Société immobilière HsH

***

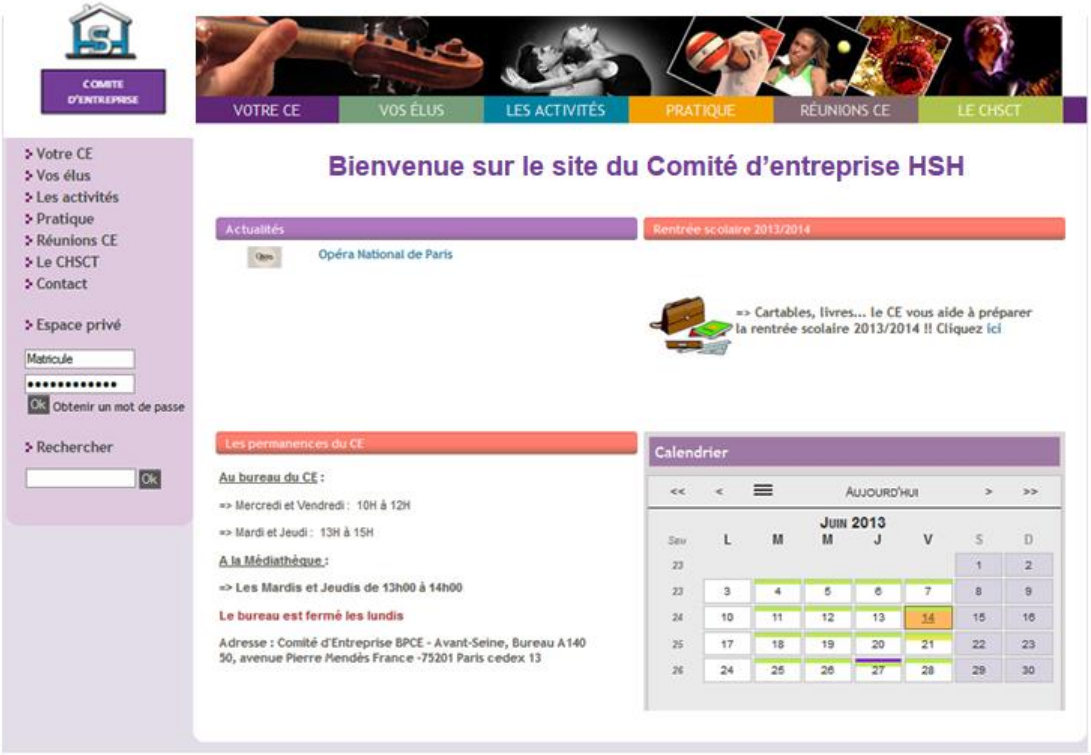
Maintenance correctrice et évolutive d'une application de jeu pour le Comité d'Entreprise

Présentation

La société HsH e(« Home Sweet Home ») est l'un des premiers réseaux immobilier français.

- Crée en 1984, HsH est aujourd'hui le troisième réseau d'agences immobilières en France.
- Plus de 1 000 points de ventes répartis sur tout le territoire, une implantation et des moyens qui favorisent les transactions grâce aux 5 500 collaborateurs.
- Le réseau HsH bénéficie d'un atout unique dans le secteur de l'immobilier : son fichier commun de plus de 100 000 biens.
- Un total de 5 500 collaborateurs et plus de 50 000 transactions par an.
- Plus de 60 GIE locaux (Groupement d'Intérêt Economique).
- Plus de 100 000 annonces sur www.HsH-immo.com.
- Un site central et des sites locaux.

Le Comité d'Entreprise



Dans les entreprises d'au moins 50 salariés, l'employeur est tenu d'organiser la mise en place d'un comité d'entreprise composé :

- de représentants élus du personnel ;
- de représentants syndicaux désignés par les organisations syndicales.

Ce comité assume d'une part, des attributions économiques et d'autre part, sociales et culturelles et dispose pour ce faire, des moyens matériels et financiers nécessaires. Le CE de HsH dispose de deux subventions distinctes :

- la subvention de fonctionnement, versée tous les ans et correspondant à un pourcentage de la masse salariale brute;
- la contribution aux activités sociales et culturelles.

L'employeur met à la disposition du CE un local aménagé et le matériel nécessaire à son fonctionnement (téléphone, mobilier, photocopie…).

Le CE assure, organise et développe, en faveur des salariés de l'entreprise, des ACTIVITES SOCIALES ET CULTURELLES. Il peut s'agir notamment :

- de la prise en charge de tout ou partie d'une mutuelle de santé,

- de la prise en charge de tout ou partie d'une cantine,
- d'activités sportives ou de loisirs (colonies de vacance, séjours...),
- d'activités culturelles (bibliothèques, tarifs préférentiels pour des spectacles ou des musées...),
- etc…

Dans ce contexte, le CE de HSH souhaite offrir un ESPACE DETENTE destiné au personnel de HsH et à leurs enfants, permettant :

- l'accès Internet aux personnes qui ne possèdent pas d'ordinateur portable Wifi ou de Smartphones pour surfer sur le net avec le WIFI gratuit ;
- l'utilisation de JEUX INFORMATIQUES sur des ordinateurs sous Windows. Certains de ces jeux ont été développés par la Direction des Systèmes d'Information de HsH, autour de la plateforme VISUAL STUDIO, Framework .NET et langage C#.

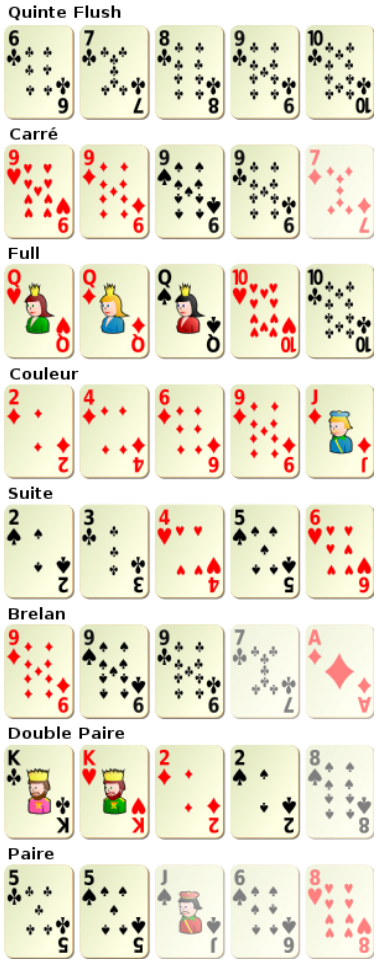
CAHIER DE CHARGES - Le besoin exprimé

Il est apparu que certains jeux ne marchent pas (ou mal) et présentent des fonctionnalités trop sommaires. La plupart doivent faire l'objet d'une maintenance corrective et/ou évolutive.

Les informaticiens de HsH sont actuellement occupés sur des projets internes ; c'est la raison pour laquelle, le CE de HsH fait appel à l'ESN « SIVY » afin d'assurer la maintenance corrective et/ou évolutive d'un ensemble de jeux, parmi lesquels « POKER ».

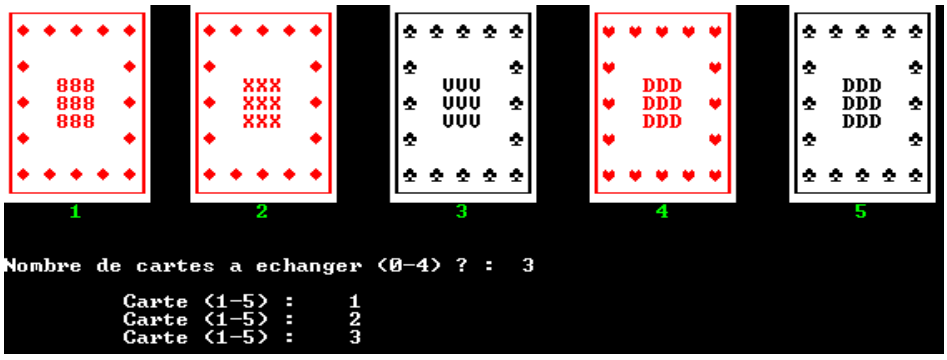
Le programme à développer s'inspire du jeu du Poker.

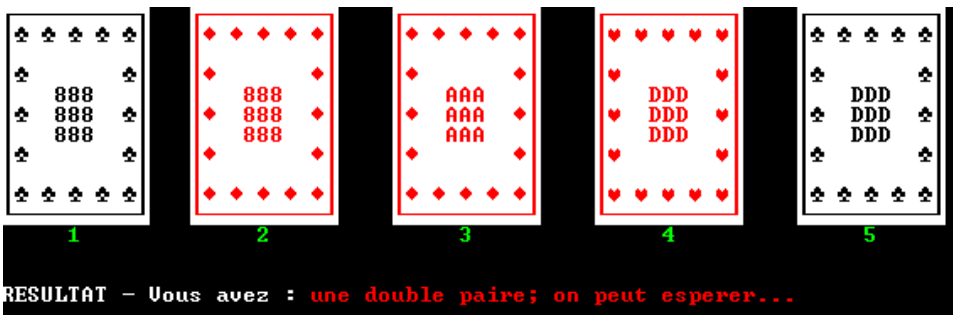
- Un tirage aléatoire de 5 cartes est effectué.
- Il faudra s'assurer, bien sûr, que chaque carte est unique, et ensuite, l'utilisateur aura la possibilité, soit de conserver son jeu, soit d'échanger au plus quatre cartes, et obtenir ainsi une nouvelle main.
  Le logiciel doit ensuite calculer et afficher le résultat parmi les combinaisons suivantes :



L'interface à respecter est la suivante :

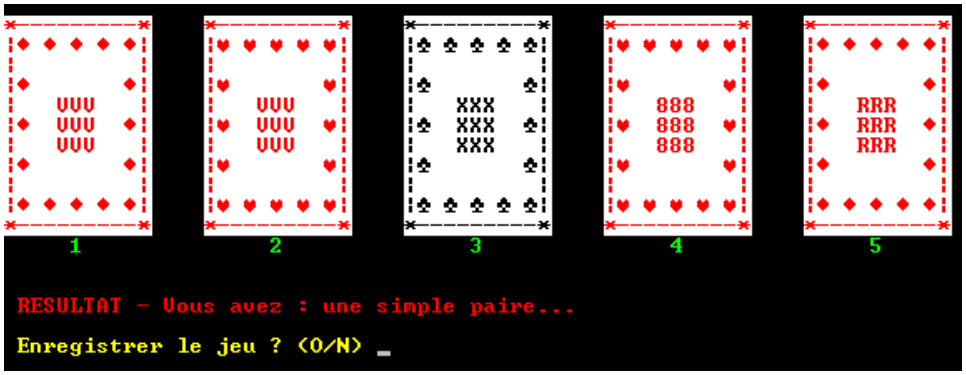
- Tirage aléatoire d'un jeu de 5 cartes.
- Affichage du jeu, carte par carte, en mode console.
- Demande, saisie et contrôle du nombre de cartes à échanger (4 maximum).
- Nouveau tirage pour chaque carte à échanger.
- Affichage du jeu et de la combinaison définitive.

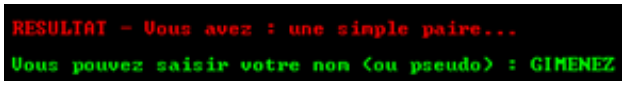




Enregistrement (sauvegarde) du SCORE obtenu

Après avoir affiché le résultat du jeu, il est souhaitable de proposer à l'utilisateur d'ENREGISTER LE JEU OBTENU DANS UN FICHIER TEXTE. Une fois le nom (ou pseudo) saisi, un article supplémentaire sera ajouté au FICHIER SCORES. Exemple d'interface :



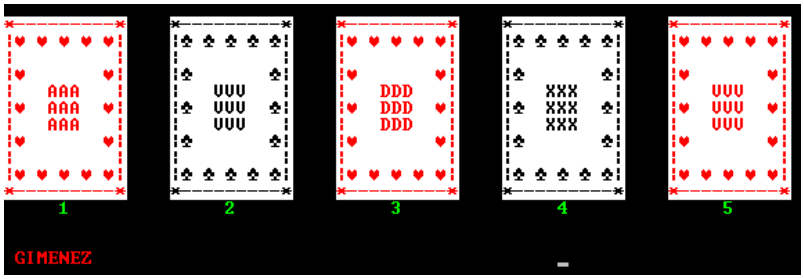


- Un jeu doit être enregistré dans le fichier sous la forme d'un nouvel ENREGISTREMENT (ou LIGNE).
- Par exemple, les valeurs du PSEUDO + le JEU [= (famille carte + valeur carte) * 5] vont formater un objet « string » qui sera ensuite écrit dans le fichier.
- Vous êtes cependant libres dans la manière dont ces informations seront enregistrées dans le fichier.

Consultation des SCORES enregistrés

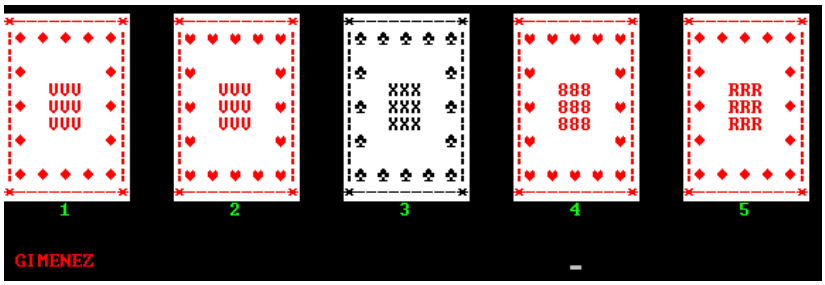
Une option dans le menu principal doit proposer à l'utilisateur de consulter l'intégralité des scores enregistrés dans le fichier.

- Le fichier devra être ouvert en mode LECTURE.
- Chaque ligne (ou enregistrement) sera récupéré.
- Les données vont ainsi se traduire par l'affichage du PSEUDO et du détail des 5 cartes composant le jeu enregistré.



1er jeu enregistré

...



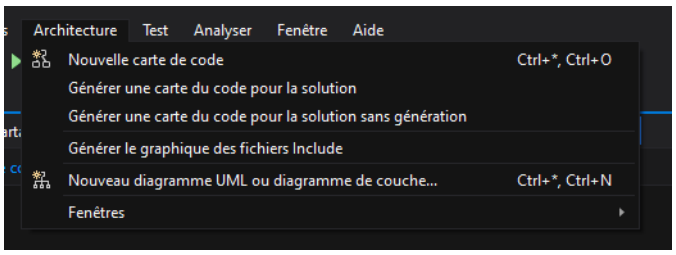
Nième jeu enregistré

ELEMENTS TECHNIQUES

Analyse de l'architecture avec des cartes de Code

Vous pouvez utiliser les outils de modélisation et de visualisation dans Visual Studio pour aider à comprendre le code existant et à décrire votre application. Cela vous permet de savoir visuellement comment vos changements peuvent affecter le code et vous aide à évaluer le travail et les risques qui résultent de ces modifications. Utilisez des cartes de code pour naviguer parmi les éléments de code et comprendre les relations entre eux. Ainsi, vous n'avez pas à effectuer le suivi de ce code dans votre tête, ni à dessiner un diagramme distinct.

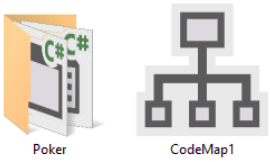
https://docs.microsoft.com/fr-fr/visualstudio/modeling/visualize-code?view=vs-2017



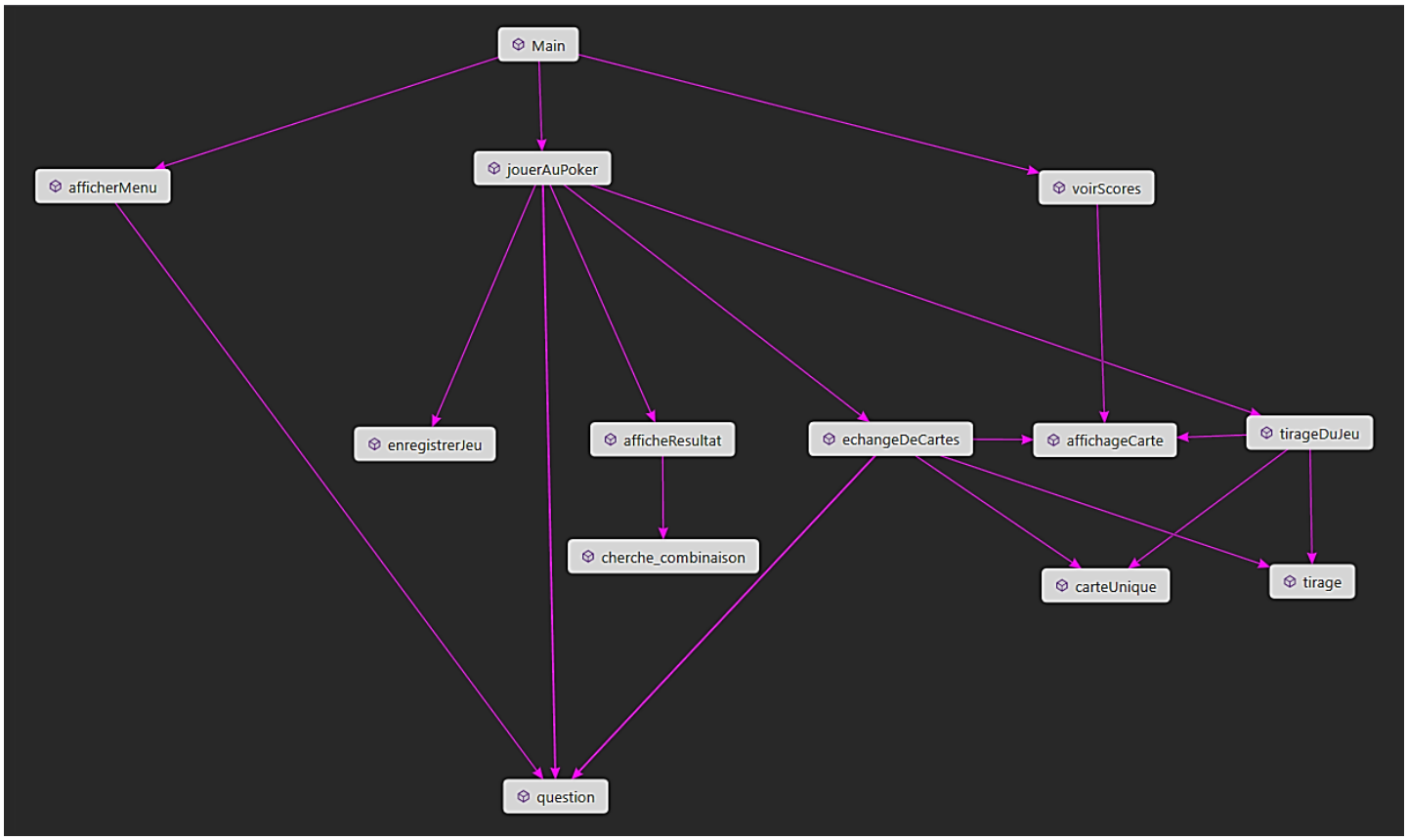
Le code ainsi généré peut être :

- Filtré (choix des éléments à afficher ou à masquer).
- Réorganisé afin de faciliter sa lecture.

Un fichier supplémentaire est ajouté à l'ensemble des fichiers constitutifs de la solution :



REMARQUE : la fonction « afficher_message() » n'apparaît pas sur la carte en raison du grand nombre de fonctions qui l'utilisent. Sa présence rends l'affichage du code trop complexe.



Architecture de la solution

1) Espaces de noms (dans des fichiers « dll ») importés

```
using System;
using System.IO;
using System.Runtime.InteropServices; // Pour _getch() et les couleurs
```

2) Organisation du code en REGIONS

```
0 références
class Program
{
    RESSOURCES EXTERNES

    TYPES DE DONNEES

    VARIABLES

    FONCTIONS PRIVEES

    FONCTIONS PUBLIQUES DU MENU PRINCIPAL

    MENU PRINCIPAL DU JEU
}
```

3) Ressources externes

Il s'agit pour l'essentiel de référencer des fichiers « dll » contenant du code qui n'est pas géré par la plateforme .NET (il s'agit, par exemple des fonctions écrites en langage C), dont le but est la gestion des couleurs et de la saisie en mode console.

```
 // Gestion des couleurs
[DllImport("kernel32.dll")]
public static extern bool SetConsoleTextAttribute(IntPtr hConsoleOutput, int wAttributes);
[DllImport("kernel32.dll")]
public static extern IntPtr GetStdHandle(uint nStdHandle);

static uint STD_OUTPUT_HANDLE = 0xfffffff5;
static IntPtr hConsole = GetStdHandle(STD_OUTPUT_HANDLE);

// Pour utiliser la fonction C 'getchar()' : sasie d'un caractère
[DllImport("msvcrt")]
static extern int _getch();
```

Pour gérer les couleurs :

- Les DLL correspondantes ont été incluses.
- Des constantes correspondantes aux codes couleurs ont été déclarées.

L'instruction suivante permet d'afficher les différentes combinaisons de couleurs :

```
for (int k = 1; k < 255; k++)
{
SetConsoleTextAttribute(hConsole, k);
Console.WriteLine("{0:d3} Hello",k);
}

SetConsoleTextAttribute(hConsole, 236);
```

4) Types de données & variables

```
// Codes COULEUR
public enum couleur
{
    VERT = 10, ROUGE = 12,
    JAUNE = 14, BLANC = 15, NOIRE = 0,
    ROUGESURBLANC = 252, NOIRESURBLANC = 240
};

// Coordonnées pour l'affichage
public struct coordonnees
{
    public int x;
    public int y;
}

// Une carte
public struct carte
{
    public char valeur;
    public int famille;
};

// Liste des combinaisons possibles
public enum combinaison
{
    RIEN, PAIRE, DOUBLE_PAIRE,
    BRELAN, QUINTE, FULL, COULEUR,
    CARRE, QUINTE_FLUSH
};
```

REMARQUE : un type énuméré (« enum ») est un type de données (généralement défini par l'utilisateur) qui consiste en un ensemble de CONSTANTES appelées énumérateurs.

Lorsque l'on crée un type énuméré on définit ainsi une « énumération ».

Lorsqu'une variable est déclarée comme étant de type énuméré, cette variable peut recevoir n'importe quel énumérateur comme valeur.

```
 // Coordonnées de départ pour l'affichage
public static coordonnees depart;

// Fin du jeu
public static bool fin = false;

// Valeurs des cartes : As, Roi,...
public static char [] valeurs = {'A','R','D','V','X','9','8','7'};

// Codes ASCII (3 : coeur, 4 : carreau, 5 : trèfle, 6 : pique)
public static int [] familles = {3,4,5,6};

// Numéros des cartes à échanger
public static int [] echange = {0,0,0,0};

// Jeu de 5 cartes
public static carte[] MonJeu = new carte[5];
```

5) Fonction PRINCIPALE (=menu du jeu)

```
// Fonction PRINCIPALE (=point d'entrée de la solution)
static void Main(string[] args)
{
        Console.Clear();

        // Point de départ du curseur
        // Top : Par rapport au sommet de la fenêtre
        // Left : Par rapport à la gauche de la fenêtre
        depart.x  = Console.CursorTop;
        depart.y = Console.CursorLeft;

        //---------------
        // BOUCLE DU JEU
        //---------------
        while(true)
        {
            // Affichage du menu et saisir du choix
            switch (afficherMenu())
            {
                // Jouer
                case '1' :
                    jouerAuPoker(MonJeu, echange);
                    break;

                // Meilleurs scores
                case '2' :
                    Console.Clear();
                    voirScores();
                    Console.ReadKey();
                    break;

                // Quitter
                case '3':
                    fin = true;
                    break;
                };

            if (fin)
                break;  // Casser la boucle !
        }

        Console.Clear();
        Console.ReadKey();
}
```

6) Fonctions PUBLIQUES (cf. menu principal)

Il s'agit des SEULES fonctions VISIBLES par la fonction principale (cf. mot clé « public ») :

- afficherMenu()
- jouerAuPoker()
- voirScores()

```
//-----------------------------------------------------
// Affiche le menu du jeu et retourne l'option choisie
//  1. AFFICHAGE DU MENU
//  2. LECTURE ET RETOUR DU CHOIX (=caractère)
// Fonctions PRIVEES appelées :
//      * afficher_message()
//      * question()
//-----------------------------------------------------
public static char afficherMenu()
{}

//-------------------------------------------------------------------
// Jouer au poker
//  1. TIRAGE D'UN JEU DE 5 CARTES (cf. tirageDuJeu())
//  2. ECHANGE DE CARTES (cf. echangeDeCartes())
//  3. CALCUL ET AFFICHAGE DU RESULTAT DU JEU (cf. afficheResultat())
//  4. DEMANDE DE NOUVEAU TIRAGE
//
// Fonctions PRIVEES appelées :
//      * tirageDuJeu()
//      * echangeDeCartes()
//      * afficheResultat()
//      * question()
//      * enregistrerJeu()
//-------------------------------------------------------------------
public static void jouerAuPoker(carte [] leJeu, int [] ech)
{}

// ---------------------------------------------------------------------------------
// Voir les scores stockés dans un FICHIER TEXTE
//  1. Ouverture en LECTURE du fichier "scores.txt"
//  2. ACCES à chaque enregistrement et AFFICHAGE sous la forme d'un JEU de CARTES
//
// Fonctions PRIVEES appelées :
//      * afficher_message()
//      * affichageCarte()
//---------------------------------------------------------------------------------
public static void voirScores()
{}
```

REMARQUE : les fonctions « afficherMenu() » et « jouerAuPoker() » ont déjà été écrites. Cette dernière, plus particulièrement, fait appel à un ensemble de fonctions « privées » qui devront pour la plupart être écrites :

```
// TIRAGE D'UN JEU DE 5 CARTES
tirageDuJeu(leJeu);

// ECHANGE DE CARTES
echangeDeCartes(leJeu, ech);

// CALCUL ET AFFICHAGE DU RESULTAT DU JEU
afficheResultat(leJeu);

// NOUVEAU TIRAGE ?

// ENREGISTRER LE JEU ?
```

Il restera ensuite à développer la dernière fonction « voirScores() », qui permet de consulter les scores enregistrés dans un FICHIER TEXTE.

```
// ---------------------------------------------------------------------------------
// Voir les scores stockés dans un FICHIER TEXTE
//  1. Ouverture en LECTURE du fichier "scores.txt"
//  2. ACCES à chaque enregistrement et AFFICHAGE sous la forme d'un JEU de CARTES
//
// Fonctions PRIVEES appelées :
//      * afficher_message()
//      * affichageCarte()
//---------------------------------------------------------------------------------
public static void voirScores()
{
    // A ECRIRE
}
```

7) Fonctions PRIVEES

- **Fonctions déjà écrites** (pour l'essentiel, il s'agit de la gestion de l'interface console).

```
// Affiche un message de couleur spécifiée après avoir placé le curseur
// PARAMÈTRES :
//    * Le texte à afficher
//    * Sa COULEUR
//    * Les COORDONNEES d'affichage
private static void afficher_message(string message, couleur lacouleur, coordonnees c)
{
    // Se placer
    Console.SetCursorPosition(depart.x + c.x, depart.y + c.y);

    // Définir la couleur
    SetConsoleTextAttribute(hConsole, (int)lacouleur);

    // Afficher le message
    Console.Write(message);

    // Remise de la couleur par défaut
    SetConsoleTextAttribute(hConsole, (int)couleur.BLANC);
}
```

```
// Pose une question et retourne la réponse sous la forme d'un caractère
// PARAMÈTRES :
//    * Le message (éventuel) à afficher
//    * Les coordonnées d'affichage
//    * La valeur à ajouter à la ligne (X) pour attendre la saisie de la réponse
// FONCTIONS APPELÉES :
//    * afficher_message()
private static char question(string texte, coordonnees c, int deplacement_sur_x)
{
    afficher_message(texte, couleur.JAUNE, c);
    afficher_message("", couleur.BLANC, new coordonnees { x = c.x + deplacement_sur_x, y = c.y });
    char reponse = (char)_getch();
    afficher_message(new string(new char[] { reponse }),
        couleur.BLANC, new coordonnees { x = c.x + deplacement_sur_x, y = c.y });
    return reponse;
}
```

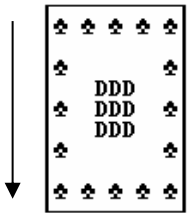
- **Fonctions à écrire et/ou à compléter**

```
// Génère aléatoirement une carte
// RETOURNE une expression de Type "CARTE" : STRUCTURE {valeur;famille}
private static carte tirage()
{}

// Indique si une carte est déjà présente dans le jeu
// PARAMÈTRES :
//     * Une CARTE
//     * Un TABLEAU de CARTES (=jeu 5 cartes)
//     * Le numéro de la carte dans le jeu (1...5)
// RETOURNE un booléen (carte présente ou non)
private static bool carteUnique(carte uneCarte, carte[] unJeu, int numero)
{}

// Affiche à l'écran une carte {valeur;famille} en fournisant la colonne de départ
// PARAMÈTRES :
//     * Une variable de type CARTE
//     * Un ENTIER correspondant à la colonne de départ pour l'affichage
// FONCTIONS APPELÉES :
//    * afficher_message()
private static void affichageCarte(carte uneCarte, int colonne)
{}
```

Dans cette fonction, l'affichage de la carte se fait depuis le haut vers le bas:



…

```
// Tirage d'un jeu de 5 cartes
// PARAMÈTRE : un tableau de 5 CARTES à remplir
// FONCTIONS APPELÉES :
//    * carteUnique()
//    * affichageCarte()
private static void tirageDuJeu(carte[] unJeu)
{}

// Echange des cartes
// PARAMÈTRES :
//     * Le tableau de 5 CARTES
//     * Le tableau des numéros des cartes à échanger
// FONCTIONS APPELÉES :
//    * carteUnique()
//    * affichageCarte() pour afficher chaque carte du jeu
private static void echangeDeCartes(carte[] unJeu, int[] e)
{}

// Calcule et retourne la combinaison
// (paire, double-paire... , quinte-flush) pour un jeu complet de 5 cartes
// PARAMÈTRE :
//      * Une énumération 'combinaison'
private static combinaison cherche_combinaison(carte[] unJeu)
{}
```

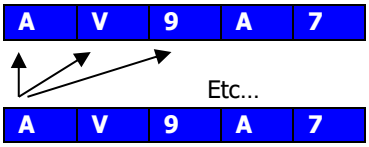
Principe de calcul du résultat

Il s'agit de calculer la COMBINAISON (paire, double paire..., quinte flush) pour un jeu complet de 5 cartes. La valeur retournée est un élément de l'énumération 'combinaison' (=constante). Deux tableaux peuvent être utilisés :

```
// Nombre de valeurs similaires dans le jeu pour chaque carte
int [] similaire = {0,0,0,0,0};

// Possibilités de quinte. Tableau 4*5
char [,] quintes = {    {'X','V','D','R','A'},
                        {'9','X','V','D','R'},
                        {'8','9','X','V','D'},
                        {'7','8','9','X','V'}
                   };
```

Soit le jeu suivant :



On compte, pour chaque carte, le nombre de fois où sa valeur apparaît dans le jeu, ce qui permet de remplir le TABLEAU « similaire ».

Les combinaisons sont les suivantes :

PAIRE : S'il existe un élément du tableau « similaire » égal à 2, on a une paire.

DOUBLE PAIRE : Chaque fois que l'on a une paire on incrémente un compteur.
Si ce compteur / 2 = 2 alors on a une double paire…

CARRE : Il existe un élément du tableau « similaire » égal à 4.

BRELAN : Il existe un élément du tableau « similaire » égal à 3.

QUINTE : Tous les éléments du tableau « similaire » doivent être 1.
Ensuite, il faut comparer chaque carte avec chaque possibilité de quinte (chaque élément du tableau « quintes ». Si égalité, on incrémente un compteur. Lorsque ce compteur est égal à 5, alors on a une quinte…

QUINTE FLUSH : Il faut avoir une quinte et toutes les cartes de la MEME FAMILLE…

FULL : Il faut avoir à la fois un BRELAN et une PAIRE.

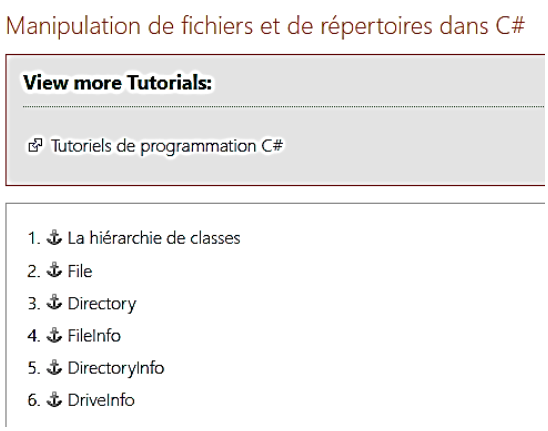
COULEUR : Les 5 cartes doivent être de la MEME FAILLE, mais sans constituer une QUNTE…

```
// Calcul et affichage du résultat
// PARAMÈTRE : le tableau de 5 cartes
// FONCTIONS APPELÉES :
//    * afficher_message()
private static void afficheResultat(carte [] unJeu)
{}

// Enregistrer le jeu dans un fichier
// PARAMÈTRE : le tableau de 5 cartes
// FONCTIONS APPELÉES :
//    * afficher_message()
//    * cherche_combinaison()
private static void enregistrerJeu(carte [] unJeu)
{}
```

Gestion de fichiers en C# (RESSOURCES)

https://devstory.net/10453/manipulation-de-fichiers-et-de-repertoires-dans-csharp



https://docs.microsoft.com/fr-fr/troubleshoot/dotnet/csharp/read-write-text-file

Utiliser Visual C# pour lire et écrire dans un fichier texte

Une alternative aux fichiers TEXTE est les fichiers dits « binaires ». On peut alors lire et écrire octet par octet. Deux classes sont nécessaires :

https://docs.microsoft.com/fr-fr/dotnet/api/system.io.binaryreader?view=net-6.0
https://docs.microsoft.com/fr-fr/dotnet/api/system.io.binarywriter?view=net-6.0

BinaryReader Classe

BinaryWriter Classe