# Rapport du projet d’IFB : Jeu de Belote Coinché en C dans la console

## Introduction

Dans le carde de l’UV IFB nous avons du réaliser un jeu de Belote Coinché en C. Le cahier des charge nous impose de programmer uniquement en C et de faire un programme qui s’exécute dans la console.

## Sources utilisée

<https://www.belote.com/regles-et-variantes/regle-belote-coinche/>

<http://www.ffbelote.org/regles-coinche/#7>

## Organisation générale du projet

Afin de réaliser ce projet et sachant qu’à cause du confinement il serait très difficile pour les membres du projet de se retrouver en présentiel pour ce coordonner sur le projet nous avons utiliser la plateforme discord afin de communiquer sur les problème que nous avons rencontré et sur les fonction à réaliser. Pour stoker et versionner notre code, nous avons utiliser la plateforme GitHub qui nous permet de travailler simultanément sur le projet et de toujours avoir la version du code la plus à jour. Voici le lien de notre projet public sur GitHub :

<https://github.com/Flo3171/IFB_projet_Belote>

(Flo3171 est le pseudo de Florian CLOAREC et Fituning celui de Carlo AZANCOTH)

## Solutions techniques et fonctionnement du programme

Nous allons maintenant vous détailler les solutions techniques que nous avons mis en place et la façon dont nous avons réalisé les différents points du cahier des charges dans l’ordre chronologique ou cela à été implémenter dans le projet.

### Modularité

Dès le début du projet vous avions conscience que le projet allait être composé d’un grand nombre de fonctions et de lignes de code, c’est pourquoi afin d’avoir un projet clair et ordonné, nous avons décidé de séparer les différentes fonctions dans des fichiers séparés, à l’intérieur de ces fichiers les fonctions sont regroupées selon un thème commun (affichage, formatage, gestion des cartes, gestion des fichier…). Afin d’inclure les prototypes des fonctions à tous les endroits où cela est nécessaire nous avons créé un fichier main.h, ce ficher regroupe toute les constantes, les énumérations est les structure utilisées dans ce projet, mais c’est surtout dans ce fichier que sont inclut tous les fichier .h associer à chaque fichier .c contenant nos fonctions. Ce fichier main.h est alors inclus au début de chaque fichier contenant les fonctions du projet. Ainsi cela permet de s’assurer que toutes les fonctions créées spécifiquement pour ce projet ainsi que tous les autres objets susceptible d’être manipulé par les fonctions soient utilisables qu’importe le fichier dans lequel elles se trouvent. C’est aussi dans ce fichier main.h que nous avons inclus les bibliothèques standards que nous utilisons dans le programme. Nous avons bien conscience que cette solution n’est pas la plus optimale, en effet dans certains fichiers des portions de code sont incus alors qu’elles ne seront jamais utilisées, ce qui augmente de façon non négligeable la taille de l’exécutable du programme. Mais nous avons choisi cette solution car elle permet une grande libérer lors du développement du projet, en effet avec autant de fichier on est souvent amené à passer très souvent d’un fichier à l’autre. Grâce à cette solution lorsque l’on veut rajouter une fonction ou simplement appeler une autre fonction dans une fonction, il n’est pas nécessaire de s’assurer que le prototype de la fonction est bien inclus dans le fichier ou l’on travail. Cela permet de se concentrer sur ce que l’on fait, de gagner du temps et surtout d’éviter de nombreuses erreurs de compilations. Néanmoins cette manière de faire entraine un autre problème, en effets avec autant de fichier inclus les uns dans les autres il est alors possible que par inadvertance on crée une boucle d’inclusion infini ce qui empêcherais la compilation, c’est pourquoi chacun des fichiers .h est entouré d’un code préprocesseur qui permet de ne compiler le contenu du fichier seulement si c’est la première fois qu’il est lu par le préprocesseur.

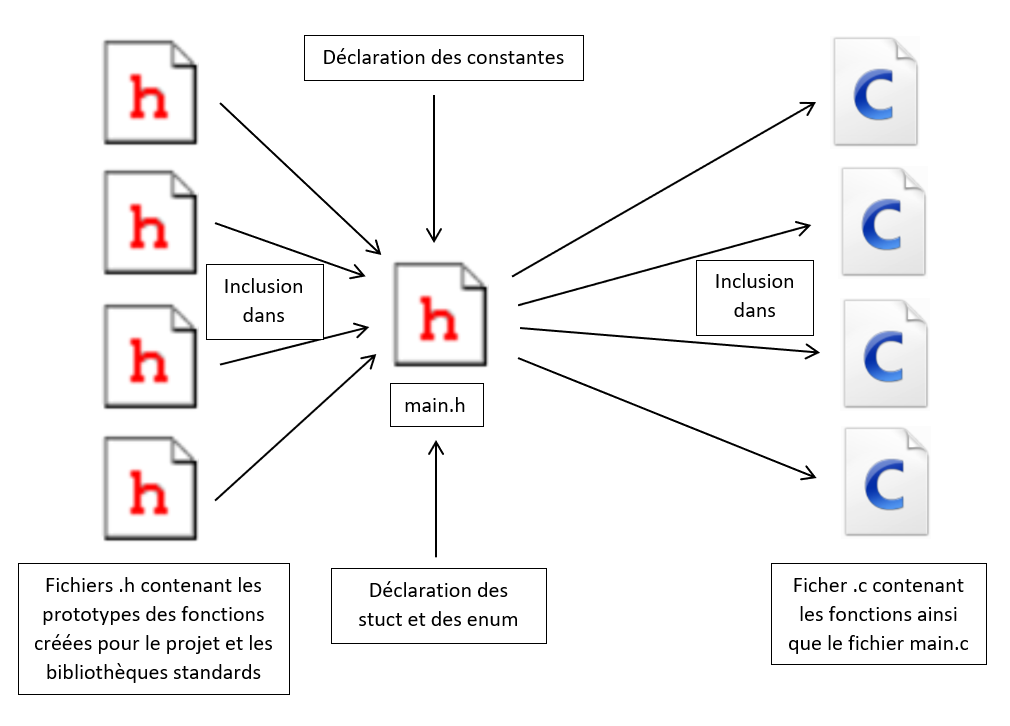
/\*Code utilisé pour protéger le programme des boucle d'inclusions infinie\*/

#ifndef \_NOM\_FICHIER\_H\_

#define \_NOM\_FICHIER\_H\_

/\*contenu du fichier\*/

#endif /\* \_NOM\_FICHIER\_H\_ \*/



### Les type de variable personnalisée

Dans ce projet un grand nombre de donnée à stoker en mémoire n’était pas sous la forme d’un nombre afin d’éviter de devoir stocker ces information sous forme de chaine de caractère ce qui aurai été très lourd a manipuler et aurai pris beaucoup de place en mémoire, nous avons décidé de créer des types de variable. Nous avons donc créé un enum nommer Couleur et une autre nommer Valeur puis une stcuct nommer Carte qui est composée d’une sous variable de type Couleur et un sous variable de type Valeur ainsi nous pouvons manipuler les Carte comme une unique variable et il devient alors très facile de manipuler des tableau de variable de type Carte. Nous avons aussi créé des type de variable personnalisé pour les Contrat

### L’affichage

Nous avons décidé de développer les fonctions d’affichage dès le début du projet afin de pouvoir voir ce qui ce passe lors des test et du debug des autres fonctions. Etant donné que nous utilisons des type de variable personnalisé pour les carte et non des chaine de caractères, nous ne pouvions pas afficher directement le contenu de la variable tel quel. C’est pourquoi nous avons mis en place les fonctions de formatages, qui permettent de faire la convention entre une variable de type Carte (ou Contrat) et une ou plusieurs chaine de caractère. Puis ces chaine de caractère une fois formatée sont afficher dans la console à l’aide de la fonction printf par fonction chargées de l’affichage.

Sachant que nous devons réaliser ce projet et qu’il n’était pas envisageable de mettre en place une interface graphique, nous avons tenté de rendre l’interface avec l’utilisateur la plus belle et ergonomique, pour ce faire nous avons créé des fonctions de formatage qui permettent de centrer les chaines de caractère pour que le pseudo des joueur quelle qu’il soit s’affiche toujours au centre de l’espace où il est sensé s’afficher. Nous avons de plus prévu le cas où la chaine de caractère à afficher déplacerais la taille disponible dans le cadre ou elle est affichée, dans ce cas, la chaine est coupée afin de ne pas décaler les autres affichages qui sont parfois sur la même ligne. Afin de formater les messages de tailles plus longues une fonction (decoupeChaine) se charge de couper la chaine au niveau d’un espace et d’afficher le reste de la chaine sur la ligne suivante.

### Les Menus (Maxi best of frite coca)

### Gestion des pseudo

### Phase d’annonces des contrats

### Calcul de la force d’une carte

### Intelligence Artificielle (ia)

Afin de réaliser ce programme, nous avons dû crée deux ia. La première permet de déterminer quelle sera le contrat que l’ordinateur va prendre ou s’il va passer et la seconde permet de choisir quelle carte l’ordinateur va jouer lors de chaque pli. Ces deux ia prennent la forme de deux fonction (proposeContratIa et choixCarteIa) qui prennent comme paramètre de nombreuse information sur la partie comme les joueur que le joueur à en main ; et retournent respectivement une variable de type contrat et un entier qui correspond à la position de la carte à jouer dans la main du joueur.

Afin de détemier quelle contrat l’ordinateur va prendre on calcule la force de la main, cette force est un nombre réelle comprise entre 0 et 1