RAPPORT DE PROJET - PROGRAMMATION C

PROJET JEU "INCOGNITO"

Projet réalisé par

Kylian DESAULNOIX
Florian GRANDPIERRE

Rapport rédigé par

Florian GRANDPIERRE

Binôme Z

30 Septembre 2024 - 27 Octobre 2024

1 Introduction

Le devoir maison de programmation C avait pour objectif de développer le jeu Incognito en utilisant le langage de programmation C. Incognito est un jeu de plateau jouable à 2 joueurs. Le but du jeu est de capturer l'espion de l'adversaire par interrogation, ou de faire atteindre son propre espion au château ennemi sans connaître les positions des espions. A chaque tour un joueur peut effectuer soit un déplacement à la méthode d'une dame aux jeu d'échec ou alors une interrogation sur un pion voisin orthogonal.

2 Organisation

Le développement du projet a été divisé en deux parties pour les deux membres du groupe, chacun prenant en charge un aspect spécifique du jeu. Répartir les tâches a permis de progresser de manière efficace et complémentaire dans la réalisation du jeu.

Florian

Florian, en tant que responsable du fonctionnement du jeu a travaillé sur la mise en place du moteur du jeu. Incluant la définition des structures de données ainsi que l'implémentation des fonctions de base du moteur, telles que le déplacement des pions. Son rôle a été crucial pour s'assurer le bon déroulement du jeu.

Kylian

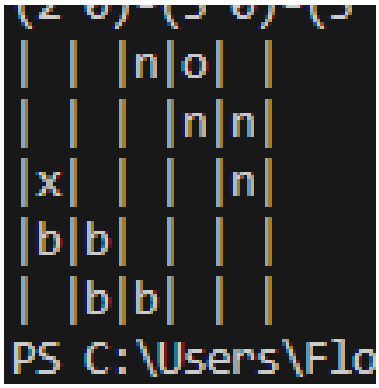
Kylian, de son côté, a pris en charge une partie du fonctionnement des interrogations, plus précisement certaines conditions et l'assemblage des fonctions. Il a pris en charge l'affichage du plateau de jeu, en s'assurant d'intégrer de manière assez compréhensible les éléments Son travail a contribué à rendre le jeu plus attractif visuellement, tout en assurant une clarté de jeu. L'objectif était de créer une interface simple et intuitive, facilitant la compréhension du déroulement du jeu.

3 Progression de la réalisation du jeu

3.1 Semaine du 7 octobre 2024

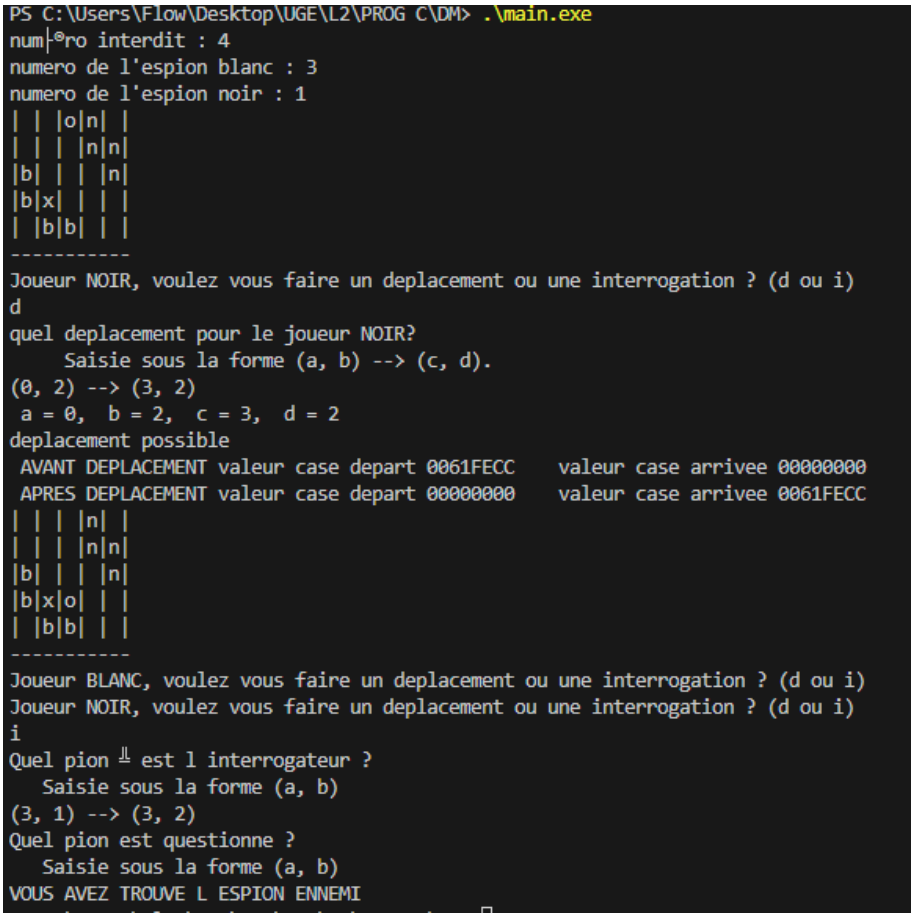
Nous avons commencé à répartir les taches et à structurer la logique du jeu en prévoyant d'avance les différents cas de figure auxquels nous devront faire face, plus précisément toutes les possibilités et conditions pour les pions qui seront nécessaires pour vérifier qu'un déplacement ou une interrogation est faisable. Création de schémas afin de mieux se comprendre, de noter les idées,

d'organiser les données et finalement de prévoir à quoi doit ressembler le produit final. Florian a commencé par développer la base du jeu en créant le plateau et les pions à partir des typedef fournis dans les consignes données et Kylian s'est tout de suite occupé de l'affichage console pour avoir un aperçu du jeu dès le départ et pouvoir mieux s'organiser et mieux observer le comportement des actions de nos fonctions. Suite à cela nous avons implanté les fonctions déplacement, movPossible, ordonneeVide, abscisseVide, diagonaleVide et checkChateau qui fonctionnent toutes ensemble pour demander au joueur de faire un déplacement et vérifier toutes les conditions requises pour assurer le bon fonctionnement du jeu ainsi que le respect des règles de celui ci.



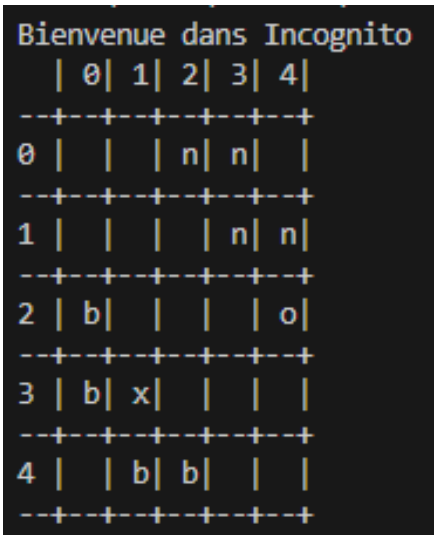
3.2 Semaine du 14 octobre 2024

Lors de cette semaine nous avons commencé par régler quelques soucis liés aux conditions sur les déplacements, comme par exemple le fait qu'un pion allié pouvait aller dans son propre château, pour remedier à cela nous avons alors mis en place la fonction dansSonChateau, le détail des différentes fonctions est expliqué dans la documentation technique qui viendra ensuite. Après avoir résolu ces quelques bugs nous avons commencé la création de la fonction interrogation, l'enjeu ici est encore les différentes conditions car selon le pion interrogateur et le pion interrogé cela pouvait avoir un impact direct sur l'issue de la partie. La fonction interrogation est liée à la fonction dansQuatreVoisins vérifiant la position de la case interrogée comme étant une des quatre case voisine de la case qui interroge.



3.3 Semaine du 21 octobre 2024

Durant cette dernière semaine nous avons acceléré la cadence pour parfaire au mieux et proposer un jeu le plus complet possible. Nous avons commencé par développer la gestion des options en ligne de commande en récupérant les arguments et en agissant en fonction, pour débuter nous n'avions implémenté aucune option. Après avoir réalisé la gestion des options nous avons mis en place la sauvegarde qui fut assez simple à mettre en place, il suffisait d'ouvrir correctement le fichier selon les options et de convertir et écrire les coordonnées des espions et des mouvements selon les normes demandées dans les consignes. Enfin après avoir mis en place la sauvegarde nous avons débuté la création de la fonction chargerJeu permettant de reprendre une partie sauvegardée precedemment et de si souhaité y enregistrer la suite de celle ci dans un nouveau fichier. Enfin nous avons réajusté l'affichage pour le rendre plus intuitif. (Ne pas prendre en compte le x et le o qui sont présents sur cette version de l'affichage pour des raisons de debug.)



4 Documentation utilisateur

4.1 Règles du jeu

Déroulement du jeu : Le jeu commence avec les deux joueurs ayant leurs pions positionnés sur le plateau. Les joueurs jouent chacun leur tour. À chaque tour, un joueur peut soit déplacer un de ses pions d'un nombre illimité de cases dans une des huit direction tant qu'elles sont vides soit interroger un pion ennemi tant qu'il est dans une des 4 cases adjacentes. A savoir que chacun des joueurs n'a pas connaissance de quel pion est son espion.

- Chaque joueur dispose de plusieurs chevaliers et d'un espion, placés sur un plateau carré de taille $T \times T$.
- Les pions peuvent se déplacer horizontalement, verticalement ou en diagonale, à condition que le chemin soit libre.
- Un joueur peut interroger un pion ennemi adjacent (orthogonalement) à l'un de ses pions.
- Si un espion interroge un chevalier ennemi, l'espion est éliminé et le joueur perd.
- Si un chevalier interroge un espion ennemi, l'espion est éliminé et le joueur gagne.
- Un joueur gagne en capturant l'espion adverse ou en faisant atteindre son espion au château ennemi.

Le plateau est affiché avec les coordonnées des lignes et des colonnes, et chaque pion est représenté par un caractère spécifique :

- Case vide : ' ' (espace)
- Pion noir : 'n'
- Pion blanc : 'b'

4.2 Lancement et paramétrage du jeu

Pour compiler le jeu, assurez-vous que le fichier `incognito.c` est dans votre répertoire de travail. Exécutez la commande suivante dans un terminal :

```
gcc -o incognito incognito.c
```

Lancez le jeu avec les options de votre choix :

```
./jeu [options]
```

Le programme accepte plusieurs options pour configurer l'affichage, le mode de jeu et la sauvegarde. Voici la liste des options disponibles :

- `-a` : Affiche le plateau en mode ASCII dans le terminal. Cette option est exclusive avec l'option `-g`.
- `-g` : Affiche le plateau en mode graphique (non implémenté). Cette option est exclusive avec l'option `-a`.
- `-s <chemin_du_fichier>` : Sauvegarde chaque coup joué par les joueurs dans le fichier spécifié. Le chemin d'accès au fichier doit suivre directement l'option. Par exemple :

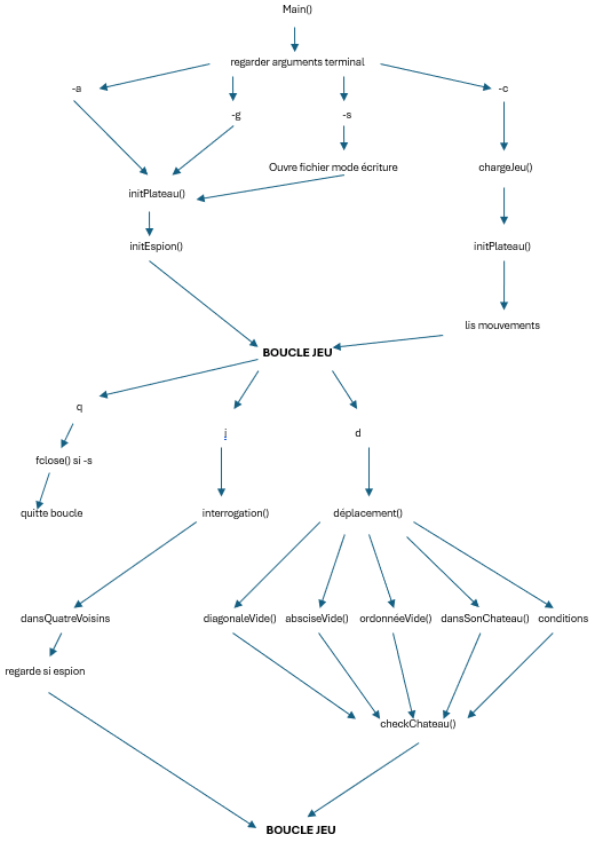
  ```
      ./jeu -s chemin/vers/fichier.txt
  ```

  Cette commande lancera le jeu et enregistre chaque coup dans le fichier `fichier.txt`.
- `-c <chemin_du_fichier>` : Charge une partie sauvegardée depuis un fichier. Le chemin d'accès au fichier doit suivre directement l'option. Par exemple :

  ```
      ./jeu -c chemin/vers/sauvegarde.txt
  ```

  Cette commande chargera la partie depuis `sauvegarde.txt`.

5 Documentation technique



5.1 Couleur

```
1 /* Definition des couleurs des pions */
2 typedef enum _couleur {BLANC, NOIR} Couleur;
```

5.2 Type

```
1 /* Definition des types de pions */
2 typedef enum _type{CHEVALIER, ESPION} Type;
```

5.3 Pion

```c
/* Structure representant un pion */
typedef struct _pion {
    Type type;
    Couleur couleur;
} Pion;
```

5.4 Jeu

```c
/* Structure representant le jeu */
typedef struct _jeu {
    Pion * plateau[T][T];
    Couleur joueur;
} Jeu;
```

5.5 Case et Direction

```c
/* Structure pour representer une case ou une direction */
typedef struct {
    int x, y;
} Case, Direction;
```

5.6 Mouvement

```c
/* Structure representant un mouvement */
typedef struct _mouvement {
    Case depart;
    Case arrivee;
} Mouvement;
```

5.7 affichePlateauAscii

```c
void affichePlateauAscii(int taille, Jeu jeu, Pion *chevnoir
    , Pion *chevblanc, Pion *espnoir, Pion *espblanc);
```

Description

Affiche le plateau de jeu en mode ASCII dans le terminal. Chaque type de pion est représenté par un caractère spécifique :

- `'n'` : Pion noir
- `'b'` : Pion blanc
- `' '` : Case vide

Paramètres

- `taille` : Taille du plateau de jeu.
- `jeu` : Structure du jeu contenant le plateau et le joueur courant.
- `chevnoir` : Pointeur vers le chevalier noir.
- `chevblanc` : Pointeur vers le chevalier blanc.
- `espnoir` : Pointeur vers l'espion noir.
- `espblanc` : Pointeur vers l'espion blanc.

Retour

Aucun.

5.8 affiche

```
1 void affiche(int opt, int taille, Jeu jeu, Pion *chevnoir,
    Pion *chevblanc, Pion *espnoir, Pion *espblanc);
```

Description

Permet de choisir entre l'affichage ASCII et l'affichage graphique du plateau de jeu. Dans l'implémentation actuelle, seul l'affichage ASCII est disponible.

Paramètres

- `opt` : Option d'affichage (0 pour ASCII, 1 pour graphique).
- `taille` : Taille du plateau de jeu.
- `jeu` : Structure du jeu.
- `chevnoir`, `chevblanc`, `espnoir`, `espblanc` : Pointeurs vers les différents types de pions.

Retour

Aucun.

5.9 diagonaleVide

```
1 int diagonaleVide(Jeu jeu, Mouvement mov);
```

Description

Vérifie si toutes les cases sur la diagonale entre la case de départ et la case d'arrivée d'un mouvement sont vides.

Paramètres

— `jeu` : Structure du jeu.
— `mov` : Mouvement à vérifier.

Retour

— `0` si la diagonale est vide.
— `1` sinon.

5.10 `abscisseVide`

```
1 int abscisseVide(Jeu jeu, Mouvement mov);
```

Description

Vérifie si toutes les cases sur l'axe horizontal (abscisse) entre la case de départ et la case d'arrivée d'un mouvement sont vides.

Paramètres

— `jeu` : Structure du jeu.
— `mov` : Mouvement à vérifier.

Retour

— `0` si l'abscisse est vide.
— `1` sinon.

5.11 `ordonneeVide`

```
1 int ordonneeVide(Jeu jeu, Mouvement mov);
```

Description

Vérifie si toutes les cases sur l'axe vertical (ordonnée) entre la case de départ et la case d'arrivée d'un mouvement sont vides.

Paramètres

— `jeu` : Structure du jeu.
— `mov` : Mouvement à vérifier.

Retour

— `0` si l'ordonnée est vide.
— `1` sinon.

5.12 dansSonChateau

```
1 int dansSonChateau(Jeu jeu, int x, int y, int taille);
```

Description

Vérifie si un joueur tente de placer un de ses pions dans son propre château, ce qui est interdit.

Paramètres

— `jeu` : Structure du jeu.
— `x`, `y` : Coordonnées du pion.
— `taille` : Taille du plateau de jeu.

Retour

— `1` si le pion est dans son propre château.
— `0` sinon.

5.13 movPossible

```
1 int movPossible(int taille, Jeu jeu, Mouvement mov, Pion *
    chevnoir, Pion *chevblanc, Pion *espnoir, Pion *espblanc)
    ;
```

Description

Vérifie si un mouvement est possible en respectant les règles du jeu, y compris les déplacements autorisés et les contraintes liées aux pions.

Paramètres

— `taille` : Taille du plateau.
— `jeu` : Structure du jeu.
— `mov` : Mouvement à vérifier.
— `chevnoir`, `chevblanc`, `espnoir`, `espblanc` : Pointeurs vers les différents types de pions.

Retour

— `0` si le mouvement est possible.
— `1` sinon.

5.14 dansQuatreVoisins

```
1 int dansQuatreVoisins(int taille, Jeu *jeu, Mouvement mov);
```

Description

Vérifie si la case d'arrivée est dans les quatre voisins orthogonaux (haut, bas, gauche, droite) de la case de départ. Utilisé lors des interrogations.

Paramètres

— `taille` : Taille du plateau.
— `jeu` : Pointeur vers la structure du jeu.
— `mov` : Mouvement à vérifier.

Retour

— `0` si la case d'arrivée est un voisin valide.
— `1` sinon.

5.15 deplacement

```
1 int deplacement(FILE *f, Jeu *jeu, int taille, Pion *
    chevnoir, Pion *chevblanc, Pion *espnoir, Pion *espblanc)
    ;
```

Description

Gère le déplacement des pions sur le plateau. Demande au joueur courant de saisir un mouvement et vérifie sa validité.

Paramètres

— `f` : Pointeur vers le fichier de sauvegarde (peut être `NULL`).
— `jeu` : Pointeur vers la structure du jeu.
— `taille` : Taille du plateau.
— `chevnoir`, `chevblanc`, `espnoir`, `espblanc` : Pointeurs vers les différents types de pions.

Retour

— `0` si le déplacement a été effectué avec succès.
— `1` en cas d'erreur.

5.16 interrogation

```
int interrogation(FILE *f, Jeu *jeu, int taille, Pion *
    chevnoir, Pion *chevblanc, Pion *espnoir, Pion *espblanc)
    ;
```

Description

Gère l'interrogation d'un pion ennemi. Permet au joueur courant d'interroger un pion ennemi adjacent à l'un de ses pions.

Paramètres

- `f` : Pointeur vers le fichier de sauvegarde (peut être `NULL`).
- `jeu` : Pointeur vers la structure du jeu.
- `taille` : Taille du plateau.
- `chevnoir`, `chevblanc`, `espnoir`, `espblanc` : Pointeurs vers les différents types de pions.

Retour

- `0` si l'espion ennemi a été capturé.
- `1` si l'espion allié a été perdu.
- `2` si l'interrogation n'a pas abouti.
- `3` en cas d'erreur de saisie.

5.17 checkChateau

```
int checkChateau(Jeu *jeu, int taille, Pion *espnoir, Pion *
    espblanc);
```

Description

Vérifie si un espion a atteint le château adverse, ce qui met fin à la partie.

Paramètres

- `jeu` : Pointeur vers la structure du jeu.
- `taille` : Taille du plateau.
- `espnoir`, `espblanc` : Pointeurs vers les espions.

Retour

- `1` si un espion a atteint le château adverse.
- `0` sinon.

5.18 initPlateau

```
1 void initPlateau(Jeu *jeu, int taille, Pion *chevnoir, Pion
    *chevblanc);
```

Description

Initialise le plateau de jeu en plaçant les chevaliers sur leurs positions de départ.

Paramètres

— `jeu` : Pointeur vers la structure du jeu.
— `taille` : Taille du plateau.
— `chevnoir`, `chevblanc` : Pointeurs vers les chevaliers.

Retour

Aucun.

5.19 initEspion

```
1 void initEspion(FILE *f, Jeu *jeu, int taille, Pion *espnoir
    , Pion *espblanc);
```

Description

Place aléatoirement les espions sur le plateau en remplaçant deux chevaliers.

Paramètres

— `f` : Pointeur vers le fichier de sauvegarde (peut être `NULL`).
— `jeu` : Pointeur vers la structure du jeu.
— `taille` : Taille du plateau.
— `espnoir`, `espblanc` : Pointeurs vers les espions.

Retour

Aucun.

5.20 clearInputBuffer

```
1 void clearInputBuffer();
```

Description

Vide le tampon d'entrée pour éviter les problèmes lors de la saisie utilisateur.

Paramètres

Aucun.

Retour

Aucun.

5.21 chargeJeu

```
void chargeJeu(FILE *f, Jeu *jeu, int taille, Pion *chevnoir
    , Pion *chevblanc, Pion *espnoir, Pion *espblanc);
```

Description

Charge une partie sauvegardée depuis un fichier et initialise le plateau de jeu en conséquence.

Paramètres

— `f` : Pointeur vers le fichier de sauvegarde.
— `jeu` : Pointeur vers la structure du jeu.
— `taille` : Taille du plateau.
— `chevnoir`, `chevblanc`, `espnoir`, `espblanc` : Pointeurs vers les pions.

Retour

Aucun.

5.22 main

```
int main(int argc, char* argv[]);
```

Description

Point d'entrée principal du programme. Gère la boucle principale du jeu, l'initialisation, le traitement des options en ligne de commande et la gestion des tours de jeu.

Paramètres

— `argc` : Nombre d'arguments en ligne de commande.
— `argv` : Tableau des arguments en ligne de commande.

Retour

— `0` si le programme s'est exécuté correctement.
— Autre valeur en cas d'erreur.