

IFB_projet
V1.0

Généré par Doxygen 1.8.18

1 Index des structures de données	1
1.1 Structures de données	1
2 Index des fichiers	3
2.1 Liste des fichiers	3
3 Documentation des structures de données	5
3.1 Référence de la structure Carte	5
3.1.1 Description détaillée	5
3.1.2 Documentation des champs	5
3.1.2.1 couleur	5
3.1.2.2 valeur	6
3.2 Référence de la structure Contrat	6
3.2.1 Description détaillée	6
3.2.2 Documentation des champs	6
3.2.2.1 atout	6
3.2.2.2 coinche	7
3.2.2.3 nbPoint	7
3.2.2.4 preneur	7
4 Documentation des fichiers	9
4.1 Référence du fichier acquisition.c	9
4.2 Référence du fichier acquisition.h	9
4.2.1 Description détaillée	9
4.2.2 Documentation des fonctions	10
4.2.2.1 acquisitionEntierAvecMessage()	10
4.2.2.2 acquisitionEntierSansMessage()	10
4.2.2.3 acquisitionEntierSansMessageAvecConsigne()	10
4.2.2.4 acquisitionEntierSecurise()	11
4.2.2.5 acquisitionOuiNonSansMessage()	11
4.2.2.6 acquisitionPseudoAvecMessage()	11
4.3 Référence du fichier affichage.c	12
4.3.1 Description détaillée	12
4.3.2 Documentation des fonctions	13
4.3.2.1 afficheContrat()	13
4.3.2.2 afficheInterfacePli()	13
4.3.2.3 afficheMain()	14
4.3.2.4 afficheMenuPrincipal()	14
4.3.2.5 afficheMenuSelection()	15
4.3.2.6 afficheSousMenus()	15
4.3.2.7 modifieTailleFenetre()	16
4.3.2.8 proposeContratUtilisateur()	16
4.4 Référence du fichier affichage.h	17

4.4.1 Description détaillée	17
4.4.2 Documentation des fonctions	18
4.4.2.1 afficheContrat()	18
4.4.2.2 afficheInterfacePli()	18
4.4.2.3 afficheMain()	19
4.4.2.4 afficheMenuPrincipal()	20
4.4.2.5 afficheMenuSelection()	20
4.4.2.6 afficheSousMenus()	20
4.4.2.7 modifieTailleFenetre()	21
4.4.2.8 proposeContratUtilisateur()	21
4.5 Référence du fichier autre.c	22
4.5.1 Description détaillée	22
4.5.2 Documentation des fonctions	23
4.5.2.1 ajusteEchelle()	23
4.5.2.2 joue1000Partie()	23
4.5.2.3 joueurSuivant()	24
4.5.2.4 nbAleatoire()	24
4.5.2.5 pointPli()	24
4.5.2.6 setContrat()	25
4.6 Référence du fichier autre.h	25
4.6.1 Description détaillée	26
4.6.2 Documentation des fonctions	26
4.6.2.1 ajusteEchelle()	26
4.6.2.2 joue1000Partie()	27
4.6.2.3 joueurSuivant()	27
4.6.2.4 nbAleatoire()	27
4.6.2.5 pointPli()	28
4.6.2.6 setContrat()	28
4.7 Référence du fichier formatageChaine.c	29
4.7.1 Description détaillée	29
4.7.2 Documentation des fonctions	30
4.7.2.1 aligneModifieChaine()	30
4.7.2.2 centreChaine()	30
4.7.2.3 centreModifieChaine()	31
4.7.2.4 decoupeChaine()	31
4.7.2.5 formateCarte()	31
4.7.2.6 formateContrat()	32
4.7.2.7 formatePseudo()	33
4.7.2.8 genereMessage()	33
4.7.2.9 rempliEspace()	34
4.7.2.10 stockeInfoCarte()	34
4.8 Référence du fichier formatageChaine.h	35

4.8.1 Description détaillée	35
4.8.2 Documentation des fonctions	35
4.8.2.1 aligneModifieChaine()	35
4.8.2.2 centreChaine()	36
4.8.2.3 centreModifieChaine()	36
4.8.2.4 decoupeChaine()	37
4.8.2.5 formateCarte()	37
4.8.2.6 formateContrat()	38
4.8.2.7 formatePseudo()	38
4.8.2.8 genereMessage()	39
4.8.2.9 rempliEspace()	39
4.8.2.10 stockeInfoCarte()	40
4.9 Référence du fichier general.c	40
4.9.1 Description détaillée	41
4.9.2 Documentation des fonctions	41
4.9.2.1 annonceContrat()	41
4.9.2.2 calculPointManche()	42
4.9.2.3 initialisation()	43
4.9.2.4 manche()	43
4.9.2.5 menuPrincipal()	44
4.9.2.6 nouvellePartie()	44
4.9.2.7 pli()	45
4.9.2.8 poseCarte()	46
4.9.2.9 proposeContrat()	46
4.10 Référence du fichier general.h	47
4.10.1 Description détaillée	47
4.10.2 Documentation des fonctions	48
4.10.2.1 annonceContrat()	48
4.10.2.2 calculPointManche()	48
4.10.2.3 initialisation()	49
4.10.2.4 manche()	49
4.10.2.5 menuPrincipal()	50
4.10.2.6 nouvellePartie()	50
4.10.2.7 pli()	51
4.10.2.8 poseCarte()	52
4.10.2.9 proposeContrat()	52
4.11 Référence du fichier gestionCarte.c	53
4.11.1 Description détaillée	54
4.11.2 Documentation des fonctions	54
4.11.2.1 cartePlaceAvant()	54
4.11.2.2 carteValide()	55
4.11.2.3 distribueCarte()	55

4.11.2.4 forceCarte()	56
4.11.2.5 rechercheAnnonce()	56
4.11.2.6 rechercherCarte()	57
4.11.2.7 rechercherCarteSuperieur()	57
4.11.2.8 setCarte()	59
4.11.2.9 sommeForceCarte()	59
4.11.2.10 supprimeCarte()	60
4.11.2.11 trieCarte()	60
4.11.2.12 vainqueurPli()	61
4.12 Référence du fichier gestionCarte.h	61
4.12.1 Description détaillée	62
4.12.2 Documentation des fonctions	62
4.12.2.1 cartePlaceAvant()	62
4.12.2.2 carteValide()	63
4.12.2.3 distribueCarte()	64
4.12.2.4 forceCarte()	64
4.12.2.5 rechercheAnnonce()	65
4.12.2.6 rechercherCarte()	65
4.12.2.7 rechercherCarteSuperieur()	66
4.12.2.8 setCarte()	66
4.12.2.9 sommeForceCarte()	67
4.12.2.10 supprimeCarte()	67
4.12.2.11 trieCarte()	68
4.12.2.12 vainqueurPli()	68
4.13 Référence du fichier gestionFichier.c	69
4.13.1 Description détaillée	69
4.13.2 Documentation des fonctions	69
4.13.2.1 ecrireLeaderboard()	69
4.13.2.2 ecrireStatistique()	70
4.13.2.3 ecriturePseudo()	70
4.14 Référence du fichier gestionFichier.h	71
4.14.1 Description détaillée	71
4.14.2 Documentation des fonctions	71
4.14.2.1 ecrireLeaderboard()	71
4.14.2.2 ecrireStatistique()	72
4.14.2.3 ecriturePseudo()	72
4.15 Référence du fichier ia.c	73
4.15.1 Description détaillée	73
4.15.2 Documentation des fonctions	73
4.15.2.1 choixCartelA()	73
4.15.2.2 proposeContratla()	74
4.16 Référence du fichier ia.h	74

4.16.1 Description détaillée	75
4.16.2 Documentation des fonctions	75
4.16.2.1 choixCartelA()	75
4.16.2.2 proposeContratla()	76
4.17 Référence du fichier main.c	76
4.17.1 Description détaillée	77
4.17.2 Documentation des fonctions	77
4.17.2.1 main()	77
4.18 Référence du fichier main.h	78
4.18.1 Description détaillée	79
4.18.2 Documentation des macros	79
4.18.2.1 DEBUG_MODE	79
4.18.2.2 MODE_1_MANCHE	80
4.18.2.3 NB_CARRACTERE_LEADERBOARD	80
4.18.2.4 NB_CARRACTERE_SCORE	80
4.18.2.5 NB_JOUEUR	80
4.18.2.6 NB_TOATAL_CARTE	80
4.18.2.7 NIVEAU_IA	80
4.18.2.8 POSITION_NB_MANCHES_POUR_GAGNER	81
4.18.2.9 POSITION_NB_VICTOIRE	81
4.18.2.10 POSITION_RECORD_VICTOIRE	81
4.18.2.11 POSITION_SCORE_MAX	81
4.18.2.12 TAILLE_MAXI_COULEUR	81
4.18.2.13 TAILLE_MAXI_MESSAGE	81
4.18.2.14 TAILLE_MAXI_PESEUDO	82
4.18.3 Documentation des définitions de type	82
4.18.3.1 Carte	82
4.18.3.2 Coinche	82
4.18.3.3 Contrat	82
4.18.3.4 Couleur	82
4.18.3.5 Joueur	82
4.18.3.6 NbPoint	82
4.18.3.7 TypeMessage	83
4.18.3.8 Valeur	83
4.18.4 Documentation du type de l'énumération	83
4.18.4.1 Coinche	83
4.18.4.2 Couleur	83
4.18.4.3 Joueur	84
4.18.4.4 NbPoint	84
4.18.4.5 TypeMessage	85
4.18.4.6 Valeur	85
4.19 Référence du fichier sous-menus.c	85

4.19.1 Documentation des fonctions	86
4.19.1.1 leaderboard()	86
4.19.1.2 parametre()	86
4.19.1.3 statistiqueJoueur()	87
4.20 Référence du fichier sous-menus.h	87
4.20.1 Documentation des fonctions	87
4.20.1.1 leaderboard()	87
4.20.1.2 parametre()	88
4.20.1.3 statistiqueJoueur()	88
4.21 Référence du fichier tableau.c	89
4.21.1 Description détaillée	89
4.21.2 Documentation des fonctions	89
4.21.2.1 acquiertTableau()	89
4.21.2.2 afficheTableau()	90
4.21.2.3 aleatoireTableau()	90
4.21.2.4 constanteTableau()	91
4.21.2.5 copie()	91
4.21.2.6 maxi()	92
4.21.2.7 mini()	92
4.21.2.8 moyenneTableau()	92
4.21.2.9 ordonnerTableau()	93
4.21.2.10 rechercheDichotomie()	93
4.21.2.11 sommeTableau()	94
4.21.2.12 zeroSiSuperieur()	94
4.22 Référence du fichier tableau.h	95
4.22.1 Description détaillée	95
4.22.2 Documentation des fonctions	95
4.22.2.1 acquiertTableau()	95
4.22.2.2 afficheTableau()	96
4.22.2.3 aleatoireTableau()	96
4.22.2.4 constanteTableau()	97
4.22.2.5 copie()	97
4.22.2.6 maxi()	98
4.22.2.7 mini()	98
4.22.2.8 moyenneTableau()	98
4.22.2.9 ordonnerTableau()	99
4.22.2.10 rechercheDichotomie()	99
4.22.2.11 sommeTableau()	100
4.22.2.12 zeroSiSuperieur()	100

Chapitre 1

Index des structures de données

1.1 Structures de données

Liste des structures de données avec une brève description :

Carte	Represente une carte d'un jeu de 32 cartes	5
Contrat	Represente un contrat Ã la belote coichÃ©e	6

Chapitre 2

Index des fichiers

2.1 Liste des fichiers

Liste de tous les fichiers avec une brève description :

acquisition.c	Fichier source contenant les fonctions relative a l'acquirion	9
acquisition.h	Fichier header contenant les prototype des fonctions relative a l'acquirion	9
affichage.c	Fichier source contenant les fonctions relative à l'affichage	12
affichage.h	Fichier header contenant les prototypes des fonctions relative à l'affichage	17
autre.c	Fichier source contenant les fonctions qui n'allait dans aucun autre fichier	22
autre.h	Fichier header contenant les prototype des fonctions qui n'allait dans aucun autre fichier	25
formatageChaine.c	Fichier source contenant les fonctions relative au formatage	29
formatageChaine.h	Fichier header contenant les prototypes des fonctions relative au formatage	35
general.c	Fichier source contenant les fonctions les plus impotantes pour le jeu de la belote	40
general.h	Fichier header contenant les prototypes des fonctions les plus impotantes pour le jeu de la belote	47
gestionCarte.c	Fichier contenant les fonctions relative a la gestion des cartes	53
gestionCarte.h	Fichier header contenant les protypes des fonctions relative a la gestion des cartes	61
gestionFichier.c	Fichier source contenant les fonctions relative à la gestion des fichiers	69
gestionFichier.h	Fichier header contenant les prototypes des fonctions relative à la gestion des fichiers	71
ia.c	Fichier source contenant les fonctions relative au intelidences artificielles	73
ia.h	Fichier header contenant les prototypes des fonctions relative au intelidences artificielles	74
main.c	Fichier contenant la fonction main	76

main.h	Fichier la déclaration des contrainte, les énumérations, les structe et l'inclusion des header du projet	78
sous-menus.c	85
sous-menus.h	87
tableau.c	Fichier source contenant les fonctions relative à la gestion des tableau	89
tableau.h	Fichier header contenant les prototypes des fonctions relative à la gestion des tableau	95

Chapitre 3

Documentation des structures de données

3.1 Référence de la structure Carte

represente une carte d'un jeu de 32 cartes

```
#include <main.h>
```

Graphe de collaboration de Carte:

Champs de données

- Couleur couleur
- Valeur valeur

3.1.1 Description détaillée

represente une carte d'un jeu de 32 cartes

Définition à la ligne 133 du fichier main.h.

3.1.2 Documentation des champs

3.1.2.1 couleur

Couleur couleur

Définition à la ligne 135 du fichier main.h.

3.1.2.2 valeur

`Valeur` valeur

type Couleur, donne la famille Ã laquel la carte appartient

Définition à la ligne 136 du fichier main.h.

La documentation de cette structure a été générée à partir du fichier suivant :

— [main.h](#)

3.2 Référence de la structure Contrat

represente un contrat Ã la belote coichÃ©e

```
#include <main.h>
```

Graphe de collaboration de Contrat:

Champs de données

- [Joueur preneur](#)
- [NbPoint nbPoint](#)
- [Couleur atout](#)
- [Coinche coinche](#)

3.2.1 Description détaillée

represente un contrat Ã la belote coichÃ©e

Permet d'avoir toutes les information concernant un contrat stockÃ© dans une seuil et mÃame variable

Définition à la ligne 148 du fichier main.h.

3.2.2 Documentation des champs

3.2.2.1 atout

`Couleur` atout

type NbPoint, nombre de point du contrat

Définition à la ligne 152 du fichier main.h.

3.2.2.2 coinche

`Coinche` coinche

type Couleur, couleur de l'atout qui Ã Ã©tÃ© prise pour le contrat

Définition à la ligne 153 du fichier main.h.

3.2.2.3 nbPoint

`NbPoint` nbPoint

type Joueur, joueur qui a pris le contrat

Définition à la ligne 151 du fichier main.h.

3.2.2.4 preneur

`Joueur` preneur

Définition à la ligne 150 du fichier main.h.

La documentation de cette structure a été générée à partir du fichier suivant :

— [main.h](#)

Chapitre 4

Documentation des fichiers

4.1 Référence du fichier acquisition.c

fichier source contenant les fonctions relative a l'acquisition

```
#include "main.h"
```

Graphe des dépendances par inclusion de acquisition.c:

4.2 Référence du fichier acquisition.h

fichier header contenant les prototype des fonctions relative a l'acquisition

Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :

Fonctions

- int [acquisitionEntierAvecMessage](#) (int min, int max)
renvois au programme un entier saisi par l'utilisateur avec message d'erreur entre 2 entiers
- int [acquisitionEntierSansMessage](#) (int min, int max)
acquisition d'un entier saisi par l'utilisateur sans message d'erreur entre 2 entiers
- char [acquisitionOuiNonSansMessage](#) ()
acquisition O/N 'oui ou non' saisi par l'utilisateur sans message d'erreur
- int [acquisitionEntierSansMessageAvecConsigne](#) (int min, int max, char consigne[])
acquisition d'un entier saisi par l'utilisateur sans message d'erreur entre 2 entiers avec une consigne paramétrable
- void [acquisitionPseudoAvecMessage](#) (char *pointeurPseudo, char instruction[], int type)
acquisition d'un pseudo saisi par l'utilisateur avec message d'erreur
- int [acquisitionEntierSecurise](#) ()
fonction qui fait l'acquisition sécurisée d'un nombre

4.2.1 Description détaillée

fichier header contenant les prototype des fonctions relative a l'acquisition

Auteur

Carlo.A & Florian.C

Version

v1.0

Date

12 juin 2020

4.2.2 Documentation des fonctions

4.2.2.1 acquisitionEntierAvecMessage()

```
int acquisitionEntierAvecMessage (
    int min,
    int max )
```

renvois au programme un entier saisi par l'utilisateur avec message d'erreur entre 2 entiers

Paramètres

<i>min</i>	: borne inférieur de l'ensemble dans lequel la valeur saisi doit ce trouver
<i>max</i>	: borne superieur de l'ensemble dans lequel la valeur saisi doit ce trouver

Renvoie

la valeur saise par l'utilisateur et vérifier

Définition à la ligne 11 du fichier acquisition.c.

Voici le graphe d'appel pour cette fonction :

4.2.2.2 acquisitionEntierSansMessage()

```
int acquisitionEntierSansMessage (
    int min,
    int max )
```

accision d'un entier saisi par l'utilisateur sans message d'erreur entre 2 entiers

Paramètres

<i>min</i>	: borne inférieur de l'ensemble dans lequel la valeur saisi doit ce trouver
<i>max</i>	: borne superieur de l'ensemble dans lequel la valeur saisi doit ce trouver

Renvoie

la valeure entiere saise par l'utilisateur et vérifier

Définition à la ligne 24 du fichier acquisition.c.

Voici le graphe d'appel pour cette fonction :

4.2.2.3 acquisitionEntierSansMessageAvecConsigne()

```
int acquisitionEntierSansMessageAvecConsigne (
    int min,
```

```
int max,
char consigne[ ] )
```

acquisition d'un entier saisi par l'utilisateur sans message d'erreur entre 2 entiers avec une consigne paramétrable

Paramètres

<i>min</i>	: borne inférieure de l'ensemble dans lequel la valeur saisi doit se trouver
<i>max</i>	: borne supérieure de l'ensemble dans lequel la valeur saisi doit se trouver
<i>consigne</i>	: message affiché avant à l'utilisateur pour lui indiquer ce qu'il doit saisir

Renvoie

la valeur entière saisie par l'utilisateur et vérifiée

Définition à la ligne 42 du fichier acquisition.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.2.2.4 acquisitionEntierSecurise()

```
int acquisitionEntierSecurise ( )
```

fonction qui fait l'acquisition sécurisée d'un nombre

Renvoie

renvoie le nombre saisi par l'utilisateur et -1 si aucun nombre a été saisi

Définition à la ligne 94 du fichier acquisition.c.

Voici le graphe des appelants de cette fonction :

4.2.2.5 acquisitionOuiNonSansMessage()

```
char acquisitionOuiNonSansMessage ( )
```

acquisition O/N 'oui ou non' saisi par l'utilisateur sans message d'erreur

Renvoie

le caractère saisi par l'utilisateur, 'O' ou 'N' qui a été vérifié

Définition à la ligne 33 du fichier acquisition.c.

4.2.2.6 acquisitionPseudoAvecMessage()

```
void acquisitionPseudoAvecMessage (
    char * pointeurPseudo,
    char instruction[],
    int type )
```

acquisition d'un pseudo saisi par l'utilisateur avec message d'erreur

Paramètres

<i>PPseudo</i>	pointeur renvoyant vers le premier caractère du pseudo à modifier
<i>instruction</i>	phrase à afficher pour donner l'instruction à l'utilisateur
<i>type</i>	: vaut 1 si la fonction est appelée dans la fonction paramètre

Renvoie

void

Définition à la ligne 52 du fichier acquisition.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.3 Référence du fichier affichage.c

fichier source contenant les fonctions relative à l'affichage

```
#include "main.h"
```

Graphe des dépendances par inclusion de affichage.c:

Fonctions

- int [afficheMenuPrincipal](#) (int type)
affiche le logo du jeu puis le menu principal
- int [afficheInterfacePli](#) ([Carte](#) dernierPli[], [Carte](#) pli[], char *pseudo[], [Carte](#) cartesEnMain[], [Contrat](#) contratActuel, char message[], [Joueur](#) utilisateur, [Joueur](#) dernierVainqueur, int score[], int pointManche[], int type)
affiche l'interface de jeu durant un pli
- void [modifieTailleFenetre](#) (int nbLigneFenetre, int nbColloneFentre)
modifie la taille de la fenetre dans lequel le programme s'execute
- void [afficheSousMenus](#) (char phrase[], char intitule[])
affiche les différents sous menus
- void [afficheMain](#) ([Carte](#) carte[])
affiche les carte qu'un joueur a en main
- void [afficheContrat](#) ([Contrat](#) contrat, char *pseudo[], int version)
affiche un contrat
- void [afficheMenuSelection](#) (char intitule[], char phrase[], int sautDeLigne)
affiche dans un cadre plusieurs chaines de caractères avec un certain nombre de lignes sautée entre chacunes d'elles
- [Contrat](#) [proposeContratUtilisateur](#) ([Contrat](#) dernierContrat, [Joueur](#) parle, [Carte](#) *pCarteMain)
gère le choix du contrat par l'utilisateur

4.3.1 Description détaillée

fichier source contenant les fonctions relative à l'affichage

Auteur

Carlo.A & Florian.C

Version

v1.0

Date

12 juin 2020

4.3.2 Documentation des fonctions

4.3.2.1 afficheContrat()

```
void afficheContrat (
    Contrat contrat,
    char * pseudo[],
    int version )
```

affiche un contrat

Paramètres

<i>Contrat</i>	contrat : contrat a afficher
<i>char</i>	*pseudo[]: tableau de pointeur contenant les pseudo des différents joueurs
<i>int</i>	version : 1 pour la version belle et de grande taille et 2 pour la version courte

Renvoie

void

Définition à la ligne 247 du fichier affichage.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.3.2.2 afficheInterfacePli()

```
afficheInterfacePli (
    Carte dernierPli[],
    Carte pli[],
    char * pseudo[],
    Carte cartesEnMain[],
    Contrat contratActuel,
    char message[],
    Joueur utilisateur,
    Joueur dernierVainqueur,
    int score[],
    int pointManche[],
    int type )
```

affiche l'interface de jeu durant un pli

Paramètres

<i>Carte</i>	dernierPli[] : tableau contenant les 4 cartes du dernier plis
<i>Carte</i>	pli[] : tableau contenant les cartes du pli en cours, mettre 0, 0 si il n'y a pas de carte
<i>char</i>	*pseudo[] : tableau contenant les pseudo des 4 joueur
<i>Carte</i>	cartesEnMain[] : tableau contenant les carte dans la main de l'utilisateur
<i>Contrat</i>	contratActuel : contrat qui est en cour dans cette manche

Paramètres

<i>char</i>	message[500]
<i>Joueur</i>	utilisateur : donne la position de l'utilisateur, mettre SANS_Joueur pour faire une partie avec uniquement des ordinateur
<i>Joueur</i>	dernierVainqueur : joueur ayant gagné le dernier pli
<i>int</i>	score[] : tableau qui contient les score de la partie
<i>int</i>	pointManche[] : tableau qui contient les points de la manche
<i>int</i>	type : version de la fonction : 0 pour un affichage et une acquisition et 1 pour l'acquisition seule

Renvoie

int : valeur choisie par l'utilisateur

< Formatage des chaîne de caractère relative aux pseudo

< Formatage des chaîne de caractère relative au message

< Formatage des chaîne de caractère relative aux cartes

< Formatage des chaîne de caractère relative au contrat

Définition à la ligne 59 du fichier affichage.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.3.2.3 afficheMain()

```
void afficheMain (
    Carte carte[] )
```

affiche les carte qu'un joueur a en main

Paramètres

<i>Carte</i>	carte[] : tableau contenant les carte
--------------	---------------------------------------

Renvoie

void

Définition à la ligne 227 du fichier affichage.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.3.2.4 afficheMenuPrincipal()

```
int afficheMenuPrincipal (
    int type )
```

affiche le logo du jeu puis le menu principal

Paramètres

<i>type</i>	quel type de menu est souhaité
-------------	--------------------------------

Renvoie

la valeur de l'action que l'utilisateur decide de faire

Définition à la ligne 11 du fichier `affichage.c`.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.3.2.5 `afficheMenuSelection()`

```
void afficheMenuSelection (
    char intitule[],
    char phrase[],
    int sautDeLigne )
```

affiche dans un cadre plusieurs chaines de caractères avec un certain nombre de lignes sautée entre chacunes d'elles

Paramètres

<i>char</i>	<code>intitule []</code> : titre du du cadre
<i>char</i>	<code>phrase[]</code> : les chaines de caractère séparés par des points virgules
<i>int</i>	<code>sautDeLigne</code> : nombre de lignes à sauter entre chaque chaine de caractères

Renvoie

void

< contrôle le centrage du titre du sous-menu

< effacement de la chaine de caractères

< efface l'écran et affiche le sous-menu avec le texte correspondant

Définition à la ligne 269 du fichier `affichage.c`.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.3.2.6 `afficheSousMenus()`

```
void afficheSousMenus (
    char phrase[],
    char intitule[] )
```

affiche les différents sous menus

Paramètres

<i>phrase</i>	phrase correspondant au sous menu choisi
---------------	--

Renvoie

void

< contrôle le centrage du titre du sous-menu

< découpe la phrase pour pouvoir l'afficher

< efface l'écran et affiche le sous-menu avec le texte correspondant

Définition à la ligne 186 du fichier `affichage.c`.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.3.2.7 modifieTailleFenetre()

```
void modifieTailleFenetre (
    int  nbLigneFenetre,
    int  nbColloneFentre )
```

modifie la taille de la fenetre dans lequel le programme s'exécute

Paramètres

<i>nbLingeFenetre</i>	: nombre de ligne de la fenetre
<i>nbColloneFenetre</i>	: nombre de collone de la fenetre

Renvoie

void

Définition à la ligne 177 du fichier `affichage.c`.

Voici le graphe des appelants de cette fonction :

4.3.2.8 proposeContratUtilisateur()

```
Contrat proposeContratUtilisateur (
    Contrat dernierContrat,
    Joueur parle,
    Carte * pCarteMain )
```

gère le choix du contrat par l'utilisateur

Paramètres

<i>Contrat</i>	dernierContrat : dernier contrat proposé
<i>Joueur</i>	parle : joueur qui parle
<i>Carte</i>	*pCarteMain : pointeur sur le tableau qui stocke les carte dans la main du joueur

Renvoi

Contrat : nouveau contrat proposé par le joueur

Paramètres

--	--

Définition à la ligne 348 du fichier affichage.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.4 Référence du fichier affichage.h

fichier header contenant les prototypes des fonctions relative à l'affichage

Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :

Fonctions

- int *afficheMenuPrincipal* (int type)
affiche le logo du jeu puis le menu principal
- int *afficheInterfacePli* (*Carte* dernierPli[], *Carte* pli[], char *pseudo[], *Carte* cartesEnMain[], *Contrat* contratActuel, char message[], *Joueur* utilisateur, *Joueur* dernierVainqueur, int score[], int pointManche[], int type)
affiche l'interface de jeu durant un pli
- void *modifieTailleFenetre* (int nbLigneFenetre, int nbColloneFentre)
modifie la taille de la fenetre dans laquelle le programme s'exécute
- void *afficheSousMenus* (char phrase[], char intitule[])
affiche les différents sous menus
- void *afficheMain* (*Carte* carte[])
affiche les carte qu'un joueur a en main
- void *afficheContrat* (*Contrat* contrat, char *pseudo[], int version)
affiche un contrat
- void *afficheMenuSelection* (char intitule[], char phrase[], int sautDeLigne)
affiche dans un cadre plusieurs chaines de caractères avec un certain nombre de lignes sautée entre chacunes d'elles
- *Contrat* *proposeContratUtilisateur* (*Contrat* dernierContrat, *Joueur* parle, *Carte* *pCarteMain)
gère le choix du contrat par l'utilisateur

4.4.1 Description détaillée

fichier header contenant les prototypes des fonctions relative à l'affichage

Auteur

Carlo.A & Florian.C

Version

v1.0

Date

12 juin 2020

4.4.2 Documentation des fonctions

4.4.2.1 afficheContrat()

```
void afficheContrat (
    Contrat contrat,
    char * pseudo[],
    int version )
```

affiche un contrat

Paramètres

<i>Contrat</i>	contrat : contrat a afficher
<i>char</i>	*pseudo[]: tableau de pointeur contenant les pseudo des différents joueurs
<i>int</i>	version : 1 pour la version belle et de grande taille et 2 pour la version courte

Renvoie

void

Définition à la ligne 247 du fichier affichage.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.4.2.2 afficheInterfacePli()

```
int afficheInterfacePli (
    Carte dernierPli[],
    Carte pli[],
    char * pseudo[],
    Carte cartesEnMain[],
    Contrat contratActuel,
    char message[],
    Joueur utilisateur,
    Joueur dernierVainqueur,
    int score[],
    int pointManche[],
    int type )
```

affiche l'interface de jeu durant un pli

Paramètres

<i>Carte</i>	dernierPli[] : tableau contenant les 4 cartes du dernier plis
<i>Carte</i>	pli[] : tableau contenant les cartes du pli en cours, mettre 0, 0 si il n'y a pas de carte
<i>char</i>	*pseudo[] : tableau contenant les pseudo des 4 joueur
<i>Carte</i>	cartesEnMain[] : tableau contenant les carte dans la main de l'utilisateur
<i>Contrat</i>	contratActuel : contrat qui est en cour dans cette manche
<i>char</i>	message[500]
<i>Joueur</i>	utilisateur : donne la position de l'utilisateur, mettre SANS_Joueur pour faire une partie avec uniquement des ordinateur
<i>Joueur</i>	dernierVainqueur : joueur ayant gagné le dernier pli
<i>int</i>	score[] : tableau qui contient les score de la partie
<i>int</i>	pointManche[] : tableau qui contient les points de la manche
<i>int</i>	type : version de la fonction : 0 pour un affichage et une acquisition et 1 pour l'acquisition seule

Renvoi

int : valeur choisie par l'utilisateur

< Formatage des chaine de caractère relative aux pseudo

< Formatage des chaine de caractère relative au message

< Formatage des chaine de caractère relative aux cartes

< Formatage des chaine de caractère relative au contrat

Définition à la ligne 59 du fichier affichage.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.4.2.3 afficheMain()

```
void afficheMain (
    Carte carte[] )
```

affiche les carte qu'un joueur a en main

Paramètres

<i>Carte</i>	carte[] : tableau contenant les carte
--------------	---------------------------------------

Renvoi

void

Définition à la ligne 227 du fichier affichage.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.4.2.4 afficheMenuPrincipal()

```
int afficheMenuPrincipal (
    int type )
```

affiche le logo du jeu puis le menu principal

Paramètres

<i>type</i>	quel type de menu est souhaité
-------------	--------------------------------

Renvoie

la valeur de l'action que l'utilisateur decide de faire

Définition à la ligne 11 du fichier affichage.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.4.2.5 afficheMenuSelection()

```
void afficheMenuSelection (
    char intitule[],
    char phrase[],
    int sautDeLigne )
```

affiche dans un cadre plusieurs chaines de caractères avec un certain nombre de lignes sautée entre chacunes d'elles

Paramètres

<i>char</i>	intitule []: titre du du cadre
<i>char</i>	phrase[] : les chaines de caractère séparés par des points virgules
<i>int</i>	sautDeLigne : nombre de lignes à sauter entre chaque chaine de caractères

Renvoie

void

< contrôle le centrage du titre du sous-menu

< effacement de la chaine de carrateres

<efface l'écran et affiche le sous-menu avec le texte correspondant

Définition à la ligne 269 du fichier affichage.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.4.2.6 afficheSousMenus()

```
void afficheSousMenus (
    char phrase[],
    char intitule[] )
```

affiche les différents sous menus

Paramètres

<i>phrase</i>	phrase correspondant au sous menu choisi
---------------	--

Renvoie

`void`

< contrôle le centrage du titre du sous-menu

< découpe la phrase pour pouvoir l'afficher

< efface l'écran et affiche le sous-menu avec le texte correspondant

Définition à la ligne 186 du fichier `affichage.c`.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.4.2.7 `modifieTailleFenetre()`

```
void modifieTailleFenetre (
    int  nbLigneFenetre,
    int  nbColloneFentre )
```

modifie la taille de la fenetre dans lequel le programme s'exécute

Paramètres

<i>nbLingeFenetre</i>	: nombre de ligne de la fenetre
<i>nbColloneFenetre</i>	: nombre de collone de la fenetre

Renvoie

`void`

Définition à la ligne 177 du fichier `affichage.c`.

Voici le graphe des appelants de cette fonction :

4.4.2.8 `proposeContratUtilisateur()`

```
Contrat proposeContratUtilisateur (
    Contrat dernierContrat,
    Joueur parle,
    Carte * pCarteMain )
```

gère le choix du contrat par l'utilisateur

Paramètres

<i>Contrat</i>	dernierContrat : dernier contrat proposé
<i>Joueur</i>	parle : joueur qui parle
<i>Carte</i>	*pCarteMain : pointeur sur le tableau qui stocke les carte dans la main du joueur

Renvoi

Contrat : nouveau contrat proposé par le joueur

Paramètres

--	--

Définition à la ligne 348 du fichier `affichage.c`.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.5 Référence du fichier `autre.c`

fichier source contenant les fonctions qui n'allait dans aucun autre fichier

```
#include "main.h"
```

Graphe des dépendances par inclusion de `autre.c`:

Fonctions

- *Joueur* *joueurSuivant* (*Joueur* joueur)
passe au joueur suivant
- int *nbAleatoire* (int *mini*, int *maxi*)
renvoie in nombre aléatoire entre deux bornes
- void *setContrat* (*Contrat* *contrat, *Joueur* preneur, int nbPoint, *Couleur* atout, *Coinche* coinche)
définit un contrat
- int *pointPli* (*Carte* pli[], *Couleur* atout, int nbCarte)
compte le nombre de points d'un pli
- float *ajusteEchelle* (float valeur, float entreMin, float entreMax, float sortieMin, float sortieMax)
met a l'echelle la valeur d'une variable comprise entre deux bornes pour que la variable de retour soit compris entre deux autres bornes
- void *joue1000Partie* (int nbPartie)
joue n partie aves uniquement des ia pour controler si tout ce passe bien et reueillir des statistique

4.5.1 Description détaillée

fichier source contenant les fonctions qui n'allait dans aucun autre fichier

Auteur

Carlo.A & Florian.C

Version

v1.0

Date

12 juin 2020

4.5.2 Documentation des fonctions

4.5.2.1 ajusteEchelle()

```
float ajusteEchelle (
    float valeur,
    float entreMin,
    float entreMax,
    float sortieMin,
    float sortieMax )
```

met a l'echelle la valeur d'une variable comprise entre deux bornes pour que la variable de retour soit compris entre deux autres bornes

Paramètres

<i>float</i>	valeur : valeur a mettre à l'echelle
<i>float</i>	entreMin : valeur minimum que peut prendre l'entrée
<i>float</i>	entreMax : valeur maximum que peut prendre l'entrée
<i>float</i>	sortieMin : valeur minimum que peut prendre la sortie
<i>float</i>	sortieMax : valeur maximum que peut prendre la sortie

Renvoie

float : la valeur de valeur une fois mise a l'echelle

Définition à la ligne 147 du fichier autre.c.

Voici le graphe des appelants de cette fonction :

4.5.2.2 joue1000Partie()

```
void joue1000Partie (
    int nbPartie )
```

joue n partie avec uniquement des ia pour contrôler si tout se passe bien et recueillir des statistiques

Paramètres

<i>int</i>	nbPartie nombre de partie que l'on veut jouer
------------	---

Renvoie

voir

Définition à la ligne 153 du fichier autre.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.5.2.3 joueurSuivant()

```
Joueur joueurSuivant (
    Joueur joueur )
```

passé au joueur suivant

Paramètres

<i>Joueur</i>	joueur : joueur actuel
---------------	------------------------

Renvoie

Joueur : le joueur suivant

Définition à la ligne 11 du fichier autre.c.

Voici le graphe des appelants de cette fonction :

4.5.2.4 nbAleatoire()

```
int nbAleatoire (
    int mini,
    int maxi )
```

renvoie un nombre aléatoire entre deux bornes

Paramètres

<i>int</i>	mini : borne inférieur
<i>int</i>	maxi : borne supérieur

Renvoie

int : le nombre aléatoire

Définition à la ligne 21 du fichier autre.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.5.2.5 pointPli()

```
int pointPli (
    Carte pli[],
    Couleur atout,
    int nbCarte )
```

compte le nombre de points d'un pli

Paramètres

<i>Carte</i>	pli []: tableau des cartes possédées dont on doit faire la somme des points
<i>Couleur</i>	atout: couleur de l'atout joué dans la manche
<i>nbCarte</i>	nombre de carte à compter

Renvoie

points

Définition à la ligne 34 du fichier autre.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.5.2.6 setContrat()

```
void setContrat (
    Contrat * contrat,
    Joueur preneur,
    int nbPoint,
    Couleur atout,
    Coinche coinche )
```

définit un contrat

Paramètres

<i>Contat</i>	*contrat : pointeur sur le contrat a modifier
<i>Joueur</i>	preneur : preneur du contrat
<i>int</i>	nbPoint : nombre de point
<i>Couleur</i>	atout : couleur de l'aout
<i>Coinche</i>	coinche : normal, coinché , surcoinché

Renvoie

void

Définition à la ligne 26 du fichier autre.c.

Voici le graphe des appelants de cette fonction :

4.6 Référence du fichier autre.h

fichier header contenant les prototype des fonctions qui n'allait dans aucun autre fichier

Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :

Fonctions

- `Joueur joueurSuivant` (`Joueur` joueur)
passse au joueur suivant
- `int nbAleatoire` (`int mini`, `int maxi`)
renvoie in nombre aléatoire entre deux bornes
- `void setContrat` (`Contrat *contrat`, `Joueur` preneur, `int nbPoint`, `Couleur` atout, `Coinche` coinche)
définit un contrat
- `int pointPli` (`Carte pli[]`, `Couleur` atout, `int nbCarte`)
compte le nombre de points d'un pli
- `float ajusteEchelle` (`float valeur`, `float entreMin`, `float entreMax`, `float sortieMin`, `float sortieMax`)
met a l'echelle la valeur d'une variable comprise entre deux bornes pour que la variable de retour soit compris entre deux autres bornes
- `void joue1000Partie` (`int nbPartie`)
joue n partie aves uniquement des ia pour controler si tout ce passe bien et reuecillir des statistique

4.6.1 Description détaillée

ficher header contenant les prototype des fonctions qui n'allait dans aucun autre fichier

Auteur

Carlo.A & Florian.C

Version

v1.0

Date

12 juin 2020

4.6.2 Documentation des fonctions

4.6.2.1 `ajusteEchelle()`

```
float ajusteEchelle (
    float valeur,
    float entreMin,
    float entreMax,
    float sortieMin,
    float sortieMax )
```

met a l'echelle la valeur d'une variable comprise entre deux bornes pour que la variable de retour soit compris entre deux autres bornes

Paramètres

<i>float</i>	valeur : valeur a mettre à l'echelle
<i>float</i>	entreMin : valeur minimum que peut prendre l'entrée
<i>float</i>	entreMax : valeur maximum que peut prendre l'entrée
<i>float</i>	sortieMin : valeur minimum que peut prendre la sortie
<i>float</i>	sortieMax : valeur maximum que peut prendre la sortie

Renvoie

float : la valeur de valeur une fois mise a l'echelle

Définition à la ligne 147 du fichier autre.c.

Voici le graphe des appelants de cette fonction :

4.6.2.2 joue1000Partie()

```
void joue1000Partie (
    int nbPartie )
```

joue n partie avec uniquement des ia pour contrôler si tout se passe bien et recueillir des statistiques

Paramètres

<i>int</i>	nbPartie nombre de partie que l'on veut jouer
------------	---

Renvoie

voir

Définition à la ligne 153 du fichier autre.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.6.2.3 joueurSuivant()

```
Joueur joueurSuivant (
    Joueur joueur )
```

passer au joueur suivant

Paramètres

<i>Joueur</i>	joueur : joueur actuel
---------------	------------------------

Renvoie

Joueur : le joueur suivant

Définition à la ligne 11 du fichier autre.c.

Voici le graphe des appelants de cette fonction :

4.6.2.4 nbAleatoire()

```
int nbAleatoire (
    int mini,
    int maxi )
```

renvoie un nombre aléatoire entre deux bornes

Paramètres

<i>int</i>	mini : borne inférieur
<i>int</i>	maxi : borne supérieur

Renvoie

int : le nombre aléatoire

Définition à la ligne 21 du fichier autre.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.6.2.5 pointPli()

```
int pointPli (
    Carte pli[],
    Couleur atout,
    int nbCarte )
```

compte le nombre de points d'un pli

Paramètres

<i>Carte</i>	pli []: tableau des cartes possées dont on doit faire la somme des points
<i>Couleur</i>	atout: couleur de l'atout joué dans la manche
<i>nbCarte</i>	nombre de carte à compter

Renvoie

points

Définition à la ligne 34 du fichier autre.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.6.2.6 setContrat()

```
void setContrat (
    Contrat * contrat,
    Joueur preneur,
    int nbPoint,
    Couleur atout,
    Coinche coinche )
```

définit un contrat

Paramètres

<i>Contat</i>	*contrat : pointeur sur le contrat a modifier
<i>Joueur</i>	preneur : preneur du contrat
<i>int</i>	nbPoint : nombre de point
<i>Couleur</i>	atout : couleur de l'aout
<i>Coinche</i>	coinche : normal, coinché , surcoinché

Renvoie

void

Définition à la ligne 26 du fichier autre.c.

Voici le graphe des appelants de cette fonction :

4.7 Référence du fichier formatageChaine.c

fichier source contenant les fonctions relative au formatage

```
#include "main.h"
```

Graphe des dépendances par inclusion de formatageChaine.c:

Fonctions

- int [centreChaine](#) (char chaineInitial[], char chaineFinal[], int longueurChaine)
transforme une chaine de caractère de longueur inconnue en une chaine de caractère fixé de tel sorte que le texte soit centré
- int [centreModifieChaine](#) (char chaine[], int longueurChaine)
transforme une chaine de caractère en sa version centrée
- int [decoupeChaine](#) (char chaineInitiale[], char *chaineFinale, int tailleLigne, int nbLigne)
fonction qui permet de découper une chaine de caractère en plusieurs chaine de taille fixée en coupant sur un espace
- void [rempliEspace](#) (char *chaine, int nbEspace)
rempli une chaine de caractère de nbEspace caractère espace ' '
- void [formateCarte](#) ([Carte](#) carte[], char *chaineFinale, int nbCarte, int tailleChaine, int version)
formate les chaine de caractère qui affiche la valeur et la couleur des cartes
- void [stockeInfoCarte](#) ([Carte](#) carte, char *valeur, char *couleur, int version, int tailleChaine)
met dans une chaine de caractère la valeur et la couleur d'une carte
- void [formatePseudo](#) ([Joueur](#) joueur, char *pseudo[], int tailleChaine, char chaineFinal[], int version)
stocke dans une chaine de caractère le pseudo d'un des joueur
- void [formateContrat](#) ([Contrat](#) contrat, char *chaineFinal, int tailleLigne, char *pseudo[])
stocke dans des chaine de caractère les info du contrat
- int [aligneModifieChaine](#) (char chaine[], int longueurChaine)
transforme une chaine de caractère en sa version aligné a gauche
- void [genereMessage](#) (char message[], [Joueur](#) parle, char *pseudo[], [Carte](#) carteJoue, int score, [TypeMessage](#) typeMessage)
génère un message et l'enregistre dans une chaine de caractère

4.7.1 Description détaillée

fichier source contenant les fonctions relative au formatage

Auteur

Carlo.A & Florian.C

Version

v1.0

Date

12 juin 2020

4.7.2 Documentation des fonctions

4.7.2.1 aligneModifieChaine()

```
int aligneModifieChaine (
    char chaine[],
    int longueurChaine )
```

transforme une chaine de caractère en sa version aligné a gauche

Paramètres

<i>char</i>	chaine : chaine a modifier
<i>int</i>	longueurChaine

Renvoie

int : 0 si tout c'est bien passé

Définition à la ligne 299 du fichier formatageChaine.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.7.2.2 centreChaine()

```
int centreChaine (
    char chaineInitial[],
    char chaineFinal[],
    int longueurChaine )
```

transforme une chaine de caractère de longueur inconnue en une chaine de caractère fixé de telle sorte que le texte soit centré

Paramètres

<i>char</i>	chaineInitial : chaine a modifier
<i>char</i>	chaineFinal : pointeur vers la variable qui stocke la chaîne finale
<i>int</i>	longueurChaine : longueur de la chaîne finale

Renvoie

int : 0 si tout va bien, 1 si la chaîne est initiale test plus longue que la longueur désirée

Définition à la ligne 10 du fichier formatageChaine.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.7.2.3 `centreModifieChaine()`

```
int centreModifieChaine (
    char chaine[],
    int longueurChaine )
```

transforme une chaine de caractère en sa version centrée

Paramètres

<i>char</i>	chaine : chaine a modifier
<i>int</i>	longueurChaine

Renvoie

int : 0 si tout c'est bien passé

Définition à la ligne 35 du fichier `formatageChaine.c`.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.7.2.4 `decoupeChaine()`

```
int decoupeChaine (
    char chaineInitiale[],
    char * chaineFinale,
    int tailleLigne,
    int nbLigne )
```

fonction qui permet de découper une chaine de caractère en plusieurs chaines de taille fixée en coupant sur un espace

Paramètres

<i>chaineInitiale[]</i>	: chaine a decouper
<i>chaineFinale[][]</i>	: tableau contenant les chaines une fois découpées
<i>tailleLigne</i>	: nombre de caractère maximum par ligne
<i>nbLigne</i>	: nombre maximum de lignes

Renvoie

0 si tout va bien 1 si la chaine est trop longue

Définition à la ligne 45 du fichier `formatageChaine.c`.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.7.2.5 `formateCarte()`

```
void formateCarte (
    Carte carte[],
```

```
char * chaineFinale,
int nbCarte,
int tailleChaine,
int version )
```

formate les chaine de caractère qui affiche la valeur et la couleur des cartes

Paramètres

<i>chaineFinale</i> [[]]	tableau a trois dimation qui stocke les chaine de caractère de la valeur et la couleurs des cartes
<i>carte</i> []	: tableau de type Carte contenant les carte a afficher
<i>nbCarte</i>	: nombre de carte dans le tableau <i>carte</i> []
<i>tailleChaine</i>	: taille des chaineFinale
<i>version</i>	: 0 si c'est la verion courte et 1 pour la verision longue

Renvoie

void

Définition à la ligne 101 du fichier `formatageChaine.c`.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.7.2.6 `formateContrat()`

```
void formateContrat (
    Contrat contrat,
    char * chaineFinal,
    int tailleLigne,
    char * pseudo[] )
```

stocke dans des chaine de caractère les info du contrat

Paramètres

Contrat	contrat : contrat a afficher
<i>char</i>	*chaineFinal : chaine final ou on enregistre les info du contrat
<i>int</i>	tailleLigne : taille d'une ligne
<i>char</i>	*pseudo[] : tableau qui contient les pseudo des joueur

Renvoie

void

Définition à la ligne 251 du fichier `formatageChaine.c`.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.7.2.7 formatePseudo()

```
void formatePseudo (
    Joueur joueur,
    char * pseudo[],
    int tailleChaine,
    char chaineFinal[],
    int version )
```

stocke dans une chaine de caractère le pseudo d'un des joueur

Paramètres

<i>Joueur</i>	joueur : joueur dont on veut afficher le pseudo
<i>char</i>	*pseudo[] : tableau contenant les pseudo des joueur
<i>int</i>	tailleChaine : taille de la chaine final
<i>char</i>	chaineFinal[] : chaine final ou est stocké le pseudo
<i>int</i>	version : 1 pour la version ou la chaine finale est centrée 0 pour l'aligné a gauche

Renvoie

void

Définition à la ligne 220 du fichier formatageChaine.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.7.2.8 genereMessage()

```
void genereMessage (
    char message[],
    Joueur parle,
    char * pseudo[],
    Carte carteJoue,
    int score,
    TypeMessage typeMessage )
```

génère un message et l'enregistre dans une chaine de caractère

Paramètres

<i>char</i>	message[] : tableau de char ou on enregistre le message
<i>Joueur</i>	parle : joueur qui pose une carte
<i>char</i>	*pseudo[] : tableau qui contient les pseudo des joueur
<i>Carte</i>	carteJoue : carte qui vine d'être jouée
<i>int</i>	score : score a afficher
<i>TypeMessage</i>	typeMessage : permet de savoir quelle message on veut afficher

Renvoie

void

Définition à la ligne 314 du fichier `formatageChaine.c`.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.7.2.9 `rempliEspace()`

```
void rempliEspace (
    char * chaine,
    int nbEspace )
```

rempli une chaine de caractère de nbEspace caractère espace ' '

Paramètres

<i>chaine[]</i>	: chaine a remplir;
<i>nbEspace[]</i>	: nombre d'espaca a placer dans la chaine

Renvoie

void

Définition à la ligne 93 du fichier `formatageChaine.c`.

Voici le graphe des appelants de cette fonction :

4.7.2.10 `stockeInfoCarte()`

```
void stockeInfoCarte (
    Carte carte,
    char * valeur,
    char * couleur,
    int version,
    int tailleChaine )
```

met dans une chaine de caractère la valeur et la couleur d'une carte

Paramètres

<i>carte</i>	: variable de type carte qui contient la carte a afficher
<i>valeur</i>	: pointeur vers la chaine qui stocke la valeur (ou la valeur et la couleur si mode court)
<i>couleur</i>	: pointeur vers la chaine qui stocke la couleur
<i>vesion</i>	: 0 si c'est la version courte et 1 pour la version longue
<i>tailleChaine</i>	: taille de la chaine de caratère dans laquel on écrit

Renvoie

void

Définition à la ligne 108 du fichier `formatageChaine.c`.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.8 Référence du fichier formatageChaine.h

fichier header contenant les prototypes des fonctions relative au formatage

Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :

Fonctions

- int [centreChaine](#) (char chaineInitial[], char chaineFinal[], int longueurChaine)
transforme une chaine de caractère de longueur inconnue en une chaine de caractère fixé de telle sorte que le texte soit centré
- int [centreModifieChaine](#) (char chaine[], int longueurChaine)
transforme une chaine de caractère en sa version centrée
- int [decoupeChaine](#) (char chaineInitiale[], char *chaineFinale, int tailleLigne, int nbLigne)
fonction qui permet de découper une chaine de caractère en plusieurs chaines de taille fixée en coupant sur un espace
- void [rempliEspace](#) (char *chaine, int nbEspace)
remplit une chaine de caractère de nbEspace caractères espace ' '
- void [formateCarte](#) ([Carte](#) carte[], char *chaineFinale, int nbCarte, int tailleChaine, int version)
formate les chaines de caractère qui affichent la valeur et la couleur des cartes
- void [stockeInfoCarte](#) ([Carte](#) carte, char *valeur, char *couleur, int version, int tailleChaine)
met dans une chaine de caractère la valeur et la couleur d'une carte
- void [formatePseudo](#) ([Joueur](#) joueur, char *pseudo[], int tailleChaine, char chaineFinal[], int version)
stocke dans une chaine de caractère le pseudo d'un des joueurs
- void [formateContrat](#) ([Contrat](#) contrat, char *chaineFinale, int tailleLigne, char *pseudo[])
stocke dans des chaines de caractère les infos du contrat
- int [aligneModifieChaine](#) (char chaine[], int longueurChaine)
transforme une chaine de caractère en sa version alignée à gauche
- void [genereMessage](#) (char message[], [Joueur](#) parle, char *pseudo[], [Carte](#) carteJoue, int score, [TypeMessage](#) typeMessage)
génère un message et l'enregistre dans une chaine de caractère

4.8.1 Description détaillée

fichier header contenant les prototypes des fonctions relative au formatage

Auteur

Carlo.A & Florian.C

Version

v1.0

Date

12 juin 2020

4.8.2 Documentation des fonctions

4.8.2.1 aligneModifieChaine()

```
int aligneModifieChaine (
    char chaine[],
    int longueurChaine )
```

transforme une chaine de caractère en sa version alignée à gauche

Paramètres

<i>char</i>	chaîne : chaîne a modifier
<i>int</i>	longueurChaîne

Renvoie

int : 0 si tout c'est bien passé

Définition à la ligne 299 du fichier `formatageChaîne.c`.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.8.2.2 centreChaîne()

```
int centreChaîne (
    char chaîneInitial[],
    char chaîneFinal[],
    int longueurChaîne )
```

transforme une chaîne de caractère de longueur inconnue en une chaîne de caractère fixé de telle sorte que le texte soit centré

Paramètres

<i>char</i>	chaîneInitial : chaîne a modifier
<i>char</i>	chaîneFinal : pointeur vers la variable qui stocke la chaîne finale
<i>int</i>	longueurChaîne : longueur de la chaîne finale

Renvoie

int : 0 si tout va bien, 1 si la chaîne est initiale test plus longue que la longueur désirée

Définition à la ligne 10 du fichier `formatageChaîne.c`.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.8.2.3 centreModifieChaîne()

```
int centreModifieChaîne (
    char chaîne[],
    int longueurChaîne )
```

transforme une chaîne de caractère en sa version centrée

Paramètres

<i>char</i>	chaîne : chaîne a modifier
<i>int</i>	longueurChaîne

Renvoie

int : 0 si tout c'est bien passé

Définition à la ligne 35 du fichier `formatageChaine.c`.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.8.2.4 decoupeChaine()

```
int decoupeChaine (
    char chaineInitiale[],
    char * chaineFinale,
    int tailleLigne,
    int nbLigne )
```

fonction qui permet de découper une chaîne de caractère en plusieurs chaînes de taille fixée en coupant sur un espace

Paramètres

<i>chaineInitiale[]</i>	: chaîne à découper
<i>chaineFinale[][]</i>	: tableau contenant les chaînes une fois découpées
<i>tailleLigne</i>	: nombre de caractères maximum par ligne
<i>nbLigne</i>	: nombre maximum de lignes

Renvoie

0 si tout va bien 1 si la chaîne est trop longue

Définition à la ligne 45 du fichier `formatageChaine.c`.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.8.2.5 formateCarte()

```
void formateCarte (
    Carte carte[],
    char * chaineFinale,
    int nbCarte,
    int tailleChaine,
    int version )
```

formate les chaînes de caractère qui affichent la valeur et la couleur des cartes

Paramètres

<i>chaineFinale[][][]</i>	tableau à trois dimensions qui stocke les chaînes de caractère de la valeur et la couleur des cartes
<i>carte[]</i>	: tableau de type <code>Carte</code> contenant les cartes à afficher
<i>nbCarte</i>	: nombre de cartes dans le tableau <code>carte[]</code>
<i>tailleChaine</i>	: taille des chaînes <code>chaineFinale</code>
<i>version</i>	: 0 si c'est la version courte et 1 pour la version longue

Renvoie

void

Définition à la ligne 101 du fichier `formatageChaine.c`.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.8.2.6 formateContrat()

```
void formateContrat (
    Contrat contrat,
    char * chaineFinal,
    int tailleLigne,
    char * pseudo[] )
```

stocke dans des chaine de caractère les info du contrat

Paramètres

<i>Contrat</i>	contrat : contrat a afficher
<i>char</i>	*chaineFinal : chaine final ou on enregistre les info du contrat
<i>int</i>	tailleLigne : taille d'une ligne
<i>char</i>	*pseudo[] : tableau qui contient les pseudo des joueur

Renvoie

void

Définition à la ligne 251 du fichier `formatageChaine.c`.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.8.2.7 formatePseudo()

```
void formatePseudo (
    Joueur joueur,
    char * pseudo[],
    int tailleChaine,
    char chaineFinal[],
    int version )
```

stocke dans une chaine de caractère le pseudo d'un des joueur

Paramètres

<i>Joueur</i>	joueur : joueur dont on veut afficher le pseudo
<i>char</i>	*pseudo[] : tableau contenant les pseudo des joueur
<i>int</i>	tailleChaine : taille de la chaine final
<i>char</i>	chaineFinal[] : chaine final ou est stocké le pseudo
<i>int</i>	version : 1 pour la version ou la chaine finale est centrée 0 pour l'aligné a gauche

Renvoie`void`

Définition à la ligne 220 du fichier `formatageChaine.c`.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.8.2.8 `genereMessage()`

```
void genereMessage (
    char message[],
    Joueur parle,
    char * pseudo[],
    Carte carteJoue,
    int score,
    TypeMessage typeMessage )
```

génère un message et l'enregistre dans une chaîne de caractère

Paramètres

<i>char</i>	<code>message[]</code> : tableau de char ou on enregistre le message
<i>Joueur</i>	<code>parle</code> : joueur qui pose une carte
<i>char</i>	<code>*pseudo[]</code> : tableau qui contient les pseudo des joueur
<i>Carte</i>	<code>carteJoue</code> : carte qui vine d'être jouée
<i>int</i>	<code>score</code> : score a afficher
<i>TypeMessage</i>	<code>typeMessage</code> : permet de savoir quelle message on veut afficher

Renvoie`void`

Définition à la ligne 314 du fichier `formatageChaine.c`.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.8.2.9 `rempliEspace()`

```
void rempliEspace (
    char * chaine,
    int nbEspace )
```

rempli une chaîne de caractère de `nbEspace` caractère espace ' '

Paramètres

<i>chaine[]</i>	: chaîne a remplir;
<i>nbEspace[]</i>	: nombre d'espaca a placer dans la chaîne

Renvoie

void

Définition à la ligne 93 du fichier `formatageChaine.c`.

Voici le graphe des appelants de cette fonction :

4.8.2.10 stockeInfoCarte()

```
void stockeInfoCarte (
    Carte carte,
    char * valeur,
    char * couleur,
    int version,
    int tailleChaine )
```

met dans une chaine de caractère la valeur et la couleur d'une carte

Paramètres

<i>carte</i>	: variable de type carte qui contient la carte a afficher
<i>valeur</i>	: pointeur vers la chaine qui stocke la valeur (ou la valeur et la couleur si mode court)
<i>couleur</i>	: pointeur vers la chaine qui stocke la couleur
<i>vesion</i>	: 0 si c'est la version courte et 1 pour la version longue
<i>tailleChaine</i>	: taille de la chaine de caratère dans laquel on écrit

Renvoie

void

Définition à la ligne 108 du fichier `formatageChaine.c`.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.9 Référence du fichier general.c

fichier source contenant les fonctions les plus impotantes pour le jeu de la belote

```
#include "main.h"
```

Graphe des dépendances par inclusion de `general.c`:

Fonctions

- void `initialisation` (int nbLigneFenetre, int nbColloneFenetre)
fonction qui gère l'initialitation et le lancement du programme
- void `menuPrincipal` ()
fonction qui gère le debut de la partie
- int `nouvellePartie` (char *pseudo[], `Joueur` utilisateur, int *pStatistique, int infoEcritureFicher[])
lancement d'une nouvelle partie
- char `manche` (char *pseudo[], int score[], `Joueur` dealer, `Joueur` utilisateur, int *pStatistique)
lancement d'une manche

- `Contrat annonceContrat` (`char *pseudo[]`, `Joueur dealer`, `Carte *pCarteMain`, `Joueur utilisateur`)
gère l'annonce et la surenchère des contrats par les joueurs
- `Contrat proposeContrat` (`Contrat dernierContrat`, `Joueur parle`, `char *pseudo[]`, `Carte *pCarteMain`, `Joueur utilisateur`)
proposition d'un contrat par un joueur
- `Joueur pli` (`Contrat contrat`, `Joueur premierAJouer`, `char *pseudo[]`, `Carte *pCarteMain`, `int pointManche[]`, `int pointAnonce[]`, `char belote[]`, `Carte cartePli[]`, `Carte carteAncienPli[]`, `int score[]`, `Joueur utilisateur`, `int numPli`)
fonction qui gère un pli
- `int poseCarte` (`Joueur joueur`, `int numCarte`, `Carte *pMainJoueurs`, `Carte pli[]`, `int carteRestante`)
fonction qui vérifie la validité d'une carte est la pose
- `char calculPointManche` (`Contrat contrat`, `int pointManche[]`, `int pointAnonce[]`, `int pointBelote[]`, `int score[]`, `char *pseudo[]`, `Joueur utilisateur`)
compte les points gagnés par chaque joueur au cours de la manche et fonction du contrat et ajoute ces points au tableau qui contient tous les scores de la partie

4.9.1 Description détaillée

fichier source contenant les fonctions les plus importantes pour le jeu de la belote

Auteur

Carlo.A & Florian.C

Version

v1.0

Date

12 juin 2020

4.9.2 Documentation des fonctions

4.9.2.1 annonceContrat()

```
Contrat annonceContrat (
    char * pseudo[],
    Joueur dealer,
    Carte * pCarteMain,
    Joueur utilisateur )
```

gère l'annonce et la surenchère des contrats par les joueurs

Paramètres

<code>char</code>	<code>*pseudo[]</code> : tableau de pointeurs contenant les pseudos des différents joueurs
<code>Joueur</code>	<code>dealer</code> : joueur qui distribue les cartes
<code>Carte</code>	<code>*pCarteMain</code> : pointeur sur le tableau qui stocke les cartes dans la main de chaque joueur
<code>Joueur</code>	<code>utilisateur</code> : donne la position de l'utilisateur, mettre <code>SANS_Joueur</code> pour faire une partie avec uniquement des ordinateurs

Renvoie

Contrat : contrat final qui a été choisi pour la partie

< Affichage du choix du joueur

Définition à la ligne 287 du fichier `general.c`.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.9.2.2 calculPointManche()

```
char calculPointManche (
    Contrat contrat,
    int pointManche[],
    int pointAnonce[],
    int pointBelote[],
    int score[],
    char * pseudo[],
    Joueur utilisateur )
```

compte les points gagnés par chaque joueur au cours de la manche en fonction du contrat et ajoute ces points au tableau qui contient tous les scores de la partie

Paramètres

Contrat	contrat : contrat qui a été choisi durant la manche
int	pointManche[] : tableau qui contient les points de la manche
int	pointAnonce[] : tableau qui contient les points d'annonce de la manche
int	pointBelote[] : tableau contenant les points de belote rebelote de chaque joueur
int	score[] : tableau qui contient les scores de la partie
char	*pseudo[] : tableau de pointeur contenant les pseudos des différents joueurs
Joueur	utilisateur : donne la position de l'utilisateur, mettre SANS_Joueur pour faire une partie avec uniquement des ordinateurs

Renvoie

char : 1 si le contrat a été réussi et 0 si le contrat a échoué

< On vérifie si le contrat est rempli

< cas où un des joueurs a pris un capot

< cas où un des joueurs a pris une défaule

< Le contrat est rempli si les points de l'équipe est supérieur à ce qui est annoncé et si il y a 82 points sans les annonces

< on affiche ce qui s'est passé

Définition à la ligne 476 du fichier `general.c`.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.9.2.3 initialisation()

```
void initialisation (
    int nbLigneFenetre,
    int nbColloneFenetre )
```

fonction qui gère l'initialisation et le lancement du programme

Paramètres

<i>nbLigneFenetre</i>	: taille verticale de la fenêtre
<i>nbColloneFenetre</i>	: taille horizontale de la fenêtre

Renvoie

void

Définition à la ligne 11 du fichier general.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.9.2.4 manche()

```
char manche (
    char * pseudo[],
    int score[],
    Joueur dealer,
    Joueur utilisateur,
    int * pStatistique )
```

lancement d'une manche

Paramètres

<i>char</i>	*pseudo[] : tableau de pointeurs contenant les pseudos des différents joueurs
<i>int</i>	score[] : tableau contenant les scores des joueurs
<i>Joueur</i>	dealer : joueur qui distribue les cartes
<i>Joueur</i>	utilisateur : donne la position de l'utilisateur, mettre SANS_Joueur pour faire une partie avec uniquement des ordinateurs
<i>int</i>	*pStatistique : pointeur sur un tableau 4*4 qui contient des statistiques sur chaque joueur

Renvoie

char : 0 si tout le monde passe, 1 si la manche a bien lieu

< tableau qui stocke les annonces de chaque joueur, on y ajoute 1 pour le roi d'atout puis 1 pour la dame d'atout

< distribution des cartes

< On cherche si les joueurs ont des annonces à faire

< On passe à la phase suivante uniquement si un contrat a été pris sinon on relance une manche

< si un joueur a fait une belote rebelote il gagne 20 point

Définition à la ligne 188 du fichier general.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.9.2.5 menuPrincipal()

```
void menuPrincipal ( )
```

fonction qui gère le debut de la partie

Renvoie

void

< contrôle d'acquisition avec l'affichage de l'interface

<executer la fonction nouvelle partie

Définition à la ligne 16 du fichier general.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.9.2.6 nouvellePartie()

```
int nouvellePartie (
    char * pseudo[],
    Joueur utilisateur,
    int * pStatistique,
    int infoEcritureFichier[] )
```

lancement d'une nouvelle partie

Paramètres

<i>char</i>	*pseudo[] : tableau de pointeurs contenant les pseudos des différents joueurs
<i>Joueur</i>	utilisateur : donne la position de l'utilisateur, mettre SANS_Joueur pour faire une partie avec uniquement des ordinateur
<i>int</i>	*pStatistique : pointeur sur un tableau 4*4 qui contient des statistique sur chaque joueur
<i>int</i>	infoEcritureFichier[] : tableau contenant les info qui devront etre écrites dans le fichier

Renvoie

int : nombre de manche jusqu'a la victoire d'une équipe

< On change l'affichage pour avoir un affichage plus simple a lire lors d'une partie entre 4 ordinateur

< a supprimer une fois de debug fini

< Fin de partie

< affichage des resultat

< L'utilisatuer et nord gagnent

< est et touest gagnent

Définition à la ligne 99 du fichier general.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.9.2.7 pli()

```
Joueur pli (
    Contrat contrat,
    Joueur premierAJouer,
    char * pseudo[],
    Carte * pCarteMain,
    int pointManche[],
    int pointAnonce[],
    char belote[],
    Carte cartePli[],
    Carte carteAncienPli[],
    int score[],
    Joueur utilisateur,
    int numPli )
```

fonction qui gère un pli

Paramètres

<i>Contrat</i>	contrat : contrat qui a été choisi pour cette manche
<i>Joueur</i>	premierAJouer : premier joueur a jouer dans le pli
<i>char</i>	*pseudo[] : tableau de pointeur contenant les pseudo des différents joueur
<i>Carte</i>	*pCarteMain : pointeur sur le tableau qui stocke les carte dans la main de chaque joueur
<i>int</i>	pointManche[] : tableau contenant les point de chaque joueur dans la manche
<i>int</i>	pointAnonce[] : tableau contenant les point d'anonce de chaque joueur
<i>char</i>	belote[] : tableau contenant les ancone belote rebelote de chaque joueur
<i>Carte</i>	cartePli[] : tableau contenant les 4 carte du pli
<i>Carte</i>	carteAncienPli[] : tableau contenant les carte du pli précédent
<i>int</i>	score[] : tableau qui contient les score de la partie
<i>Joueur</i>	utilisateur : donne la position de l'utilisateur, mettre SANS_Joueur pour faire une partie avec uniquement des ordinateur

Renvoie

Joueur : le vainqueur du pli

< interface de pli Utilisateur

< interface de pli ordinateur

< on regarde si une dame ou un roi d'atout vien d'etre jouer pour ajouter au tableau belote[]

< affichage de la carte qui vien d'etre jou  e

< 10 de der, on ajoute 10 point au vainqueur si on est dans le dernier pli

D  finition    la ligne 390 du fichier general.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.9.2.8 poseCarte()

```
int poseCarte (
    Joueur joueur,
    int numCarte,
    Carte * pMainJoueurs,
    Carte pli[],
    int carteRestante )
```

fonction qui verifie la validit   d'une carte est la pose

Param  tres

<i>joueur</i>	variable qui defini la position du joueur qui effectue l'action
<i>numCarte</i>	variable qui defini la carte choisie par le joueur
<i>pMainJoueurs</i>	: pointeur qui renvoie vers le tableau de la main de tous les joueurs
<i>pPli</i>	: pointeur qui renvoie vers le tableau des carte jou��es dans le pli

Renvoie

D  finition    la ligne 463 du fichier general.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.9.2.9 proposeContrat()

```
Contrat proposeContrat (
    Contrat dernierContrat,
    Joueur parle,
    char * pseudo[],
    Carte * pCarteMain,
    Joueur utilisateur )
```

proposition d'un contrat par un joueur

Param  tres

<i>Contrat</i>	dernierContrat : dernier contrat propos��
<i>Joueur</i>	parle : joueur qui parle
<i>char</i>	*pseudo[]: tableau de pointeurs contenant les pseudos des diff��rents joueurs
<i>Carte</i>	*pCarteMain : pointeur sur le tableau qui stocke les cartes dans la main de chaque joueurs
<i>Joueur</i>	utilisateur : donne la position de l'utilisateur, mettre SANS_Joueur pour faire une partie avec uniquement des ordinateur

Renvoi

Contrat : nouveau contrat proposé par le joueur

< acquisition par l'utilisateur

< choix par l'ia d'un contrat

Définition à la ligne 371 du fichier general.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.10 Référence du fichier general.h

fichier header contenant les prototypes des fonctions les plus importantes pour le jeu de la belote

Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :

Fonctions

- void **initialisation** (int nbLigneFenetre, int nbColloneFenetre)
fonction qui gère l'initialisation et le lancement du programme
- void **menuPrincipal** ()
fonction qui gère le début de la partie
- int **nouvellePartie** (char *pseudo[], **Joueur** utilisateur, int *pStatistique, int infoEcritureFichier[])
lancement d'une nouvelle partie
- char **manche** (char *pseudo[], int score[], **Joueur** dealer, **Joueur** utilisateur, int *pStatistique)
lancement d'une manche
- **Contrat** **annonceContrat** (char *pseudo[], **Joueur** dealer, **Carte** *pCarteMain, **Joueur** utilisateur)
gère l'annonce et la surenchère des contrats par les joueurs
- **Contrat** **proposeContrat** (**Contrat** dernierContrat, **Joueur** parle, char *pseudo[], **Carte** *pCarteMain, **Joueur** utilisateur)
proposition d'un contrat par un joueur
- **Joueur** **pli** (**Contrat** contrat, **Joueur** premierAJouer, char *pseudo[], **Carte** *pCarteMain, int pointManche[], int pointAnonce[], char belote[], **Carte** cartePli[], **Carte** carteAncienPli[], int score[], **Joueur** utilisateur, int numPli)
fonction qui gère un pli
- int **poseCarte** (**Joueur** joueur, int numCarte, **Carte** *pMainJoueurs, **Carte** pli[], int carteRestante)
fonction qui vérifie la validité d'une carte et la pose
- char **calculPointManche** (**Contrat** contrat, int pointManche[], int pointAnonce[], int pointBelote[], int score[], char *pseudo[], **Joueur** utilisateur)
compte les points gagnés par chaque joueur au cours de la manche et fonction du contrat et ajoute ces points au tableau qui contient tous les scores de la partie

4.10.1 Description détaillée

fichier header contenant les prototypes des fonctions les plus importantes pour le jeu de la belote

Auteur

Carlo.A & Florian.C

Version

v1.0

Date

12 juin 2020

4.10.2 Documentation des fonctions

4.10.2.1 annonceContrat()

```
Contrat annonceContrat (
    char * pseudo[],
    Joueur dealer,
    Carte * pCarteMain,
    Joueur utilisateur )
```

gère l'annonce et la surenchère des contrats par les joueurs

Paramètres

<i>char</i>	*pseudo[]: tableau de pointeurs contenant les pseudos des différents joueurs
<i>Joueur</i>	dealer : joueur qui distribue les cartes
<i>Carte</i>	*pCarteMain : pointeur sur le tableau qui stocke les cartes dans la main de chaque joueur
<i>Joueur</i>	utilisateur : donne la position de l'utilisateur, mettre SANS_Joueur pour faire une partie avec uniquement des ordinateurs

Renvoie

Contrat : contrat final qui a été choisi pour la partie

< Affichage du choix du joueur

Définition à la ligne 287 du fichier general.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.10.2.2 calculPointManche()

```
char calculPointManche (
    Contrat contrat,
    int pointManche[],
    int pointAnonce[],
    int pointBelote[],
    int score[],
    char * pseudo[],
    Joueur utilisateur )
```

compte les points gagnés par chaque joueur au cours de la manche et fonction du contrat et ajoute ces points au tableau qui contient tous les scores de la partie

Paramètres

<i>Contrat</i>	contrat : contrat qui a été choisi durant la manche
<i>int</i>	pointManche[] : tableau qui contient les points de la manche
<i>int</i>	pointAnonce[] : tableau qui contient les points d'annonce de la manche
<i>int</i>	pointBelote[] : tableau contenant les points de belote et de rebelote de chaque joueur
<i>int</i>	score[] : tableau qui contient les scores de la partie
<i>char</i>	*pseudo[]: tableau de pointeurs contenant les pseudos des différents joueurs
<i>Joueur</i>	utilisateur : donne la position de l'utilisateur, mettre SANS_Joueur pour faire une partie avec uniquement des ordinateurs

Renvoie

char : 1 si le contrat a été aussi et 0 si le contrat a échoué

< On recherche si le contrat est rempli

< cas où un des joueurs a pris un capot

< cas où un des joueurs a pris une généralisation

< Le contrat est rempli si le score de l'équipe est supérieur à ce qui est annoncé et si il y a 82 points sans les annonces

< on affiche ce qui s'est passé

Définition à la ligne 476 du fichier general.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.10.2.3 initialisation()

```
void initialisation (
    int nbLigneFenetre,
    int nbCollonneFenetre )
```

fonction qui gère l'initialisation et le lancement du programme

Paramètres

<i>nbLigneFenetre</i>	: taille verticale de la fenêtre
<i>nbCollonneFenetre</i>	: taille horizontale de la fenêtre

Renvoie

void

Définition à la ligne 11 du fichier general.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.10.2.4 manche()

```
char manche (
    char * pseudo[],
    int score[],
    Joueur dealer,
    Joueur utilisateur,
    int * pStatistique )
```

lancement d'une manche

Paramètres

<i>char</i>	*pseudo[] : tableau de pointeurs contenant les pseudos des différents joueurs
<i>int</i>	score[] : tableau contenant les scores des joueur
<i>Joueur</i>	dealer : joueur qui distribue les cartes
<i>Joueur</i>	utilisateur : donne la position de l'utilisateur, mettre SANS_Joueur pour faire une partie avec uniquement des ordinateur
<i>int</i>	*pStatistique : pointeur sur un tableau 4*4 qui contient des statistique sur chaque joueur

Renvoie

char : 0 si tout le monde passe, 1 si la manche a bien lieu

< tableau qui stocke les anonce de chaque joueur, on y ajoute 1 pour le roi d'atout puis 1 pour la dame d'atout

< distribution des cartes

< On cherche si les joueurs on des anonce a faire

< On passe a la phase suivante uniquement si un contrat a ete pris sinon on relance une manche

< si un joueur a fait une belote rebelote il gagne 20 point

Définition à la ligne 188 du fichier general.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.10.2.5 menuPrincipal()

```
void menuPrincipal ( )
```

fonction qui gère le debut de la partie

Renvoie

void

< contrôle le d'acquisition avec l'affichage de l'interface

<executer la fonction nouvelle partie

Définition à la ligne 16 du fichier general.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.10.2.6 nouvellePartie()

```
int nouvellePartie (
    char * pseudo[],
    Joueur utilisateur,
    int * pStatistique,
    int infoEcritureFicher[] )
```

lancement d'une nouvelle partie

Paramètres

<i>char</i>	*pseudo[] : tableau de pointeurs contenant les pseudos des différents joueurs
<i>Joueur</i>	utilisateur : donne la position de l'utilisateur, mettre SANS_Joueur pour faire une partie avec uniquement des ordinateur
<i>int</i>	*pStatistique : pointeur sur un tableau 4*4 qui contient des statistique sur chaque joueur
<i>int</i>	infoEcritureFichier[] : tableau contenant les info qui devront etre Ã©crite dans le fichier

Renvoi

int : nombre de manche jusqu'a la victoire d'une Ã©quipe

< On change l'affichage pour avoir un affichage plus simple a lire lors d'une partie entre 4 ordinateur

< a supprimer une fois de debug fini

< Fin de partie

< affichage des resultat

< L'utilisatuer et nord gagnent

< est et ouest gagnent

Définition à la ligne 99 du fichier general.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.10.2.7 pli()

```
Joueur pli (
    Contrat contrat,
    Joueur premierAJouer,
    char * pseudo[],
    Carte * pCarteMain,
    int pointManche[],
    int pointAnonce[],
    char belote[],
    Carte cartePli[],
    Carte carteAncienPli[],
    int score[],
    Joueur utilisateur,
    int numPli )
```

fonction qui gère un pli

Paramètres

<i>Contrat</i>	contrat : contrat qui a été choisi pour cette manche
<i>Joueur</i>	premierAJouer : premier joueur a jouer dans le pli
<i>char</i>	*pseudo[]: tableau de pointeur contenant les pseudo des différents joueur
<i>Carte</i>	*pCarteMain : pointeur sur le tableau qui stocke les carte dans la main de chaque joueur
<i>int</i>	pointManche[] : tableau contenant les point de chaque joueur dans la manche
<i>int</i>	pointAnonce[] : tableau contenant les point d'annonce de chaque joueur
<i>char</i>	belote[] : tableau contenant les ancone belote rebelote de chaque joueur
<i>Carte</i>	cartePli[] : tableau contenant les 4 carte du pli
<i>Carte</i>	carteAncienPli[] : tableau contenant les carte du pli précédent
<i>int</i>	score[] : tableau qui contient les score de la partie

Renvoie

Joueur : le vainqueur du pli

< interface de pli Utilisateur

< interface de pli ordinateur

< on regarde si une dame ou un roi d'atout vien d'etre jouer pour ajouter au tableau belote[]

< affichage de la carte qui vien d'etre jou  e

< 10 de der, on ajoute 10 point au vainqueur si on est dans le dernier pli

D  finition    la ligne 390 du fichier general.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.10.2.8 poseCarte()

```
int poseCarte (
    Joueur joueur,
    int numCarte,
    Carte * pMainJoueurs,
    Carte pli[],
    int carteRestante )
```

fonction qui verifie la validit   d'une carte est la pose

Param  tres

<i>joueur</i>	variable qui defini la position du joueur qui effectue l'action
<i>numCarte</i>	variable qui defini la carte choisie par le joueur
<i>pMainJoueurs</i>	: pointeur qui renvoie vers le tableau de la main de tous les joueurs
<i>pPli</i>	: pointeur qui renvoie vers le tableau des carte jou��es dans le pli

Renvoie

D  finition    la ligne 463 du fichier general.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.10.2.9 proposeContrat()

```
Contrat proposeContrat (
    Contrat dernierContrat,
    Joueur parle,
    char * pseudo[],
    Carte * pCarteMain,
    Joueur utilisateur )
```

proposition d'un contrat par un joueur

Paramètres

<i>Contrat</i>	dernierContrat : dernier contrat proposÃ©
<i>Joueur</i>	parle : joueur qui parle
<i>char</i>	*pseudo[]: tableau de pointeurs contenant les pseudos des différents joueurs
<i>Carte</i>	*pCarteMain : pointeur sur le tableau qui stocke les cartes dans la main de chaque joueurs
<i>Joueur</i>	utilisateur : donne la position de l'utilisateur, mettre SANS_Joueur pour faire une partie avec uniquement des ordinateur

Renvoi

Contrat : nouveau contrat proposÃ© par le joueur

< acquisition par l'utilisateur

< choix par l'ia d'un contrat

Définition à la ligne 371 du fichier general.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.11 Référence du fichier gestionCarte.c

fichier contenant les fonctions relative a la gestion des cartes

```
#include "main.h"
```

Graphe des dépendances par inclusion de gestionCarte.c:

Fonctions

- void *distribueCarte* (*Carte* *pCarteMain)
distribue les cartes entre les différent joueur
- void *setCarte* (*Carte* *carte, *Valeur* valeurCarte, *Couleur* couleurCarte)
donne une valeur et une couleur a une carte
- void *supprimeCarte* (*Carte* carte[], int nbCarte, int carteASupprimer)
supprime une carte d'un tableau et réduit la taille de ce tableau par le bas
- *Joueur* vainqueurPli (*Carte* pli[], *Couleur* atout, *Joueur* premierAJouer)
donne le vainquer d'un pli
- float *forceCarte* (*Carte* carteACalculer, *Couleur* atout, *Couleur* entame)
calcul la force d'une carte, en faisant la probabilité qu'elle a de gagné contre toutes les autres cartes
- char *carteValide* (*Carte* cartePose, *Carte* pli[], *Couleur* atout, *Carte* *pCarteMainJoueur, *Joueur* premierAJouer, *Joueur* parle)
ddétermine si une carte peut etre posé par un joueur
- char *rechercherCarte* (*Carte* *pCarte, int nbCarte, *Couleur* couleurCherche, *Valeur* valeurCherche)
cherche dans un tableau de carte si il y a une certaine couleur ou une certaine valeur ou les deux
- char *rechercherCarteSuperieur* (*Carte* *pCarte, int nbCarte, *Carte* carteCherche, *Couleur* atout, *Couleur* entame)
cherche dans un tableau de carte si il y a une carte d'une force supérieur a carteCherche
- float *sommeForceCarte* (*Carte* *tableauCarte, int nbCarte, *Couleur* atout)
calucule la somme force des carte dans un tableau
- void *trieCarte* (*Carte* tableauCarte[], int nbCarte, *Couleur* atout)
trie un tableau de cartes
- char *cartePlaceAvant* (*Carte* carteRefference, *Carte* carteCompare, *Couleur* atout)
permet de savoir si une carte doit etre placée avant lors du trie des cartes
- void *rechercheAnnonce* (*Carte* *pCarteMain, int pointAnonce[], *Joueur* utilisateur, char *pseudo[])
rechere dans un tableau de cartes qui correspond a la main d'un joueur si il peut faire des anonc, et si oui ajoute les points assacier au tableau dans le tableau des annonce

4.11.1 Description détaillée

fichier contenant les fonctions relative a la gestion des cartes

Auteur

Carlo.A & Florian.C

Version

v1.0

Date

12 juin 2020

4.11.2 Documentation des fonctions

4.11.2.1 `cartePlaceAvant()`

```
char cartePlaceAvant (
    Carte carteReference,
    Carte carteCompare,
    Couleur atout )
```

permet de savoir si une carte doit etre placée avant lors du trie des cartes

Paramètres

<i>Carte</i>	carteReference : carte par raport a laquelle on compare
<i>Carte</i>	carteCompare : carte que l'on compare
<i>Couleur</i>	atout : couleur de l'atout dans la manche

Renvoie

char : renvoie 1 si la catre doit etre placée avant 0 sinon

< Les deux carte sont elle de la meme famille

< si carteCompate est un atout

< si la couleur de la carteCompare est devant celle de carteReference dans l'Ã©numÃ©ration(ordre arbitraire)

Définition à la ligne 404 du fichier gestionCarte.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.11.2.2 carteValide()

```
char carteValide (
    Carte cartePose,
    Carte pli[],
    Couleur atout,
    Carte * pCarteMainJoueur,
    Joueur premierAJouer,
    Joueur parle )
```

ddétermine si une carte peut etre posé par un joueur

Paramètres

<i>Carte</i>	cartPose : carte que l'on veut poser
<i>Carte</i>	pli[] : tableau contenet les carte du pli
<i>Couleur</i>	atout : couleur de l'atout
<i>Carte</i>	*pCarteMainJoueur pointeur sur un tableau de 8 carte contenant les cartes dans la main du joueur
<i>Joueur</i>	premierAJouer : premier joueur a jouer dans le pli
<i>Joueur</i>	parle : joueur en train de jouer

Renvoie

char 1 si la carte est valide 0 sinon

- < Fonction faite a partir de l'oranigrame qui montre comment d'Ã©terminer si une carte est valide a partir des rÃ¨gle
- < On verifie que ce n'est pas une carte vide
- < premiÃ¨re carte du plis ?
- < Le joueur posÃ© de il la couleur demandÃ©e ?
- < l'entame est en atout
- < Le joueur a il un ajout de valeur supÃ©rieur au meilleur ajout posÃ©
- < si la carte est un ajout le valeur supÃ©rieur au meilleur atout posÃ©
- < Si la carte est un atout
- < Si la carte est dans la couleur demandÃ©e
- < Le partemenaire est maitre ?
- < Le joueur a il un atout ?
- < il y a deja un ajout de posÃ©
- < Le joueur a il un ajout de valeur supÃ©rieur au meilleur ajout posÃ©
- < si la carte est un ajout le valeur supÃ©rieur au meilleur atout posÃ©
- < Si la carte est un atout alors elle est valide

Définition à la ligne 261 du fichier gestionCarte.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.11.2.3 distribueCarte()

```
void distribueCarte (
    Carte * pCarteMain )
```

distribue les cartes entre les différent joueur

Paramètres

<i>Carte</i>	*pCarteMain : pointeur sur le tableau qui stocke les carte dans la main de chaque joueur

Définition à la ligne 12 du fichier gestionCarte.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.11.2.4 forceCarte()

```
float forceCarte (
    Carte carteACalculer,
    Couleur atout,
    Couleur entame )
```

calcul la force d'une carte, en faisant la probabilité qu'elle a de gagné contre toutes les autres cartes

Paramètres

<i>Carte</i>	carteACalculer : carte dont on veut connaitre la force
<i>Couleur</i>	atout : couleur de l'atout durant la partie
<i>Couleur</i>	entame : couleur de l'entame durant la partie

Renvoi

float : probabilité de victoire de la carte

< la carte est un atout ou que la manche se joue en tout atout

< si on est en tout atout mais que la couleur de la carte n'est pas la couleur de l'entame alors la carte sera plus faible que toutes les cartes dans la bonne couleur

< la manche est en sans atout et la carte est dans la couleur de l'entame

< on est en atout d'une certaine couleur ou en sans atout avec une couleur différente de l'entame

< quand l'entame est différent de la couleur de la carte et que ce n'est pas un atout alors on perd à chaque fois

Définition à la ligne 80 du fichier gestionCarte.c.

Voici le graphe des appelants de cette fonction :

4.11.2.5 rechercheAnnonce()

```
void rechercheAnnonce (
    Carte * pCarteMain,
    int pointAnnonce[],
    Joueur utilisateur,
    char * pseudo[] )
```

recherche dans un tableau de cartes qui correspond à la main d'un joueur si il peut faire des annonces, et si oui ajoute les points associés au tableau dans le tableau des annonces

Paramètres

<i>Carte</i>	*pCarteMain : pointeur sur le premier élément du tableau qui contient les cartes de tout les joueur
<i>int</i>	pointAnonce[] : tableau contenant les point d'annonce
<i>Joueur</i>	utilisateur : donne la position de l'utilisateur, mettre SANS_Joueur pour faire une partie avec uniquement des ordinateur
<i>char</i>	*pseudo[] : tableau contenant les pseudo des 4 joueur

Renvoie

void

< on cherche si le joueur a un carré

Définition à la ligne 435 du fichier gestionCarte.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.11.2.6 rechercherCarte()

```
char rechercherCarte (
    Carte * pCarte,
    int nbCarte,
    Couleur couleurCherche,
    Valeur valeurCherche )
```

cherche dans un tableau de carte si il y a une certaine couleur ou une certaine valeur ou les deux

Paramètres

<i>Carte</i>	*pCarte : pointeur sur un tableau de carte
<i>int</i>	nbCarte : nombre de carte dans le tableau
<i>Couleur</i>	couleurCherche : couleur que l'on veut chercher dans le tableau (mettre SANS_COULEUR pour rechercher toutes les couleur)
<i>Valeur</i>	valeurCherche : valeur que l'on cherche dans le tableau (mettre SANS_VALEUR pour chercher toutes les valeur)

Renvoie

char 1 si on a trouvé une carte qui correspond au critère 0 sinon

Définition à la ligne 343 du fichier gestionCarte.c.

Voici le graphe des appelants de cette fonction :

4.11.2.7 rechercherCarteSuperieur()

```
char rechercherCarteSuperieur (
    Carte * pCarte,
    int nbCarte,
```

```
Carte carteCherche,  
Couleur atout,  
Couleur entame )
```

cherche dans un tableau de carte si il y a une carte d'une force supérieur a carteCherche

Paramètres

<i>Carte</i>	*pCarte : pointeur sur un tableau de carte
<i>int</i>	nbCarte : nombre de carte dans le tableau
<i>Carte</i>	carteCherche : carte a laquelle on veut que la carte cherchée soit supérieur
<i>Couleur</i>	atout : couleur de l'atout dans la manche
<i>Couleur</i>	entame : couleur de l'entame dans la manche

Renvoie

char 1 si on a trouvé une carte qui correspond au critère 0 sinon

Définition à la ligne 355 du fichier gestionCarte.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.11.2.8 setCarte()

```
void setCarte (
    Carte * carte,
    Valeur valeurCarte,
    Couleur couleurCarte )
```

donne une valeur et une couleur a une carte

Paramètres

<i>Carte</i>	*carte : pointeur sur la carte a modifier
<i>Valeur</i>	valeurCarte : valeur a donner a la carte
<i>Couleur</i>	couleurCarte : couleur a donner a la carte

Renvoie

void

Définition à la ligne 50 du fichier gestionCarte.c.

Voici le graphe des appelants de cette fonction :

4.11.2.9 sommeForceCarte()

```
float sommeForceCarte (
    Carte * tableauCarte,
    int nbCarte,
    Couleur atout )
```

calcule la somme force des carte dans un tableau

Paramètres

<i>Carte</i>	tableauCarte : tableau de carte sur lequel on veut travailler
<i>int</i>	nbCarte : nombre de carte dans le tableau
<i>Couleur</i>	atout : couleur de l'atout sur la manche

Renvoie

float : la valeur de la somme des force des carte

Définition à la ligne 369 du fichier gestionCarte.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.11.2.10 supprimerCarte()

```
void supprimerCarte (
    Carte carte[],
    int nbCarte,
    int carteASupprimer )
```

supprime une carte d'un tableau et réduit la taille de ce tableau par le bas

Paramètres

<i>Carte</i>	carte[] : tableau conteneant les carte a modifier
<i>int</i>	nbCarte : nombre total de carte
<i>int</i>	carteASupprimer : indice dans le tableau de la carte a enlever

Renvoie

void

Définition à la ligne 56 du fichier gestionCarte.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.11.2.11 trieCarte()

```
void trieCarte (
    Carte tableauCarte[],
    int nbCarte,
    Couleur atout )
```

trie un tableau de cartes

Paramètres

<i>Carte</i>	tableauCarte[] : tableau contenant les cartes a trier
<i>int</i>	nbCarte nombre de carte dans le tableau
<i>Couleur</i>	atout : couleur de l'atout dans la manche

Renvoie

void

< On réalise un tri a bulle pour mettre atout du plus fort au moins fort au début de la main puis les carte par famille

< Si la carte dois aller devant dans le tableau

Définition à la ligne 386 du fichier gestionCarte.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.11.2.12 vainqueurPli()

```
Joueur vainqueurPli (
    Carte pli[],
    Couleur atout,
    Joueur premierAJouer )
```

donne le vainquer d'un pli

Paramètres

<i>Carte</i>	pli[] : tableau contenant les carte du pli
<i>Couleur</i>	atout : couleur de l'atout durant cette manche
<i>Joueur</i>	premierAJouer : joueur qui pose la première carte du pli

Renvoie

Joueur : le jouer qui a gagné le pli

< La carte du joueur que l'on test est plus forte que la meilleur actuelle alors ca devient la meilleure

Définition à la ligne 64 du fichier gestionCarte.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.12 Référence du fichier gestionCarte.h

fichier header contenant les prototypes des fonctions relative a la gestion des cartes

Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :

Fonctions

- void **distribueCarte** (**Carte** *pCarteMain)
distribue les cartes entre les différent joueur
- void **setCarte** (**Carte** *carte, **Valeur** valeurCarte, **Couleur** couleurCarte)
donne une valeur et une couleur a une carte
- void **supprimeCarte** (**Carte** carte[], int nbCarte, int carteASupprimer)
supprime une carte d'un tableau et réduit la taille de ce tableau par le bas
- **Joueur** **vainqueurPli** (**Carte** pli[], **Couleur** atout, **Joueur** premierAJouer)
donne le vainquer d'un pli
- float **forceCarte** (**Carte** carteACalculer, **Couleur** atout, **Couleur** entame)
calcul la force d'une carte, en faisant la probabilité qu'elle a de gagné contre toutes les autres cartes
- char **carteValide** (**Carte** cartePose, **Carte** pli[], **Couleur** atout, **Carte** *pCarteMainJoueur, **Joueur** premierA↔Jouer, **Joueur** parle)
ddétermine si une carte peut etre posé par un joueur
- char **rechercherCarte** (**Carte** *pCarte, int nbCarte, **Couleur** couleurCherche, **Valeur** valeurCherche)

- char `rechercherCarteSuperieur` (*Carte* *pCarte, int nbCarte, *Carte* carteCherche, *Couleur* atout, *Couleur* entame)
cherche dans un tableau de carte si il y a une certaine couleur ou une certaine valeur ou les deux
- float `sommeForceCarte` (*Carte* *tableauCarte, int nbCarte, *Couleur* atout)
cherche dans un tableau de carte si il y a une carte d'une force supérieur a carteCherche
- void `trieCarte` (*Carte* tableauCarte[], int nbCarte, *Couleur* atout)
calucule la somme force des carte dans un tableau
- char `cartePlaceAvant` (*Carte* carteRefference, *Carte* carteCompare, *Couleur* atout)
trie un tableau de cartes
- void `rechercheAnnonce` (*Carte* *pCarteMain, int pointAnonce[], *Joueur* utilisateur, char *pseudo[])
permet de savoir si une carte doit etre placée avant lors du trie des cartes
- void `rechercheAnnonce` (*Carte* *pCarteMain, int pointAnonce[], *Joueur* utilisateur, char *pseudo[])
rechere dans un tableau de cartes qui correspond a la main d'un joueur si il peut faire des anonc, et si oui ajoute les points assacier au tableau dans le tableau des anonc

4.12.1 Description détaillée

ficher header contenant les prototypes des fonctions relative a la gestion des cartes

Auteur

Carlo.A & Florian.C

Version

v1.0

Date

12 juin 2020

4.12.2 Documentation des fonctions

4.12.2.1 `cartePlaceAvant()`

```
char cartePlaceAvant (
    Carte carteRefference,
    Carte carteCompare,
    Couleur atout )
```

permet de savoir si une carte doit etre placée avant lors du trie des cartes

Paramètres

<i>Carte</i>	carteRefference : carte par raport a laquelle on compare
<i>Carte</i>	carteCompare : carte que l'on compare
<i>Couleur</i>	atout : couleur de l'atout dans la manche

Renvoie

char : renvoie 1 si la catre doit etre placée avant 0 sinon

< Les deux carte sont elle de la meme famille

< si carteCompate est un atout

< si la couleur de la carteCompare est devant celle de carteReference dans l'Ã©numÃ©ration(ordre arbitraire)

Définition à la ligne 404 du fichier gestionCarte.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.12.2.2 carteValide()

```
char carteValide (
    Carte cartePose,
    Carte pli[],
    Couleur atout,
    Carte * pCarteMainJoueur,
    Joueur premierAJouer,
    Joueur parle )
```

ddétermine si une carte peut etre posé par un joueur

Paramètres

<i>Carte</i>	cartPose : carte que l'on veut poser
<i>Carte</i>	pli[] : tableau contenet les carte du pli
<i>Couleur</i>	atout : couleur de l'atout
<i>Carte</i>	*pCarteMainJoueur pointeur sur un tableau de 8 carte contenant les cartes dans la main du joueur
<i>Joueur</i>	premierAJouer : premier joueur a jouer dans le pli
<i>Joueur</i>	parle : joueur en train de jouer

Renvoie

char 1 si la carte est valide 0 sinon

< Fonction faite a partir de l'oranigramme qui montre comment dÃ©terminer si une carte est valide a partir des rÃ¨gle

< On verifier que ce n'est pas une carte vide

< premiÃ¨re carte du plis ?

< Le joueur posÃ© de il la couleur demandÃ©e ?

< l'entame est en atout

< Le joueur a il un ajout de valeur supÃ©rieur au meilleur ajout posÃ©

< si la carte est un ajout le valeur supÃ©rieur au meilleur atout posÃ©

< Si la carte est un atout

< Si la carte est dans la couleur demandÃ©e

< Le partemenaire est maitre ?

< Le joueur a il un atout ?

< il y a deja un ajout de posÃ©

< Le joueur a il un ajout de valeur supÃ©rieur au meilleur ajout posÃ©

< si la carte est un ajout le valeur supÃ©rieur au meilleur atout posÃ©

< Si la carte est un atout alors elle est valide

Définition à la ligne 261 du fichier gestionCarte.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.12.2.3 distribueCarte()

```
void distribueCarte (
    Carte * pCarteMain )
```

distribue les cartes entre les différent joueur

Paramètres

<i>Carte</i>	*pCarteMain : pointeur sur le tableau qui stocke les carte dans la main de chaque joueur

Définition à la ligne 12 du fichier gestionCarte.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.12.2.4 forceCarte()

```
float forceCarte (
    Carte carteACalculer,
    Couleur atout,
    Couleur entame )
```

calcul la force d'une carte, en faisant la probabilité qu'elle a de gagné contre toutes les autres cartes

Paramètres

<i>Carte</i>	carteACalculer : carte dont on veut connaitre la force
<i>Couleur</i>	atout : couleur de l'atout durant la partie
<i>Couleur</i>	entame : couleur de l'entame durant la partie

Renvoie

float : probabilité de victoire de la carte

< la carte est un atout ou que la manche se joue en tout atout

< si on est en tout atout mais que la couleur de la carte n'est pas la couleur de l'entame alors la carte sera plus faible que toutes les cartes dans la bonne couleur

< la manche est en sans atout et la carte est dans la couleur de l'entame

< on est en atout d'une certaine couleur ou en sans atout avec une couleur différente de l'entame

< quand l'entame est différente de la couleur de la carte et que ce n'est pas un atout alors on perd à chaque fois

Définition à la ligne 80 du fichier gestionCarte.c.

Voici le graphe des appelants de cette fonction :

4.12.2.5 rechercheAnnonce()

```
void rechercheAnnonce (
    Carte * pCarteMain,
    int pointAnnonce[],
    Joueur utilisateur,
    char * pseudo[] )
```

recherche dans un tableau de cartes qui correspond à la main d'un joueur si il peut faire des annonces, et si oui ajoute les points associés au tableau dans le tableau des annonces

Paramètres

<i>Carte</i>	*pCarteMain : pointeur sur le premier élément du tableau qui contient les cartes de tout les joueurs
<i>int</i>	pointAnnonce[] : tableau contenant les points d'annonces
<i>Joueur</i>	utilisateur : donne la position de l'utilisateur, mettre SANS_Joueur pour faire une partie avec uniquement des ordinateurs
<i>char</i>	*pseudo[] : tableau contenant les pseudos des 4 joueurs

Renvoie

void

< on cherche si le joueur a un carré

Définition à la ligne 435 du fichier gestionCarte.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.12.2.6 rechercherCarte()

```
char rechercherCarte (
    Carte * pCarte,
    int nbCarte,
    Couleur couleurCherche,
    Valeur valeurCherche )
```

cherche dans un tableau de cartes si il y a une certaine couleur ou une certaine valeur ou les deux

Paramètres

<i>Carte</i>	*pCarte : pointeur sur un tableau de carte
<i>int</i>	nbCarte : nombre de carte dans le tableau
<i>Couleur</i>	couleurCherche : couleur que l'on veut chercher dans le tableau (mettre SANS_COULEUR pour rechercher toutes les couleur)
<i>Valeur</i>	valeurCherche : valeur que l'on cherche dans le tableau (mettre SANS_VALEUR pour chercher toutes les valeur)

Renvoie

char 1 si on a trouvé une carte qui correspond au critère 0 sinon

Définition à la ligne 343 du fichier gestionCarte.c.

Voici le graphe des appelants de cette fonction :

4.12.2.7 rechercherCarteSuperieur()

```
char rechercherCarteSuperieur (
    Carte * pCarte,
    int nbCarte,
    Carte carteCherche,
    Couleur atout,
    Couleur entame )
```

cherche dans un tableau de carte si il y a une carte d'une force supérieur a carteCherche

Paramètres

<i>Carte</i>	*pCarte : pointeur sur un tableau de carte
<i>int</i>	nbCarte : nombre de carte dans le tableau
<i>Carte</i>	carteCherche : carte a laquelle on veut que la carte cherchée soit supérieur
<i>Couleur</i>	atout : couleur de l'atout dans la manche
<i>Couleur</i>	entame : couleur de l'entame dans la manche

Renvoie

char 1 si on a trouvé une carte qui correspond au critère 0 sinon

Définition à la ligne 355 du fichier gestionCarte.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.12.2.8 setCarte()

```
void setCarte (
    Carte * carte,
    Valeur valeurCarte,
    Couleur couleurCarte )
```

donne une valeur et une couleur a une carte

Paramètres

<i>Carte</i>	*carte : pointeur sur la carte a modifier
<i>Valeur</i>	valeurCarte : valeur a donner a la carte
<i>Couleur</i>	couleurCarte : couleur a donner a la carte

Renvoie

void

Définition à la ligne 50 du fichier gestionCarte.c.

Voici le graphe des appelants de cette fonction :

4.12.2.9 sommeForceCarte()

```
float sommeForceCarte (
    Carte * tableauCarte,
    int nbCarte,
    Couleur atout )
```

calucule la somme force des carte dans un tableau

Paramètres

<i>Carte</i>	tableauCarte : tableau de carte sur lequel on veut travailler
<i>int</i>	nbCarte : nombre de carte dans le tableau
<i>Couleur</i>	atout : couleur de l'atout sur la manche

Renvoie

float : la valeur de la somme des force des carte

Définition à la ligne 369 du fichier gestionCarte.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.12.2.10 supprimeCarte()

```
void supprimeCarte (
    Carte carte[],
    int nbCarte,
    int carteASupprimer )
```

supprime une carte d'un tableau et réduit la taille de ce tableau par le bas

Paramètres

<i>Carte</i>	carte[] : tableau conteneant les carte a modifier
<i>int</i>	nbCarte : nombre total de carte
<i>int</i>	carteASupprimer : indice dans le tableau de la carte a enlever

Renvoie

void

Définition à la ligne 56 du fichier gestionCarte.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.12.2.11 trieCarte()

```
void trieCarte (
    Carte tableauCarte[],
    int nbCarte,
    Couleur atout )
```

trie un tableau de cartes

Paramètres

<i>Carte</i>	tableauCarte[] : tableau contenant les cartes a trier
<i>int</i>	nbCarte nombre de carte dans le tableau
<i>Couleur</i>	atout : couleur de l'atout dans la manche

Renvoie

void

< On réalise un tri à bulle pour mettre atout du plus fort au moins fort au début de la main puis les cartes par famille

< Si la carte doit aller devant dans le tableau

Définition à la ligne 386 du fichier gestionCarte.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.12.2.12 vainqueurPli()

```
Joueur vainqueurPli (
    Carte pli[],
    Couleur atout,
    Joueur premierAJouer )
```

donne le vainqueur d'un pli

Paramètres

<i>Carte</i>	pli[] : tableau contenant les cartes du pli
<i>Couleur</i>	atout : couleur de l'atout durant cette manche
<i>Joueur</i>	premierAJouer : joueur qui pose la première carte du pli

Renvoie

Joueur : le joueur qui a gagné le pli

< La carte du joueur que l'on test est plus forte que la meilleur actuelle alors ca devient la meilleure

Définition à la ligne 64 du fichier gestionCarte.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.13 Référence du fichier gestionFichier.c

fichier source contenant les fonctions relative à la gestion des fichiers

```
#include "main.h"
```

Graphe des dépendances par inclusion de gestionFichier.c:

Fonctions

- int [ecriturePseudo](#) (char *pPseudo, FILE *pFichier)
fonction qui recherche dans un fichier si le joueur est déjà enregistré au le rajoute à la fin si ce n'est pas le cas
- int [ecrireStatistique](#) (FILE *fichier, int ligne, int statAModifier, int type)
modifie la valeur d'une statistique dans le fichier de sauvegarde des scores
- int [ecrireLeaderboard](#) (FILE *fichier, char *pseudo, int scoreJ)
vérifie si un nouveau record a été battu et l'ajoute si c'est le cas

4.13.1 Description détaillée

fichier source contenant les fonctions relative à la gestion des fichiers

Auteur

Carlo.A & Florian.C

Version

v1.0

Date

12 juin 2020

4.13.2 Documentation des fonctions

4.13.2.1 [ecrireLeaderboard\(\)](#)

```
int ecrireLeaderboard (
    FILE * fichier,
    char * pseudo,
    int score )
```

vérifie si un nouveau record a été battu et l'ajoute si c'est le cas

Paramètres

<i>FILE</i>	*fichier : pointeur vers fichier de meilleurs score
<i>char</i>	*pPseudo : pointeur vers le pseudo du joueur
<i>int</i>	score : score a éventuellement enregistrer

Renvoie

0

Définition à la ligne 97 du fichier gestionFichier.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.13.2.2 écrireStatistique()

```
int écrireStatistique (
    FILE * fichier,
    int ligne,
    int statAModifier,
    int type )
```

modifie la valeur d'une statistique dans le fichier de sauvegarde des scores

Paramètres

<i>FILE</i>	*fichier : pointeur vers le fichier
<i>int</i>	ligne : ligne du fichier où laquelle sont sauvegardées les statistiques du joueur
<i>int</i>	statAModifier : la nouvelle statistique à enregistrer
<i>int</i>	type : vaut 1 pour modifier le nombre de victoires, vaut 2 pour changer le score max, vaut 3 pour changer le nombre de manche min pour une victoire

Renvoie

valeur enregistrée

Définition à la ligne 44 du fichier gestionFichier.c.

Voici le graphe des appelants de cette fonction :

4.13.2.3 écriturePseudo()

```
int écriturePseudo (
    char * pPseudo,
    FILE * pFichier )
```

fonction qui recherche dans un fichier si le joueur est déjà enregistré ou le rajoute à la fin si ce n'est pas le cas

Paramètres

<i>char</i>	*pPseudo: pointeur vers le nom du joueur
<i>FILE</i>	*pFichier : pointeur vers le fichier contenant les statistiques

Renvoie

le numero de la ligne dans lequel est noté le pseudo

Définition à la ligne 11 du fichier gestionFichier.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.14 Référence du fichier gestionFichier.h

fichier header contenant les prototypes des fonctions relative à la gestion des fichiers

Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :

Fonctions

- int [ecriturePseudo](#) (char *pPseudo, FILE *pFichier)
fonction qui recherche dans un fichier si le joueur est déjà enregistré au le rajoute la fin si ce n'est pas le cas
- int [ecrireStatistique](#) (FILE *fichier, int ligne, int statAModifier, int type)
modifie la valeur d'une statistique dans le fichier de sauvegarde des scores
- int [ecrireLeaderboard](#) (FILE *fichier, char *pseudo, int score)
vérifie si un nouveau record a été battu et l'ajoute si c'est le cas

4.14.1 Description détaillée

fichier header contenant les prototypes des fonctions relative à la gestion des fichiers

Auteur

Carlo.A & Florian.C

Version

v1.0

Date

12 juin 2020

4.14.2 Documentation des fonctions

4.14.2.1 [ecrireLeaderboard\(\)](#)

```
int ecrireLeaderboard (
    FILE * fichier,
    char * pseudo,
    int scoreJ )
```

vérifie si un nouveau record a été battu et l'ajoute si c'est le cas

Paramètres

<i>FILE</i>	*fichier : pointeur vers fichier de meilleurs score
<i>char</i>	*pPseudo : pointeur vers le pseudo du joueur
<i>int</i>	score : score a Ã©ventuellement enregistrer

Renvoie

0

Définition à la ligne 97 du fichier gestionFichier.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.14.2.2 ecrireStatistique()

```
int ecrireStatistique (
    FILE * fichier,
    int ligne,
    int statAModifier,
    int type )
```

modifie la valeur d'une statistique dans le fichier de sauvegarde des scores

Paramètres

<i>FILE</i>	*fichier : pointeur vers le fichier
<i>int</i>	ligne : ligne du fichier Ã laquelle sont sauvegardÃ©e les statistiques du joueur
<i>int</i>	statAModifier : la nouvelle statistique Ã enregistrer
<i>int</i>	type : vaut 1 pour modifier le nombre de victoires, vaut 2 pour changer le score max, vaut 3 pour changer le nombre de manche min pour une victoire

Renvoie

valeur enregistrÃ©e

Définition à la ligne 44 du fichier gestionFichier.c.

Voici le graphe des appelants de cette fonction :

4.14.2.3 ecriturePseudo()

```
int ecriturePseudo (
    char * pPseudo,
    FILE * pFichier )
```

fonction qui recherche dans un fichier si le joueur est déjà enregistré au le rajoute Ã la fin si ce n'est pas le cas

Paramètres

<i>char</i>	*pPseudo: pointeur vers le nom du joueur
<i>FILE</i>	*pFichier : pointeur vers le fichier contenant les statistiques

Renvoi

le numero de la ligne dans lequel est noté le pseudo

Définition à la ligne 11 du fichier gestionFichier.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.15 Référence du fichier ia.c

fichier source contenant les fonctions relative au intelligences artificielles

```
#include "main.h"
```

Graphe des dépendances par inclusion de ia.c:

Fonctions

- [Contrat proposeContratla](#) ([Joueur](#) parle, [Carte](#) *pCarteMain, [Contrat](#) dernierContrat)
proposition par l'ia d'un contrat
- int [choixCartelA](#) ([Joueur](#) joueur, [Carte](#) *pMainJoueur, [Carte](#) pli[], [Joueur](#) premierJoueur, [Couleur](#) atout, int carteRestante)
proposition de la pose d'une carte par l'IA

4.15.1 Description détaillée

fichier source contenant les fonctions relative au intelligences artificielles

Auteur

Carlo.A & Florian.C

Version

v1.0

Date

12 juin 2020

4.15.2 Documentation des fonctions

4.15.2.1 choixCartelA()

```
int choixCarteIA (
    Joueur joueur,
    Carte * pMainJoueur,
    Carte pli[],
    Joueur dernierVainqueur,
    Couleur atout,
    int carteRestante )
```

proposition de la pose d'une carte par l'IA

Paramètres

<i>Joueur</i>	joueur : determine qu'elle IA doit jouer
<i>Carte</i>	*pMainJoueur : pointeur vers le tableau qui enregistre la main de l'IA
<i>Carte</i>	pli[] : tableau qui enregistre les cartes jouées pendant le pli
<i>Joueur</i>	dernierVainqueur : defini quel joueur à posé la première carte
<i>Couleur</i>	atout : defini quelle couleur est en atout
<i>int</i>	carteRestante : nombre de cartes restantes en main

Renvoi

```
debug for(int i=0;i<nbCarteValide;i++){ printf("|%d| \n",numCarteValide[i]); } afficheMain(pCarteValidee);
```

```
debug for(int i=0;i<offset;i++){ printf("|%d| \n",numCarteGagnante[i]); } afficheMain(pCarteGagante);
```

Définition à la ligne 60 du fichier ia.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.15.2.2 proposeContratIa()

```
Contrat proposeContratIa (
    Joueur parle,
    Carte * pCarteMain,
    Contrat dernierContrat )
```

proposition par l'ia d'un contrat

Paramètres

<i>Joueur</i>	parle : joueur qui parle
<i>Carte</i>	*pCarteMain pointeur sur le tableau qui contient les cartes du joueur
<i>Contrat</i>	dernierContrat : dernier contrat qui a été proposé

Renvoi

Contrat

< Permet de déterminer de manière heuristique la meilleure valeur de seuilMinPrise et seuilMaxPrise

< ON coinche !

Définition à la ligne 11 du fichier ia.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.16 Référence du fichier ia.h

fichier header contenant les prototypes des fonctions relative aux intelligences artificielles

Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :

Fonctions

- `Contrat proposeContratla` (`Joueur` parle, `Carte *pCarteMain`, `Contrat` dernierContrat)
proposition par l'ia d'un contrat
- `int choixCartelA` (`Joueur` joueur, `Carte *pMainJoueur`, `Carte pli[]`, `Joueur` dernierVainqueur, `Couleur` atout, `int` carteRestante)
proposition de la pose d'une carte par l'IA

4.16.1 Description détaillée

fichier header contenant les prototypes des fonctions relative au intellegences artificielles

Auteur

Carlo.A & Florian.C

Version

v1.0

Date

12 juin 2020

4.16.2 Documentation des fonctions

4.16.2.1 choixCartelA()

```
int choixCarteIA (
    Joueur joueur,
    Carte * pMainJoueur,
    Carte pli[],
    Joueur premierJoueur,
    Couleur atout,
    int carteRestante )
```

proposition de la pose d'une carte par l'IA

Paramètres

<i>Joueur</i>	joueur : determine qu'elle IA doit jouer
<i>Carte</i>	*pMainJoueur : pointeur vers le tableau qui enregistre la main de l'IA
<i>Carte</i>	pli[] : tableau qui enregistre les cartes jouées pendant le pli
<i>Joueur</i>	dernierVainqueur : defini quel joueur à posé la première carte
<i>Couleur</i>	atout : defini quelle couleur est en atout
<i>int</i>	carteRestante : nombre de cartes resantes en main

Renvoie

```
debug for(int i=0;i<nbCarteValide;i++){ printf("|%d| \n",numCarteValide[i]); } afficheMain(pCarteValidee);
```

```
debug for(int i=0;i<offset;i++){ printf("|%d| \n",numCarteGagnante[i]); } afficheMain(pCarteGagante);
```

Définition à la ligne 60 du fichier ia.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.16.2.2 proposeContratIa()

```
Contrat proposeContratIa (
    Joueur parle,
    Carte * pCarteMain,
    Contrat dernierContrat )
```

proposition par l'ia d'un contrat

Paramètres

<i>Joueur</i>	parle : joueur qui parle
<i>Carte</i>	*pCarteMain pointeur sur le tableau qui contient les cartes du joueur
<i>Contrat</i>	dernierContrat : dernier contrat qui a été proposé

Renvoie

Contrat

< Permet de déterminer de manière heuristique la meilleure valeur de seuilMinPrise et seuilMaxPrise

< ON coinche !

Définition à la ligne 11 du fichier ia.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.17 Référence du fichier main.c

fichier contenant la fonction main

```
#include "main.h"
```

Grappe des dépendances par inclusion de main.c:

Fonctions

— int *main* (int argc, char *argv[])
Entrée du programme.

4.17.1 Description détaillée

fichier contenant la fonction main

Auteur

Carlo.A & Florian.C

Version

v1.0

Date

12 juin 2020

4.17.2 Documentation des fonctions

4.17.2.1 main()

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

Entrée du programme.

Auteur

Florian.C

Renvoie

EXIT_SUCCESS - Arrêt normal du programme

< CODE FINAL

< definit la taille de la fenetre a 50 lignes et 91 colones

< FIN CODE FINAL

Définition à la ligne 19 du fichier main.c.

Voici le graphe d'appel pour cette fonction :

4.18 Référence du fichier main.h

fichier la déclaration des constantes, les énumérations, les structures et l'inclusion des headers du projet

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
#include <time.h>
#include <string.h>
#include "tableau.h"
#include "acquisition.h"
#include "affichage.h"
#include "gestionCarte.h"
#include "general.h"
#include "formatageChaine.h"
#include "autre.h"
#include "ia.h"
#include "sous-menus.h"
#include "gestionFichier.h"
```

Graphique des dépendances par inclusion de main.h: Ce graphique montre quels fichiers incluent directement ou indirectement ce fichier :

Structures de données

- struct [Carte](#)
représente une carte d'un jeu de 32 cartes
- struct [Contrat](#)
représente un contrat à la belote coïncide

Macros

- #define [TAILLE_MAXI_PSEUDO](#) 20
- #define [TAILLE_MAXI_COULEUR](#) 8
- #define [NB_JOUEUR](#) 4
- #define [TAILLE_MAXI_MESSAGE](#) 500
- #define [NB_TOTAL_CARTE](#) 32
- #define [NIVEAU_IA](#) 2
- #define [NB_CARACTERE_SCORE](#) 33
- #define [POSITION_NB_VICTOIRE](#) 21
- #define [POSITION_SCORE_MAX](#) 25
- #define [POSITION_NB_MANCHES_POUR_GAGNER](#) 30
- #define [NB_CARACTERE_LEADERBOARD](#) 26
- #define [POSITION_RECORD_VICTOIRE](#) 21
- #define [DEBUG_MODE](#) 0
- #define [MODE_1_MANCHE](#) 0

Définitions de type

- typedef enum [Couleur](#) [Couleur](#)
- typedef enum [Valeur](#) [Valeur](#)
- typedef enum [Joueur](#) [Joueur](#)
- typedef enum [Coinche](#) [Coinche](#)
- typedef enum [NbPoint](#) [NbPoint](#)
- typedef struct [Carte](#) [Carte](#)
- typedef struct [Contrat](#) [Contrat](#)
- typedef enum [TypeMessage](#) [TypeMessage](#)

Énumérations

- enum `Couleur` {
`SANS_COULEUR` = 0, `COEUR` = 1, `PIQUE` = 2, `CARREAU` = 3,
`TREFLE` = 4, `TOUT_ATOUT` = 5, `SANS_ATOUT` = 6 }
Les différentes couleurs d'atout possible à la belote coincée.
- enum `Valeur` {
`SANS_VALEUR` = 0, `AS` = 1, `VALET` = 11, `DAME` = 12,
`ROI` = 13, `SEPT` = 7, `HUIT` = 8, `NEUF` = 9,
`DIX` = 10 }
Les différentes valeurs possibles des cartes d'un jeu de 32 cartes.
- enum `Joueur` {
`SANS_JOUEUR` = 0, `NORD` = 1, `EST` = 2, `SUD` = 3,
`OUEST` = 4 }
Les différents joueurs lors d'une partie.
- enum `Coinche` { `NORMAL` = 0, `COINCHE` = 1, `SURCOINCHE` = 2 }
indique si un contrat est coincé ou surcoincé
- enum `NbPoint` {
`ZERO` = 0, `QUATRE_VINGT` = 80, `QUATRE_VINGT_DIX` = 90, `CENT` = 100,
`CENT_DIX` = 110, `CENT_VINGT` = 120, `CENT_TRENTE` = 130, `CENT_QUARANTE` = 140,
`CENT_CINQUANTE` = 150, `CENT_SOIXANTE` = 160, `CAPOT` = 170, `GENERALE` = 180 }
- enum `TypeMessage` { `SANS_MESSAGE`, `POSE_CARTE`, `RESULTAT_PLI`, `RESULTAT_MANCHE` }
contente à passer en paramètre à la fonction `gagnerMessage()` pour donner le type de message à gérer

4.18.1 Description détaillée

fichier la déclaration des constantes, les énumérations, les structures et l'inclusion des headers du projet

Auteur

Carlo.A & Florian.C

Version

v1.0

Date

12 juin 2020

4.18.2 Documentation des macros

4.18.2.1 `DEBUG_MODE`

```
#define DEBUG_MODE 0
```

0 pour le mode normal, 1 pour le mode debug

Définition à la ligne 26 du fichier main.h.

4.18.2.2 MODE_1_MANCHE

```
#define MODE_1_MANCHE 0
```

0 pour le mode normal, 1 pour le mode où une partie est constituée d'une unique manche Structure et Annotations

Définition à la ligne 29 du fichier main.h.

4.18.2.3 NB_CARRACTERE_LEADERBOARD

```
#define NB_CARRACTERE_LEADERBOARD 26
```

Définition à la ligne 24 du fichier main.h.

4.18.2.4 NB_CARRACTERE_SCORE

```
#define NB_CARRACTERE_SCORE 33
```

Définition à la ligne 20 du fichier main.h.

4.18.2.5 NB_JOUEUR

```
#define NB_JOUEUR 4
```

Définition à la ligne 16 du fichier main.h.

4.18.2.6 NB_TOATAL_CARTE

```
#define NB_TOATAL_CARTE 32
```

Définition à la ligne 18 du fichier main.h.

4.18.2.7 NIVEAU_IA

```
#define NIVEAU_IA 2
```

Définition à la ligne 19 du fichier main.h.

4.18.2.8 POSITION_NB_MANCHES_POUR_GAGNER

```
#define POSITION_NB_MANCHES_POUR_GAGNER 30
```

Définition à la ligne 23 du fichier main.h.

4.18.2.9 POSITION_NB_VICTOIRE

```
#define POSITION_NB_VICTOIRE 21
```

Définition à la ligne 21 du fichier main.h.

4.18.2.10 POSITION_RECORD_VICTOIRE

```
#define POSITION_RECORD_VICTOIRE 21
```

Définition à la ligne 25 du fichier main.h.

4.18.2.11 POSITION_SCORE_MAX

```
#define POSITION_SCORE_MAX 25
```

Définition à la ligne 22 du fichier main.h.

4.18.2.12 TAILLE_MAXI_COULEUR

```
#define TAILLE_MAXI_COULEUR 8
```

Définition à la ligne 15 du fichier main.h.

4.18.2.13 TAILLE_MAXI_MESSAGE

```
#define TAILLE_MAXI_MESSAGE 500
```

Définition à la ligne 17 du fichier main.h.

4.18.2.14 TAILLE_MAXI_PESEUDO

```
#define TAILLE_MAXI_PESEUDO 20
```

< Constantes

Définition à la ligne 14 du fichier main.h.

4.18.3 Documentation des définitions de type

4.18.3.1 Carte

```
typedef struct Carte Carte
```

4.18.3.2 Coinche

```
typedef enum Coinche Coinche
```

4.18.3.3 Contrat

```
typedef struct Contrat Contrat
```

4.18.3.4 Couleur

```
typedef enum Couleur Couleur
```

4.18.3.5 Joueur

```
typedef enum Joueur Joueur
```

4.18.3.6 NbPoint

```
typedef enum NbPoint NbPoint
```

4.18.3.7 TypeMessage

```
typedef enum TypeMessage TypeMessage
```

Fontions standard Fontions cr    e dans le cardre du projet

4.18.3.8 Valeur

```
typedef enum Valeur Valeur
```

4.18.4 Documentation du type de l'  num  ration

4.18.4.1 Coinche

```
enum Coinche
```

indique il un contrat est coinch   ou surcoinch  

Valeurs   num  r  es

NORMAL	
COINCHE	Le contrat n'est pas coinch��, par d��faut
SURCOINCHE	Le contrat est Coinch��

D  finition    la ligne 97 du fichier main.h.

4.18.4.2 Couleur

```
enum Couleur
```

Les diff  rentes couleur d'atout possible a la belote coinch  .

toute les couleur disponible ainsi que tout atouts et sans atout sert a la fois pour le type [Carte](#) et le type [Contrat](#)

Valeurs   num  r  es

SANS_COULEUR	
COEUR	Couleur par d��faut, pour une carte vide
PIQUE	Couleur c��ur
CARREAU	Couleur pique
TREFLE	Couleur carreau
TOUT_ATOUT	Couleur trefle
SANS_ATOUT	Tout atout

Définition à la ligne 39 du fichier main.h.

4.18.4.3 Joueur

enum `Joueur`

Les différents joueurs lors d'une partie.

Les joueurs sont nommés par les différents points cardinaux correspondant à leur placement sur la table

Valeurs énumérées

SANS_JOUEUR	
NORD	Joueur par défaut, quand il n'y a pas de joueur ou pour l'initialisation
EST	Joueur en haut
SUD	Joueur à gauche
OUEST	Joueur en bas

Définition à la ligne 81 du fichier main.h.

4.18.4.4 NbPoint

enum `NbPoint`

Valeurs énumérées

ZERO	
QUATRE_VINGT	Le contrat vaut zéro point, par défaut correspond à un joueur qui passe ou pour l'initialisation
QUATRE_VINGT_DIX	Le contrat vaut 80 points
CENT	Le contrat vaut 90 points
CENT_DIX	Le contrat vaut 100 points
CENT_VINGT	Le contrat vaut 110 points
CENT_TRENTE	Le contrat vaut 120 points
CENT_QUARANTE	Le contrat vaut 130 points
CENT_CINQUANTE	Le contrat vaut 140 points
CENT_SOIXANTE	Le contrat vaut 150 points
CAPOT	Le contrat vaut 160 points
GENERALE	Le contrat est un capot

Définition à la ligne 112 du fichier main.h.

4.18.4.5 TypeMessage

enum `TypeMessage`

contante à passer en paramètre à la fonction `getreMessage()` pour donner le type de message à gérer

Valeurs énumérées

SANS_MESSAGE	
POSE_CARTE	pas de message à afficher, par défaut
RESULTAT_PLI	lorsqu'un joueur vient de poser une carte
RESULTAT_MANCHE	annonce le vainqueur d'un pli

Définition à la ligne 164 du fichier `main.h`.

4.18.4.6 Valeur

enum `Valeur`

Les différentes valeurs possibles des cartes d'un jeu de 32 cartes.

Les valeurs associées ont été placées de manière arbitraire et n'ont pas d'importance sur le programme. Pour simplifier, le tout atout et le sans atout sont considérés comme des couleurs.

Valeurs énumérées

SANS_VALEUR	
AS	Valeur par défaut, pour une carte vide
VALET	Valeur As
DAME	Valeur Valet
ROI	Valeur Dame
SEPT	Valeur Roi
HUIT	Valeur 7
NEUF	Valeur 8
DIX	Valeur 9

Définition à la ligne 60 du fichier `main.h`.

4.19 Référence du fichier sous-menus.c

```
#include "main.h"
```

Graphe des dépendances par inclusion de `sous-menus.c`:

Fonctions

- void `parametre` (char *pseudo[])
affiche le sous menu des paramètre, permet de changer le psedo de tout les ordinateur
- int `leaderboard` (FILE *fichier)
fonction qui recupère les meilleurs score dans un fichier et les affiche dans la console
- int `statistiqueJoueur` (FILE *fichier, int ligne)
fonction qui recupère les statistiques du joueur et les affiche dans la console

4.19.1 Documentation des fonctions

4.19.1.1 `leaderboard()`

```
int leaderboard (
    FILE * fichier )
```

fonction qui recupère les meilleurs score dans un fichier et les affiche dans la console

Paramètres

<i>pointeur</i>	vers le fichier à lire
-----------------	------------------------

Renvoie

0

Définition à la ligne 32 du fichier sous-menus.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.19.1.2 `parametre()`

```
void parametre (
    char * pseudo[ ] )
```

affiche le sous menu des paramètre, permet de changer le psedo de tout les ordinateur

Paramètres

<i>char</i>	*pseudo[] : tableau de pointeurs contenant les pseudos des différents joueurs
-------------	---

Renvoie

void

Définition à la ligne 11 du fichier sous-menus.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.19.1.3 statistiqueJoueur()

```
int statistiqueJoueur (
    FILE * fichier,
    int ligne )
```

fonction qui recupère les statistiques du joueur et les affiche dans la console

Paramètres

<i>pointeur</i>	vers le fichier de sauvegarde des scores
<i>numero</i>	de la ligne à laquelle sont ecrites les statistiques du joueur

Renvoie

0

Définition à la ligne 85 du fichier sous-menus.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.20 Référence du fichier sous-menus.h

Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :

Fonctions

- void [parametre](#) (char *pseudo[])
affiche le sous menu des paramètre, permet de changer le psedo de tout les ordinateur
- int [leaderboard](#) (FILE *fichier)
fonction qui recupère les meilleurs score dans un fichier et les affiche dans la console
- int [statistiqueJoueur](#) (FILE *fichier, int ligne)
fonction qui recupère les statistiques du joueur et les affiche dans la console

4.20.1 Documentation des fonctions

4.20.1.1 leaderboard()

```
int leaderboard (
    FILE * fichier )
```

fonction qui recupère les meilleurs score dans un fichier et les affiche dans la console

Paramètres

<i>pointeur</i>	vers le fichier à lire
-----------------	------------------------

Renvoie

0

Définition à la ligne 32 du fichier sous-menus.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.20.1.2 parametre()

```
void parametre (
    char * pseudo[ ] )
```

affiche le sous menu des paramètre, permet de changer le psedo de tout les ordinateur

Paramètres

<i>char</i>	*pseudo[] : tableau de pointeurs contenant les pseudos des différents joueurs
-------------	---

Renvoie

void

Définition à la ligne 11 du fichier sous-menus.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.20.1.3 statistiqueJoueur()

```
int statistiqueJoueur (
    FILE * fichier,
    int ligne )
```

fonction qui recupère les statistiques du joueur et les affiche dans la console

Paramètres

<i>pointeur</i>	vers le fichier de sauvegarde des scores
<i>numero</i>	de la ligne à laquelle sont ecrites les statistiques du joueur

Renvoie

0

Définition à la ligne 85 du fichier sous-menus.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.21 Référence du fichier tableau.c

fichier source contenant les fonctions relative à la gestion des tableau

```
#include "main.h"
```

Graphe des dépendances par inclusion de tableau.c:

Fonctions

- int [sommeTableau](#) (int tableau[], int tailleTableau)
fait la somme des valeur d'un tableau contenant des entiers
- float [moyenneTableau](#) (int tableau[], int tailleTableau)
fait la moyenne des valeur d'un tableau contenant des entiers
- void [copie](#) (int tableauOriginal[], int tableauCopie[], int tailleTableau)
copie un tableau dans un autre
- int [mini](#) (int tableau[], int tailleTableau)
trouve la valeur minimum d'un tableau
- int [maxi](#) (int tableau[], int tailleTableau)
trouve la valeur maximum d'un tableau
- void [zeroSiSuperieur](#) (int tableau[], int tailleTableau, int maximum)
met a 0 toute les valeur d'un tableau qui sont superieur a un maximum
- void [ordonnerTableau](#) (int tableau[], int tailleTableau)
classe les valeur d'un tableau dans l'orde croissant
- void [afficheTableau](#) (int tableau[], int tailleTableau)
affiche un tableau
- void [acquiertTableau](#) (int tableau[], int tailleTableau)
demande a l'utilisateur de saisir les valeurs d'un tableau
- void [constanteTableau](#) (int tableau[], int tailleTableau, int valeur)
donne a toute les valeur d'un tableau une constante
- int [rechercheDichotomie](#) (int tableau[], int tailleTableau, int nCherche)
cherche par dichotomie si une valeur est au moin une fois dans le tableau
- void [aleatoireTableau](#) (int tableau[], int tailleTableau, int [mini](#), int [maxi](#))
donne des nombres aléatoire au valeur d'un tableau

4.21.1 Description détaillée

fichier source contenant les fonctions relative à la gestion des tableau

Auteur

Carlo.A & Florian.C

Version

v1.0

Date

12 juin 2020

4.21.2 Documentation des fonctions

4.21.2.1 acquiertTableau()

```
void acquiertTableau (
    int tableau[],
    int tailleTableau )
```

demande a l'utilisateur de saisir les valeurs d'un tableau

Paramètres

<i>tableau[]</i>	: nom du tableau a saisir
<i>tailleTableau</i>	: la taille du tableau, entier positif

Renvoie

void

Définition à la ligne 93 du fichier tableau.c.

4.21.2.2 afficheTableau()

```
void afficheTableau (
    int tableau[],
    int tailleTableau )
```

affiche un tableau

Paramètres

<i>tableau[]</i>	: le tableau a afficher
<i>tailleTableau</i>	: la taille du tableau, entier positif

Renvoie

void

Définition à la ligne 83 du fichier tableau.c.

4.21.2.3 aleatoireTableau()

```
void aleatoireTableau (
    int tableau[],
    int tailleTableau,
    int mini,
    int maxi )
```

donne des nombres aléatoire au valeur d'un tableau

Paramètres

<i>tableau[]</i>	: tableau a modifier
<i>tailleTableau</i>	: taille du tableau entier positif
<i>mini</i>	: valeur minimum des nombres aléatoires
<i>maxi</i>	: valeur maximum des nombres aléatoires

Renvoie

Définition à la ligne 127 du fichier `tableau.c`.

Voici le graphe d'appel pour cette fonction :

4.21.2.4 `constanteTableau()`

```
void constanteTableau (
    int  tableau[],
    int  tailleTableau,
    int  valeur )
```

donne a toute les valeur d'un tableau une constante

Paramètres

<i>tableau[]</i>	: tableau a modifier
<i>tailleTableau</i>	: taille du tableau entier positif
<i>valeur</i>	: valeur a donner a toutes les valeur du tableau

Renvoie

`void`

Définition à la ligne 101 du fichier `tableau.c`.

4.21.2.5 `copie()`

```
void copie (
    int  tableauOriginal[],
    int  tableauCopie[],
    int  tailleTableau )
```

copie un tableau dans un autre

Paramètres

<i>tableauOriginal[]</i>	: tableaaau que l'on veut copier
<i>tableauCopie[]</i>	: tableau dans lequel on copie tableauOriaginal
<i>tailleTableau</i>	: la taille du tableau, entier positif

Renvoie

`void`

Définition à la ligne 25 du fichier `tableau.c`.

4.21.2.6 maxi()

```
int maxi (
    int tableau[],
    int tailleTableau )
```

trouve la valeur maximum d'un tableau

Paramètres

<i>tableau</i>	[] : tableau dont on veut travailler
<i>tailleTableau</i>	: la taille du tableau, entier positif

Renvoie

int valeur la plus grande du tableau

Définition à la ligne 45 du fichier tableau.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.21.2.7 mini()

```
int mini (
    int tableau[],
    int tailleTableau )
```

trouve la valeur minimum d'un tableau

Paramètres

<i>tableau</i>	[] : tableau dont on veut travailler
<i>tailleTableau</i>	: la taille du tableau, entier positif

Renvoie

int valeur la plus petite du tableau

Définition à la ligne 33 du fichier tableau.c.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.21.2.8 moyenneTableau()

```
float moyenneTableau (
    int tableau[],
    int tailleTableau )
```

fait la moyenne des valeur d'un tableau contenant des entiers

Paramètres

<i>tableau[]</i>	: le nom du tableau
<i>tailleTableau</i>	: la taille du tableau, entier positif

Renvoie

la moyenne des valeur du tableau, réel

Définition à la ligne 20 du fichier `tableau.c`.

Voici le graphe d'appel pour cette fonction :

4.21.2.9 ordonnerTableau()

```
void ordonnerTableau (
    int tableau[],
    int tailleTableau )
```

classe les valeur d'un tableau dans l'orde croissant

Paramètres

<i>tableau[]</i>	: le tableau à classer
<i>tailleTableau</i>	: la taille du tableau, entier positif

Renvoie

void

Définition à la ligne 66 du fichier `tableau.c`.

4.21.2.10 rechercheDichotomie()

```
int rechercheDichotomie (
    int tableau[],
    int tailleTableau,
    int nCherche )
```

chercherche par dicotomie si une valeur est au moin une fois dans le tableau

Paramètres

<i>tableau[]</i>	: tableau dans lequel on fait la recherche
<i>tailleTableau</i>	: taille du tableau entier positif
<i>nCherche</i>	nombre que l'on cherche dans le tableau

Renvoie

int : 1 si on a trouvé, 0 sinon

Définition à la ligne 108 du fichier `tableau.c`.

4.21.2.11 sommeTableau()

```
int sommeTableau (
    int tableau[],
    int tailleTableau )
```

fait la somme des valeur d'un tableau contenant des entiers

Paramètres

<i>tableau[]</i>	: le nom du tableau
<i>tailleTableau</i>	: la taille du tableau, entier positif

Renvoie

la somme des valeur du table entier

Définition à la ligne 11 du fichier `tableau.c`.

Voici le graphe des appelants de cette fonction :

4.21.2.12 zeroSiSuperieur()

```
void zeroSiSuperieur (
    int tableau[],
    int tailleTableau,
    int maximum )
```

met a 0 toute les valeur d'un tableau qui sont superieur a un maximum

Paramètres

<i>tableau[]</i>	: tableau sur lequel on veut travailler
<i>tailleTableau</i>	: la taille du tableau, entier positif
<i>maximum</i>	: valeur maximum au dessus de quoi la valeur est remise a 0

Renvoie

void

Définition à la ligne 57 du fichier `tableau.c`.

4.22 Référence du fichier tableau.h

fichier header contenant les prototypes des fonctions relative à la gestion des tableau

Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :

Fonctions

- int `sommeTableau` (int tableau[], int tailleTableau)
fait la somme des valeur d'un tableau contenant des entiers
- float `moyenneTableau` (int tableau[], int tailleTableau)
fait la moyenne des valeur d'un tableau contenant des entiers
- void `copie` (int tableauOriginal[], int tableauCopie[], int tailleTableau)
copie un tableau dans un autre
- int `mini` (int tableau[], int tailleTableau)
trouve la valeur minimum d'un tableau
- int `maxi` (int tableau[], int tailleTableau)
trouve la valeur maximum d'un tableau
- void `zeroSiSuperieur` (int tableau[], int tailleTableau, int maximum)
met a 0 toute les valeur d'un tableau qui sont superieur a un maximum
- void `ordonnerTableau` (int tableau[], int tailleTableau)
classe les valeur d'un tableau dans l'orde croissant
- void `afficheTableau` (int tableau[], int tailleTableau)
affiche un tableau
- void `acquierteTableau` (int tableau[], int tailleTableau)
demande a l'utilisateur de saisir les valeurs d'un tableau
- void `constanteTableau` (int tableau[], int tailleTableau, int valeur)
donne a toute les valeur d'un tableau une constante
- void `aleatoireTableau` (int tableau[], int tailleTableau, int mini, int maxi)
donne des nombres aléatoire au valeur d'un tableau
- int `rechercheDichotomie` (int tableau[], int tailleTableau, int nCherche)
cherche par dichotomie si une valeur est au moins une fois dans le tableau

4.22.1 Description détaillée

fichier header contenant les prototypes des fonctions relative à la gestion des tableau

Auteur

Carlo.A & Florian.C

Version

v1.0

Date

12 juin 2020

4.22.2 Documentation des fonctions

4.22.2.1 `acquierteTableau()`

```
void acquierteTableau (
    int tableau[],
    int tailleTableau )
```

demande a l'utilisateur de saisir les valeurs d'un tableau

Paramètres

<i>tableau[]</i>	: nom du tableau a saisir
<i>tailleTableau</i>	: la taille du tableau, entier positif

Renvoie

void

Définition à la ligne 93 du fichier tableau.c.

4.22.2.2 afficheTableau()

```
void afficheTableau (
    int tableau[],
    int tailleTableau )
```

affiche un tableau

Paramètres

<i>tableau[]</i>	: le tableau a afficher
<i>tailleTableau</i>	: la taille du tableau, entier positif

Renvoie

void

Définition à la ligne 83 du fichier tableau.c.

4.22.2.3 aleatoireTableau()

```
void aleatoireTableau (
    int tableau[],
    int tailleTableau,
    int mini,
    int maxi )
```

donne des nombres aléatoire au valeur d'un tableau

Paramètres

<i>tableau[]</i>	: tableau a modifier
<i>tailleTableau</i>	: taille du tableau entier positif
<i>mini</i>	: valeur minimum des nombres aléatoires
<i>maxi</i>	: valeur maximum des nombres aléatoires

Renvoie

Définition à la ligne 127 du fichier `tableau.c`.

Voici le graphe d'appel pour cette fonction :

4.22.2.4 `constanteTableau()`

```
void constanteTableau (
    int  tableau[],
    int  tailleTableau,
    int  valeur )
```

donne a toute les valeur d'un tableau une constante

Paramètres

<i>tableau[]</i>	: tableau a modifier
<i>tailleTableau</i>	: taille du tableau entier positif
<i>valeur</i>	: valeur a donner a toutes les valeur du tableau

Renvoie

`void`

Définition à la ligne 101 du fichier `tableau.c`.

4.22.2.5 `copie()`

```
void copie (
    int  tableauOriginal[],
    int  tableauCopie[],
    int  tailleTableau )
```

copie un tableau dans un autre

Paramètres

<i>tableauOriginal[]</i>	: tableaau que l'on veut copier
<i>tableauCopie[]</i>	: tableau dans lequel on copie tableauOriaginal
<i>tailleTableau</i>	: la taille du tableau, entier positif

Renvoie

`void`

Définition à la ligne 25 du fichier `tableau.c`.

4.22.2.6 maxi()

```
int maxi (
    int tableau[],
    int tailleTableau )
```

trouve la valeur maximum d'un tableau

Paramètres

<i>tableau</i>	[] : tableau dont on veut travailler
<i>tailleTableau</i>	: la taille du tableau, entier positif

Renvoie

int valeur la plus grande du tableau

Définition à la ligne 45 du fichier `tableau.c`.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.22.2.7 mini()

```
int mini (
    int tableau[],
    int tailleTableau )
```

trouve la valeur minimum d'un tableau

Paramètres

<i>tableau</i>	[] : tableau dont on veut travailler
<i>tailleTableau</i>	: la taille du tableau, entier positif

Renvoie

int valeur la plus petite du tableau

Définition à la ligne 33 du fichier `tableau.c`.

Voici le graphe d'appel pour cette fonction : Voici le graphe des appelants de cette fonction :

4.22.2.8 moyenneTableau()

```
float moyenneTableau (
    int tableau[],
    int tailleTableau )
```

fait la moyenne des valeur d'un tableau contenant des entiers

Paramètres

<code>tableau[]</code>	: le nom du tableau
<code>tailleTableau</code>	: la taille du tableau, entier positif

Renvoie

la moyenne des valeur du tableau, réel

Définition à la ligne 20 du fichier `tableau.c`.

Voici le graphe d'appel pour cette fonction :

4.22.2.9 ordonnerTableau()

```
void ordonnerTableau (
    int tableau[],
    int tailleTableau )
```

classe les valeur d'un tableau dans l'orde croissant

Paramètres

<code>tableau[]</code>	: le tableau à classer
<code>tailleTableau</code>	: la taille du tableau, entier positif

Renvoie

void

Définition à la ligne 66 du fichier `tableau.c`.

4.22.2.10 rechercheDichotomie()

```
int rechercheDichotomie (
    int tableau[],
    int tailleTableau,
    int nCherche )
```

chercherche par dicotomie si une valeur est au moin une fois dans le tableau

Paramètres

<code>tableau[]</code>	: tableau dans lequel on fait la recherche
<code>tailleTableau</code>	: taille du tableau entier positif
<code>nCherche</code>	nombre que l'on cherche dans le tableau

Renvoie

int : 1 si on a trouvé, 0 sinon

Définition à la ligne 108 du fichier tableau.c.

4.22.2.11 sommeTableau()

```
int sommeTableau (
    int tableau[],
    int tailleTableau )
```

fait la somme des valeur d'un tableau contenant des entiers

Paramètres

<i>tableau[]</i>	: le nom du tableau
<i>tailleTableau</i>	: la taille du tableau, entier positif

Renvoie

la somme des valeur du table entier

Définition à la ligne 11 du fichier tableau.c.

Voici le graphe des appelants de cette fonction :

4.22.2.12 zeroSiSuperieur()

```
void zeroSiSuperieur (
    int tableau[],
    int tailleTableau,
    int maximum )
```

met a 0 toute les valeur d'un tableau qui sont superieur a un maximum

Paramètres

<i>tableau[]</i>	: tableau sur lequel on veut travailler
<i>tailleTableau</i>	: la taille du tableau, entier positif
<i>maximum</i>	: valeur maximum au dessus de quoi la valeur est remise a 0

Renvoie

void

Définition à la ligne 57 du fichier tableau.c.

Index

acquierTableau
 tableau.c, 89
 tableau.h, 95
acquisition.c, 9
acquisition.h, 9
 acquisitionEntierAvecMessage, 10
 acquisitionEntierSansMessage, 10
 acquisitionEntierSansMessageAvecConsigne, 10
 acquisitionEntierSecurise, 11
 acquisitionOuiNonSansMessage, 11
 acquisitionPseudoAvecMessage, 11
acquisitionEntierAvecMessage
 acquisition.h, 10
acquisitionEntierSansMessage
 acquisition.h, 10
acquisitionEntierSansMessageAvecConsigne
 acquisition.h, 10
acquisitionEntierSecurise
 acquisition.h, 11
acquisitionOuiNonSansMessage
 acquisition.h, 11
acquisitionPseudoAvecMessage
 acquisition.h, 11
affichage.c, 12
 afficheContrat, 13
 afficheInterfacePli, 13
 afficheMain, 14
 afficheMenuPrincipal, 14
 afficheMenuSelection, 15
 afficheSousMenus, 15
 modifieTailleFenetre, 16
 proposeContratUtilisateur, 16
affichage.h, 17
 afficheContrat, 18
 afficheInterfacePli, 18
 afficheMain, 19
 afficheMenuPrincipal, 19
 afficheMenuSelection, 20
 afficheSousMenus, 20
 modifieTailleFenetre, 21
 proposeContratUtilisateur, 21
afficheContrat
 affichage.c, 13
 affichage.h, 18
afficheInterfacePli
 affichage.c, 13
 affichage.h, 18
afficheMain
 affichage.c, 14
 affichage.h, 19
afficheMenuPrincipal
 affichage.c, 14
 affichage.h, 19
afficheMenuSelection
 affichage.c, 15
 affichage.h, 20
afficheSousMenus
 affichage.c, 15
 affichage.h, 20
afficheTableau
 tableau.c, 90
 tableau.h, 96
ajusteEchelle
 autre.c, 23
 autre.h, 26
aleatoireTableau
 tableau.c, 90
 tableau.h, 96
aligneModifieChaine
 formatageChaine.c, 30
 formatageChaine.h, 35
annonceContrat
 general.c, 41
 general.h, 48
AS
 main.h, 85
atout
 Contrat, 6
autre.c, 22
 ajusteEchelle, 23
 joue1000Partie, 23
 joueurSuivant, 23
 nbAleatoire, 24
 pointPli, 24
 setContrat, 25
autre.h, 25
 ajusteEchelle, 26
 joue1000Partie, 27
 joueurSuivant, 27
 nbAleatoire, 27
 pointPli, 28
 setContrat, 28
calculPointManche
 general.c, 42
 general.h, 48
CAPOT
 main.h, 84
CARREAU

- main.h, 83
- Carte, 5
 - couleur, 5
 - main.h, 82
 - valeur, 5
- cartePlaceAvant
 - gestionCarte.c, 54
 - gestionCarte.h, 62
- carteValide
 - gestionCarte.c, 54
 - gestionCarte.h, 63
- CENT
 - main.h, 84
- CENT_CINQUANTE
 - main.h, 84
- CENT_DIX
 - main.h, 84
- CENT_QUARANTE
 - main.h, 84
- CENT_SOIXANTE
 - main.h, 84
- CENT_TRENTE
 - main.h, 84
- CENT_VINGT
 - main.h, 84
- centreChaine
 - formatageChaine.c, 30
 - formatageChaine.h, 36
- centreModifieChaine
 - formatageChaine.c, 30
 - formatageChaine.h, 36
- choixCarteIA
 - ia.c, 73
 - ia.h, 75
- COEUR
 - main.h, 83
- COINCHE
 - main.h, 83
- Coinche
 - main.h, 82, 83
- coinche
 - Contrat, 6
- constanteTableau
 - tableau.c, 91
 - tableau.h, 97
- Contrat, 6
 - atout, 6
 - coinche, 6
 - main.h, 82
 - nbPoint, 7
 - preneur, 7
- copie
 - tableau.c, 91
 - tableau.h, 97
- Couleur
 - main.h, 82, 83
- couleur
 - Carte, 5
- DAME
 - main.h, 85
- DEBUG_MODE
 - main.h, 79
- decoupeChaine
 - formatageChaine.c, 31
 - formatageChaine.h, 37
- distribueCarte
 - gestionCarte.c, 55
 - gestionCarte.h, 64
- DIX
 - main.h, 85
- ecrireLeaderboard
 - gestionFichier.c, 69
 - gestionFichier.h, 71
- ecrireStatistique
 - gestionFichier.c, 70
 - gestionFichier.h, 72
- ecriturePseudo
 - gestionFichier.c, 70
 - gestionFichier.h, 72
- EST
 - main.h, 84
- forceCarte
 - gestionCarte.c, 56
 - gestionCarte.h, 64
- formatageChaine.c, 29
 - aligneModifieChaine, 30
 - centreChaine, 30
 - centreModifieChaine, 30
 - decoupeChaine, 31
 - formateCarte, 31
 - formateContrat, 32
 - formatePseudo, 32
 - genereMessage, 33
 - rempliEspace, 34
 - stockeInfoCarte, 34
- formatageChaine.h, 35
 - aligneModifieChaine, 35
 - centreChaine, 36
 - centreModifieChaine, 36
 - decoupeChaine, 37
 - formateCarte, 37
 - formateContrat, 38
 - formatePseudo, 38
 - genereMessage, 39
 - rempliEspace, 39
 - stockeInfoCarte, 40
- formateCarte
 - formatageChaine.c, 31
 - formatageChaine.h, 37
- formateContrat
 - formatageChaine.c, 32
 - formatageChaine.h, 38
- formatePseudo
 - formatageChaine.c, 32
 - formatageChaine.h, 38

- general.c, 40
 - annonceContrat, 41
 - calculPointManche, 42
 - initialisation, 42
 - manche, 43
 - menuPrincipal, 44
 - nouvellePartie, 44
 - pli, 45
 - poseCarte, 46
 - proposeContrat, 46
- general.h, 47
 - annonceContrat, 48
 - calculPointManche, 48
 - initialisation, 49
 - manche, 49
 - menuPrincipal, 50
 - nouvellePartie, 50
 - pli, 51
 - poseCarte, 52
 - proposeContrat, 52
- GENERALE
 - main.h, 84
- genereMessage
 - formatageChaine.c, 33
 - formatageChaine.h, 39
- gestionCarte.c, 53
 - cartePlaceAvant, 54
 - carteValide, 54
 - distribueCarte, 55
 - forceCarte, 56
 - rechercheAnnonce, 56
 - rechercherCarte, 57
 - rechercherCarteSuperieur, 57
 - setCarte, 59
 - sommeForceCarte, 59
 - supprimeCarte, 60
 - trieCarte, 60
 - vainqueurPli, 61
- gestionCarte.h, 61
 - cartePlaceAvant, 62
 - carteValide, 63
 - distribueCarte, 64
 - forceCarte, 64
 - rechercheAnnonce, 65
 - rechercherCarte, 65
 - rechercherCarteSuperieur, 66
 - setCarte, 66
 - sommeForceCarte, 67
 - supprimeCarte, 67
 - trieCarte, 68
 - vainqueurPli, 68
- gestionFichier.c, 69
 - ecrireLeaderboard, 69
 - ecrireStatistique, 70
 - ecriturePseudo, 70
- gestionFichier.h, 71
 - ecrireLeaderboard, 71
 - ecrireStatistique, 72
 - ecriturePseudo, 72
- HUIT
 - main.h, 85
- ia.c, 73
 - choixCarteIA, 73
 - proposeContratIA, 74
- ia.h, 74
 - choixCarteIA, 75
 - proposeContratIA, 76
- initialisation
 - general.c, 42
 - general.h, 49
- joue1000Partie
 - autre.c, 23
 - autre.h, 27
- Joueur
 - main.h, 82, 84
- joueurSuivant
 - autre.c, 23
 - autre.h, 27
- leaderboard
 - sous-menus.c, 86
 - sous-menus.h, 87
- main
 - main.c, 77
- main.c, 76
 - main, 77
- main.h, 78
 - AS, 85
 - CAPOT, 84
 - CARREAU, 83
 - Carte, 82
 - CENT, 84
 - CENT_CINQUANTE, 84
 - CENT_DIX, 84
 - CENT_QUARANTE, 84
 - CENT_SOIXANTE, 84
 - CENT_TRENTE, 84
 - CENT_VINGT, 84
 - COEUR, 83
 - COINCHE, 83
 - Coinche, 82, 83
 - Contrat, 82
 - Couleur, 82, 83
 - DAME, 85
 - DEBUG_MODE, 79
 - DIX, 85
 - EST, 84
 - GENERALE, 84
 - HUIT, 85
 - Joueur, 82, 84
 - MODE_1_MANCHE, 79
 - NB_CARACTERE_LEADERBOARD, 80
 - NB_CARACTERE_SCORE, 80

- NB_JOUEUR, 80
- NB_TOATAL_CARTE, 80
- NbPoint, 82, 84
- NEUF, 85
- NIVEAU_IA, 80
- NORD, 84
- NORMAL, 83
- OUEST, 84
- PIQUE, 83
- POSE_CARTE, 85
- POSITION_NB_MANCHES_POUR_GAGNER, 80
- POSITION_NB_VICTOIRE, 81
- POSITION_RECORD_VICTOIRE, 81
- POSITION_SCORE_MAX, 81
- QUATRE_VINGT, 84
- QUATRE_VINGT_DIX, 84
- RESULTAT_MANCHE, 85
- RESULTAT_PLI, 85
- ROI, 85
- SANS_ATOUT, 83
- SANS_COULEUR, 83
- SANS_JOUEUR, 84
- SANS_MESSAGE, 85
- SANS_VALEUR, 85
- SEPT, 85
- SUD, 84
- SURCOINCHE, 83
- TAILLE_MAXI_COULEUR, 81
- TAILLE_MAXI_MESSAGE, 81
- TAILLE_MAXI_PESUDO, 81
- TOUT_ATOUT, 83
- TREFLE, 83
- TypeMessage, 82, 84
- VALET, 85
- Valeur, 83, 85
- ZERO, 84
- manche
 - general.c, 43
 - general.h, 49
- maxi
 - tableau.c, 91
 - tableau.h, 97
- menuPrincipal
 - general.c, 44
 - general.h, 50
- mini
 - tableau.c, 92
 - tableau.h, 98
- MODE_1_MANCHE
 - main.h, 79
- modifieTailleFenetre
 - affichage.c, 16
 - affichage.h, 21
- moyenneTableau
 - tableau.c, 92
 - tableau.h, 98
- NB_CARRACTERE_LEADERBOARD
 - main.h, 80
- NB_CARRACTERE_SCORE
 - main.h, 80
- NB_JOUEUR
 - main.h, 80
- NB_TOATAL_CARTE
 - main.h, 80
- nbAleatoire
 - autre.c, 24
 - autre.h, 27
- NbPoint
 - main.h, 82, 84
- nbPoint
 - Contrat, 7
- NEUF
 - main.h, 85
- NIVEAU_IA
 - main.h, 80
- NORD
 - main.h, 84
- NORMAL
 - main.h, 83
- nouvellePartie
 - general.c, 44
 - general.h, 50
- ordonnerTableau
 - tableau.c, 93
 - tableau.h, 99
- OUEST
 - main.h, 84
- parametre
 - sous-menus.c, 86
 - sous-menus.h, 88
- PIQUE
 - main.h, 83
- pli
 - general.c, 45
 - general.h, 51
- pointPli
 - autre.c, 24
 - autre.h, 28
- POSE_CARTE
 - main.h, 85
- poseCarte
 - general.c, 46
 - general.h, 52
- POSITION_NB_MANCHES_POUR_GAGNER
 - main.h, 80
- POSITION_NB_VICTOIRE
 - main.h, 81
- POSITION_RECORD_VICTOIRE
 - main.h, 81
- POSITION_SCORE_MAX
 - main.h, 81
- preneur
 - Contrat, 7
- proposeContrat
 - general.c, 46

- general.h, 52
- proposeContratla
 - ia.c, 74
 - ia.h, 76
- proposeContratUtilisateur
 - affichage.c, 16
 - affichage.h, 21
- QUATRE_VINGT
 - main.h, 84
- QUATRE_VINGT_DIX
 - main.h, 84
- rechercheAnnonce
 - gestionCarte.c, 56
 - gestionCarte.h, 65
- rechercheDichotomie
 - tableau.c, 93
 - tableau.h, 99
- rechercherCarte
 - gestionCarte.c, 57
 - gestionCarte.h, 65
- rechercherCarteSuperieur
 - gestionCarte.c, 57
 - gestionCarte.h, 66
- rempliEspace
 - formatageChaine.c, 34
 - formatageChaine.h, 39
- RESULTAT_MANCHE
 - main.h, 85
- RESULTAT_PLI
 - main.h, 85
- ROI
 - main.h, 85
- SANS_ATOUT
 - main.h, 83
- SANS_COULEUR
 - main.h, 83
- SANS_JOUEUR
 - main.h, 84
- SANS_MESSAGE
 - main.h, 85
- SANS_VALEUR
 - main.h, 85
- SEPT
 - main.h, 85
- setCarte
 - gestionCarte.c, 59
 - gestionCarte.h, 66
- setContrat
 - autre.c, 25
 - autre.h, 28
- sommeForceCarte
 - gestionCarte.c, 59
 - gestionCarte.h, 67
- sommeTableau
 - tableau.c, 94
 - tableau.h, 100
- sous-menus.c, 85
 - leaderboard, 86
 - parametre, 86
 - statistiqueJoueur, 86
- sous-menus.h, 87
 - leaderboard, 87
 - parametre, 88
 - statistiqueJoueur, 88
- statistiqueJoueur
 - sous-menus.c, 86
 - sous-menus.h, 88
- stockeInfoCarte
 - formatageChaine.c, 34
 - formatageChaine.h, 40
- SUD
 - main.h, 84
- supprimeCarte
 - gestionCarte.c, 60
 - gestionCarte.h, 67
- SURCOINCHE
 - main.h, 83
- tableau.c, 89
 - acquiertTableau, 89
 - afficheTableau, 90
 - aleatoireTableau, 90
 - constanteTableau, 91
 - copie, 91
 - maxi, 91
 - mini, 92
 - moyenneTableau, 92
 - ordonnerTableau, 93
 - rechercheDichotomie, 93
 - sommeTableau, 94
 - zeroSiSuperieur, 94
- tableau.h, 95
 - acquiertTableau, 95
 - afficheTableau, 96
 - aleatoireTableau, 96
 - constanteTableau, 97
 - copie, 97
 - maxi, 97
 - mini, 98
 - moyenneTableau, 98
 - ordonnerTableau, 99
 - rechercheDichotomie, 99
 - sommeTableau, 100
 - zeroSiSuperieur, 100
- TAILLE_MAXI_COULEUR
 - main.h, 81
- TAILLE_MAXI_MESSAGE
 - main.h, 81
- TAILLE_MAXI_PSEUDO
 - main.h, 81
- TOUT_ATOUT
 - main.h, 83
- TREFLE
 - main.h, 83
- trieCarte

- gestionCarte.c, [60](#)
 - gestionCarte.h, [68](#)
- TypeMessage
 - main.h, [82](#), [84](#)
- vainqueurPli
 - gestionCarte.c, [61](#)
 - gestionCarte.h, [68](#)
- VALET
 - main.h, [85](#)
- Valeur
 - main.h, [83](#), [85](#)
- valeur
 - Carte, [5](#)
- ZERO
 - main.h, [84](#)
- zeroSiSuperieur
 - tableau.c, [94](#)
 - tableau.h, [100](#)