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

## Introduction

Dans le carde de l’UV IFB nous avons du réaliser un jeu de Belote Coinchée 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.

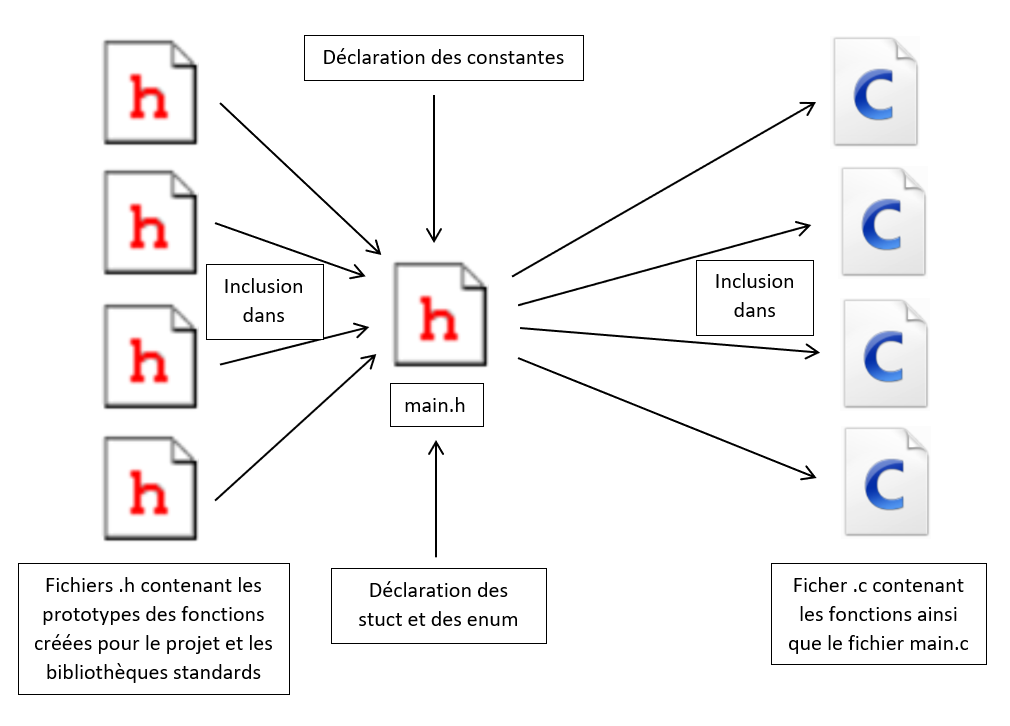
/\*Code utilisé pour protéger le programme des boucles d'inclusions infinie\*/

#ifndef \_NOM\_FICHIER\_H\_

#define \_NOM\_FICHIER\_H\_

/\*contenu du fichier\*/

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



### Les type de variable personnalisée

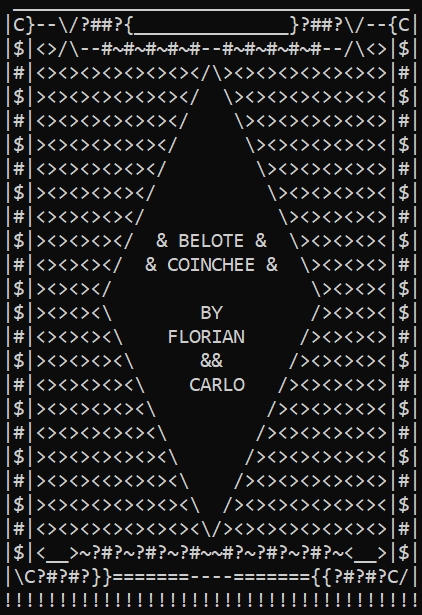
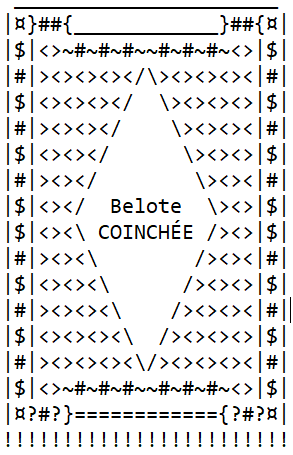
Dans ce projet un grand nombre de donnée à stoker en mémoire n’était pas sous la forme d’un nombre afin d’éviter de devoir stocker ces informations sous forme de chaine de caractère, ce qui aurai été très lourd à manipuler et aurai pris beaucoup de place en mémoire. Nous avons décidé pour éviter ce problème de créer des types de variable. Nous avons donc créé une enum nommée Couleur, une autre nommée Valeur puis une stcuct nommé Carte qui est composée d’une sous-variable de type Couleur et une sous-variable de type Valeur. Ainsi nous pouvions manipuler les variables de type Carte comme une unique variable et il devient alors très facile de manipuler des tableaux de variable de type Carte. Nous avons aussi créé des types 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 se passe lors des tests et du debug des autres fonctions. Etant donné que nous utilisons des types de variable personnalisé pour les cartes et non des chaines 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 chaines de caractère. Puis ces chaines de caractère une fois formatée sont affichées 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 joueurs 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 dépasserais 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

 Afin que l’utilisateur puisse utiliser toutes les fonctionnalités que nous avons développées, nous avons ajouté la fonction menuPrincipal qui est l’unique fonction appelée dans le main.c. Premièrement la fonction affiche le logo (qui représente une carte), que l’on a conçu avec les 256 caractère du code ASCII.

Nous avons donc réalisé 2 versions du logo, car nous nous sommes rendus compte que le premier logo était trop petit pour afficher à l’intérieur les différents sous-menus. Ensuite la fonction se charge d’acquerir le pseudo du joueur qui est directement enregistrée dans les fichiers de sauvegarde gestion\_scores\_joueurs.csv si le joueur n’existe pas déjà et enregistre la ligne à laquelle est stockée sa sauvegarde.

### Gestion des pseudos

### Phase d’annonces des contrats

### Calcul de la force d’une carte

Afin de déterminer quelle carte est la plus forte et va remporter le plis, la fonction forceCarte permet d’associer à chaque carte une valeur numérique comprise entre 0 et 1 qui reflète ca force, plus ce nombre sera grand plus la carte sera forte. Après avoir cherché qu’elle serait la méthode la plus efficace et la plus pratique afin de donner une force à chaque carte, nous avons décider de passer par une approche probabiliste. Ainsi la couleur de l’atout (qui peut être sans-atout et tout-atout) est et la couleur de l’entame (la première carte jouée dans le plis) sont des paramètres de la fonction forceCarte. La force d’une carte se calcul avec la formule suivante : . Ce qui signifie par exemple que le valet d’atout va battre les 31 autres cartes du paquet et ainsi il aura une force de 1, à l’inverse le 7 de non-atout s’il n’est pas dans la couleur de l’entame ne bâtera aucune carte et aura donc une force de 0. Avoir fait des enum et des struct pour définir la variable personnalisée Carte nous permet de manipuler les sous variable .couleur et .valeur comme des entier et ainsi nous pouvons utiliser des switch qui dans le cas de cette fonction améliorent grandement la lisibilité du code. Grâce à cette fonction, il devient très facile de trouver le vainqueur d’un pli, il faut alors chercher quelle carte à la force maximum parmi les cartes du plis. Mais cette fonction à aussi d’autres usages, elle est utilisée dans les fonctions ia, afin d’évaluer et de pouvoir faire des calcul dessus afin de prendre une décision.

### Intelligence Artificielle (ia)

Afin de réaliser ce programme, nous avons dû créer 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 cartes 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 déterminer quelle contrat l’ordinateur va prendre, le programme fait la somme de la force de chacune des cartes dans toutes les couleur possible (y compris tout-atout et sans-atout) et regarde dans quelle couleur la valeur est la plus grande, une grande valeur correspond à un main forte dans la couleur. En faisant des test sur cette méthode nous avons découvert que la somme des forces des carte de la main était plus grande en tout atout que dans les autres couleur, afin de régler ce problème, nous avons remis cette valeur à l’échelle, ainsi la plus forte main en tout-atout vaut la même valeur que la plus forte main dans une autre couleur et le nombre que l’on obtient est compris entre 0 et 1. Puis la fonction regarde si ce nombre dans la meilleur couleur est supérieur au seuil minimum de prise de contrat. Si tel est le cas, l’ordinateur va chercher à prendre un contrat dans cette couleur. La valeur du contrat que l’ordinateur va prendre est proportionnelle au nombre trouver précédemment plus ce nombre sera supérieur à la valeur du seuil minimum de prise de contrat, plus l’ordinateur prendra un contrat avec un nombre important de points. Afin de respecter les règle du jeux, la fonction va alors tester si le contrat quelle veut proposer est valide et pour se faire on teste, si le nombre de point proposé par ce nouveau contrat est supérieur au dernier contrat proposé (qui est celui avec le plus grand nombre de point sachant que les autres joueur doivent aussi surenchérir), alors la fonction renvoie une variable de type contrat avec le nombre de points et la couleur qu’elle à déterminer. Sinon ou si sa main ne lui permet pas de dépasser le seuil minimal de prise alors elle passe, ce qui ce matérialise par le renvoi pas la fonction d’une variable de type Contrat dont le nombre de point vaut 0.

-->fonction choixCartIa