = FALSE et avec sens_mode = Horizontal
si previous_sens était Vertical (et vice versa).
//        2. no_skip_action = FALSE
//           Signifie que le point joué (x_now, y_now) faisait partie du
navire. Selon la direction suivie, nous devons suivre la même.
//           Attendre la prochaine itération - CETTE PARTIE PEUT ÊTRE
AMÉLIORÉE POUR DEMANDER À L'ALGORITHME DE COMMENCER À CHERCHER DANS la même
direction, sens inverse.
//        3. Si c'est le cas, comprendre la direction du navire et le sens
général et effectuer -> Pour x_now et y_now dans ce cas, nous définirons le
point suivant dans le même sens (horizontal ou vertical) et sens inversé
profond !! Une fois next point joué (signifie que la fonction next_point est
terminée et nous avons appelé la fonction random_point avec x_now, y_now
dans le programme principal) et selon la réponse de la fonction
random_point, faire ce qui suit.
//        4. no_skip_action = TRUE
//           Signifie que le point joué (x_now, y_now) faisait partie du
```

```
navire. Selon la direction suivie, nous devons suivre la même.
//           Attendre la prochaine itération
//        5. no_skip_action = FALSE
//           Signifie que le point joué (x_now, y_now) faisait partie du
navire. Selon la direction suivie, nous devons suivre la même.
//           Attendre la prochaine itération
// 3. Dans l'état Horizontal, accéder au switch pour deep_sens et effectuer
:
//    1. Définir x_now et y_now selon leur deep_sens et l'état Horizontal
// 4. Dans l'état Vertical, accéder au switch pour deep_sens et effectuer :
//    1. Définir x_now et y_now selon leur deep_sens et l'état Vertical
```

Ici, La fonction next_point utilise une combinaison d'heuristique et d'aléatoire pour déterminer le prochain point à vérifier dans le jeu de bataille navale. La fonction essaie d'abord de trouver de nouveaux segments de navire en vérifiant les points autour du point précédent qui sont déjà connus pour faire partie d'un navire. Si un nouveau segment de navire est trouvé, la fonction passe à l'autre direction de recherche pour continuer à chercher le navire. Si aucun nouveau segment de navire n'est trouvé, la fonction sélectionne au hasard un point dans une direction qui n'a pas encore été explorée. La fonctin intègre également une logique pour gérer les cas où elle trouve un segment de navire mais ne peut pas continuer dans la même direction (à cause des bords du plateau ou parce qu'elle atteint un point déjà exploré sans succès). Dans ces situations, elle inverse son orientation (de verticale à horizontale ou vice versa) pour continuer à chercher le reste du navire.

Nota bene

Par ailleurs, nous avons intégré d'autres fonctionnalités à notre jeu , parfois avec une touche d'humour, pour rendre l'expérience encore plus agréable et surprenante. Ces éléments sont à découvrir par les utilisateurs lorsqu'ils explorent le jeu...

Problèmes - Solutions

Durant le développement de notre jeu de bataille navale, nous avons fait face à divers défis techniques, nécessitant réflexion et créativité pour trouver des solutions adéquates. Voici un résumé des principaux obstacles rencontrés et des stratégies adoptées pour les résoudre :

1. **Problème de Création de Navires :** Sur un petit plateau, la création de six navires posait problème. Le souci provenait de la non-réinitialisation d'une variable

booléenne, qui entravait l'incrémentation correcte du compteur de navires. Nous avons résolu ce problème en réinitialisant cette variable à chaque tentative de placement de navire.

2. **Détection de Navires en Modes Multijoueur et IA :** Nous avions des difficultés à détecter les navires placés manuellement. Le problème venait de l'utilisation d'un code d'identification différent pour ces navires. La solution a consisté à inclure le "ou" logique dans la fonction `est_valide_pro` afin de détecter également ces points.
3. **Utilisation de Liste Chaînée:** Cette procédure nous permet de n'avoir qu'un seul plateau où tout est analysé de manière dynamique. Cette liste ressemble au tableau suivant:| 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 |
4. **Optimisation de la méthode Liste Chaînée :** Pour minimiser le temps de traitement, nous souhaitions supprimer les navires d'une liste chaînée (`Liste_Navire`). La fonction `supprimer_liste_Navire` a été ajustée pour cette opération, améliorant l'efficacité du processus.

Nota bene: Nous avons développé une version améliorée de notre fonction d'affichage, nommée printing_the_grille_v2. Cette nouvelle version est plus adaptée et a été intégrée dans la dernière mise à jour de notre code. Elle remplace l'ancienne fonction d'impression pour offrir une meilleure expérience utilisateur.

Notes Supplémentaires

Exit codes

Vous trouverez ci-dessous la définition et l'explication des codes de sortie de nos programmes :

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
exit(7)	OUTPUT	The game ended with the professor winning	This is only a touch of humour in our project :)

Versioning

Le versioning est un élément clé en programmation, assurant la cohérence des modifications et facilitant la collaboration. Il est aussi primordial pour la récupération de données en cas de perte ou corruption. Au fil du projet, nous avons créé différentes versions de notre code, chacune marquant une étape importante de son évolution. Cela nous a permis de suivre les progrès, d'intégrer de nouvelles fonctionnalités et d'effectuer des corrections de manière structurée.

Voici les differentes versions dévelopées lors de l'evolution du projet :

- **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.
- **V8.1** Fixed some logique issues on the AI model called Spark and added some fixes.

- **V8.2** optimizzed the main program and added code to Fireball AI version
- **V8.3** Fireball ai has been updated and for now it works excpet from some extreme edge cases
- **V8.4** Code compeletd. Needs to add the translatiosn and fix the ctes[pndacnces to the single programs
- **V8.5** Many translations were added on the ui.c.
- **V8.6** Translations were added on the api.c and on the jeu_v2.c
- **V9.0** Music for Mac computers has been tested

Conclusion - pistes d'améliorations

Le développement de notre jeu de bataille navale en C a été une expérience formatrice et enrichissante, nous offrant des compétences précieuses en programmation et en résolution de problèmes. Toutefois, il reste des possibilités d'amélioration pour augmenter l'attrait et la performance du jeu.

Actuellement, notre jeu fonctionne avec une interface en ligne de commande, mais pour le rendre plus interactif et esthétique, l'ajout d'une interface graphique, via des bibliothèques externes comme GTK+, Qt ou SDL, est envisageable. Cette amélioration nécessiterait toutefois des compétences et installations supplémentaires. Par ailleurs, pour l'IA, l'implémentation d'algorithmes plus avancés rendrait l'adversaire virtuel plus stratégique et prédictif, augmentant le défi pour les joueurs. L'intégration de l'apprentissage automatique, bien que complexe, permettrait à l'IA d'évoluer en s'adaptant aux parties précédentes, nécessitant une base de données de jeux et des algorithmes spécifiques. De plus, optimiser l'algorithme d'exploration augmenterait l'efficacité du jeu.

Bien que ces améliorations soient ambitieuses, elles promettent d'ouvrir des voies passionnantes pour le développement futur de notre jeu.

Copyright