

Projet

Réalisation d'un démineur simplifié

```
*****
* Démineur simplifié (Dimitrios Lymberis) *
*****
A vous de jouer !! Mode : Difficile
6 mines se cachent dans le jeu !

  1  2  3  4  5  6
1  1  1  1  1  1
2  1  1  1  1  1
3  1  1  1  1  1
4  1  1  1  1  1
5  1  1  1  1  1
6  1  1  1  1  1

Consignes
-----
- Pour se déplacer dans le jeu utiliser les touches fléchées
- Pour explorer une case la touche Enter
- Pour définir une case en tant que mine (flag) la touche Espace
- La touche Enter sur un flag enlève le flag
- Pour quitter la touche Esc

La partie est gagnée :
- une fois que toutes les cases ont été explorées
- que toutes les mines n'ont pas été explosées

il reste encore 5 mine(s) cachée(s)
```

Sujet :

Le développement de jeux informatisés requiert souvent une conception robuste doublée de la mise sur pied de structures de données adaptées, qui permettent d'atteindre la rapidité voulue tout en offrant quantité d'options.

Ce projet consistera en la réalisation d'un programme en mode console du jeu démineur dans une version simplifiée.

La conception et l'implémentation de ce programme demandent l'utilisation de plusieurs concepts reliés au cours du module 319. Parmi ceux-ci on note la gestion dynamique de la mémoire avec tableau dynamique en 2D, l'utilisation de fonction avec passage de paramètres par valeur ou par référence et finalement l'utilisation selon votre algorithme d'une fonction récursive.

Thèmes abordés :

- Les itérations, les conditions, les opérateurs
- Les conversions
- Les couleurs
- Variables et constantes
- Les méthodes
- Les tableaux à 2 dimensions
- Objet Random (aléatoire)
- Le son Beep
- Les valeurs ASCII
- Les Enumérations

Durée : 22p

Contraintes :

Il vous est demandé de développer le projet avec une attention particulière sur la méthode de programmation, qui devra être rigoureuse et tenir compte des concepts fondamentaux de qualité des logiciels.

Vous devez utiliser Visual studio et le langage c#. La programmation se fera en mode console. Vous êtes libre d'utiliser .Net Framework ou .Net Core.

Le programme doit s'appeler **demineur_votreNom** ex : demineur_lymberis

A rendre :

- Le code source complet de votre code

L'objectif du jeu :

Le jeu démineur est un jeu de chance et d'analyse logique qui consiste à localiser toutes les mines présentes sur la grille de jeu sans en mettre aucune à nu.

La grille de jeu et comment jouer

Le jeu est constitué d'une grille rectangulaire ou carrée où sont cachées les mines.

Les mines sont placées aléatoirement dans la grille selon une formule donnée plus loin dans ce document. Au fur et à mesure que le jeu avance, certaines cases de la grille sont explorées et des informations apparaissent.

Si le joueur explore une case contenant une mine, il perd une vie et un Beep caractéristique retentit.

Pour gagner

Pour gagner la partie, le joueur doit identifier toutes les cases qui ne sont pas minées par un flag. Il peut également gagner si toutes les mines ne sont pas explosées et que les mines restantes sont identifiées par un flag.

```
*****
*          Démineur simplifié (Dimitrios Lymberis)          *
*****

A vous de jouer !! Mode : Moyen
4 mines se cachent dans le jeu !

  1  2  3  4  5  6
  +--+--+--+--+
  |  |  |  |  |  |
  |  |  |  |  |  |
  |  |  |  |  |  |
  |  |  |  |  |  |
  |  |  |  |  |  |
  |  |  |  |  |  |
  +--+--+--+--+

Consignes
-----
- Pour se déplacer dans le jeu utiliser les touches fléchées
- Pour explorer une case la touche Enter
- Pour définir une case en tant que mine (flag) la touche Espace
- La touche Enter sur un flag enlève le flag
- Pour quitter la touche Esc

La partie est gagnée :
- une fois que toutes les cases ont été explorées
- que toutes les mines n'ont pas été explosées

il reste encore 2 mine(s) cachée(s)
C'est la fin !

!! BRAVO !! Vous avez réussi à pas marcher sur toutes les mines !
Il restait 2 sur 4 mines

Voulez-vous recommencer ?
(o) pour oui autre touche pour quitter ! :
```

Une partie est terminée, lorsque :

- Il n'y a plus de cases à jouer
- Un joueur à gagner.
- Le joueur désire quitter le jeu

Il est donné aux joueurs la possibilité de recommencer !

- Au démarrage le curseur se place sur la 1^{ère} case
- Le déplacement se fait à l'aide des touches fléchées et la touche Enter permet de jouer la case
- La touche Espace permet de placer un Flag, un Enter sur la touche Flag l'enlève !
- La touche Esc permet de quitter le jeu.

Consignes

Au démarrage du jeu, un **titre** apparaît suivi d'une demande la taille du plateau de jeu en demandant le **nombre de ligne et colonne** ainsi que le niveau de **difficulté**.

```
*****
*          Démineur simplifié (Dimitrios Lymberis)          *
*****

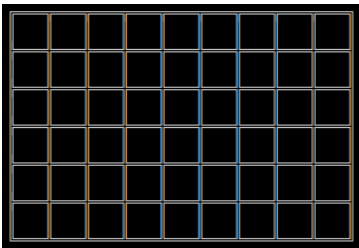
Merci d'entrer le nombre de ligne et de colonne de votre plateau de jeux
en sachant qu'au minimum on a un plateau de 6 lignes x 6 colonnes !
et au maximum un plateau de 30 lignes x 30 colonnes !
-----

Nombre de ligne : 6
Nombre de colonne : 6

Merci d'entrer la difficulté pour votre jeu
en sachant que :
    1 --> niveau facile
    2 --> niveau moyen
    3 --> niveau difficile
-----

Votre difficulté : 2
```

La taille de la grille doit être au minimum de **6 par 6** et au maximum de **30 par 30**. Elle peut donc valoir par exemple : 6x9.



Trois niveaux de difficulté sont proposés correspondant au nombre de mines positionnées aléatoirement dans la grille. Ce nombre est défini en pourcentage par rapport à une formule définissant la surface de la grille de jeu.

Formule : Surface = (nombre de ligne / 2) + 1 * (nombre de colonne / 2) + 1

La difficulté :

- 1) Facile 10% de la surface
- 2) Moyen 25% de la surface
- 3) Difficile 40% de la surface

Exemple pour 8 x 12 diff. 2) → on a 8 mines

Il vous faut gérer les erreurs d'entrée ! seul bien sûr des nombres sont autorisés.
Pour cela, penser à la méthode **TryParse** des types de données.

Ex : `valueOk = int.TryParse(Console.ReadLine(), out nRow);`

Des messages apparaissent dans le cas d'erreurs d'entrées : exemples

```
Nombre de ligne : g
Votre valeur n'est pas un nombre ! Merci de réessayer !
Nombre de ligne : 3
Valeur hors limite ! Merci de réessayer !
Nombre de ligne :
```


A droite du plateau de jeu vous y mettrez des consignes pour l'usage du jeu. Ceux-ci doivent être placés correctement selon la grandeur du plateau de jeu.

```
Consignes
-----
- Pour se déplacer dans le jeu utiliser les touches fléchées
- Pour explorer une case la touche Enter
- Pour définir une case en tant que mine (flag) la touche Espace
- La touche Enter sur un flag enlève le flag
- Pour quitter la touche Esc

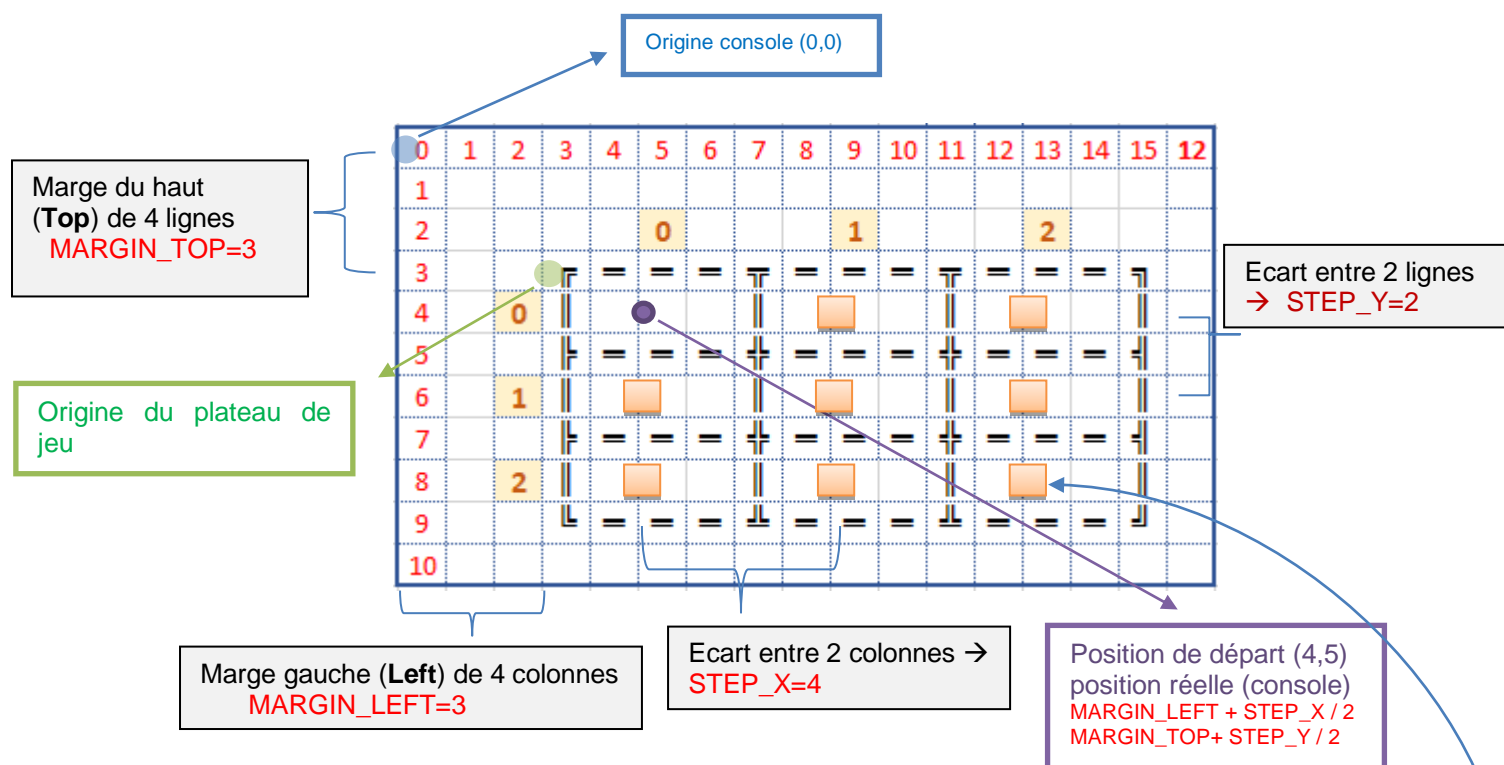
La partie est gagnée :
- une fois que toutes les cases ont été explorées
- que toutes les mines n'ont pas été explosées
```

Au démarrage le curseur se placera au milieu de la 1ère cellule.

Pour vous aider à construire le plateau de jeu, voici une représentation du plateau de jeux avec ces caractéristiques (marges, écarts, etc..)

Exemple

Grille de 3 lignes x 3 colonnes:



Vous l'avez compris, votre code devra comporter des constantes et des boucles d'itérations !


Rappel :

Code ascii nécessaire pour les bordure avec la combinaison des touches : ALT+ code exemple:

ALT+201	┌	ALT+205	=	ALT+187	┐	ALT+186		ALT+185	└
---------	---	---------	---	---------	---	---------	--	---------	---

A vous de décider les marges que vous voulez (position de votre plateau de jeu)

Remarque

En observant le plateau de jeu, on s'aperçoit que les coordonnées réelles du positionnement du curseur de jeux  se calcul grâce à la position selon la ligne et la colonne de notre plateau de jeu.

Position Left = $MARGIN_LEFT + STEP_X / 2 + pos_C * STEP_X$

Position Top = $MARGIN_TOP + STEP_Y / 2 + pos_L * STEP_Y$

Exemple pour (2,2) ligne 2 et colonne 2

Left=3+2+2*4=13

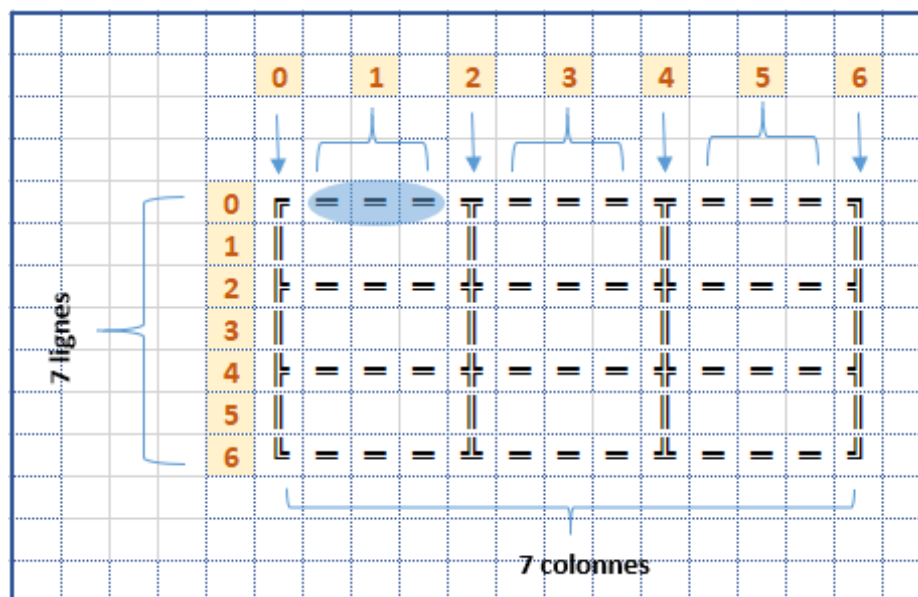
Top=3+1+2*2=8

Comment construire le plateau de jeu ?

Comme nous devons entraîner les boucles d'itération et les instructions conditionnelles, je vous propose une solution parmi tant d'autres.

Lorsque vous êtes confronté au choix d'une solution, prenez le temps de la réflexion pour analyser votre algorithme.

Observons le plateau de jeux !!

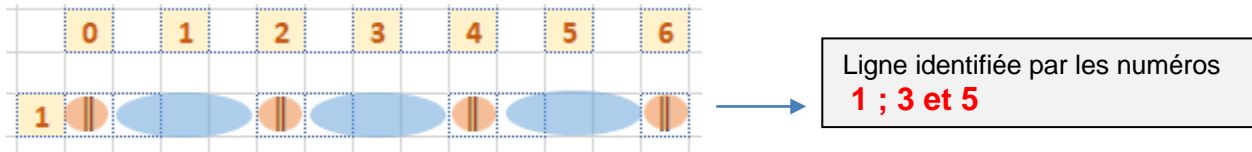


On remarque des répétitions au niveau des lignes et des colonnes !

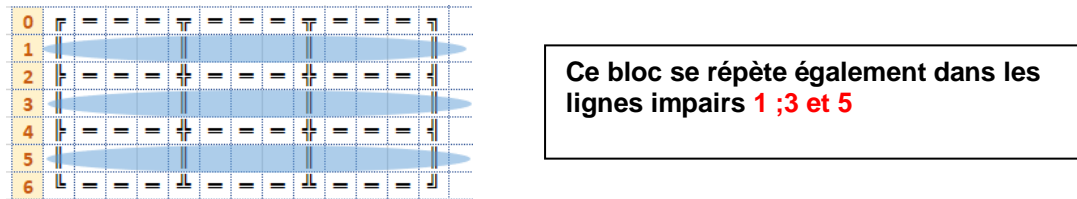
Au niveau des lignes :



A part les colonne 0 et 6 qui représente les coins du plateau, on observe que les zones bleues identifiées par les numéros **impairs 1 ; 3 et 5** se répète et les zones orangées **représentées par les numéros pairs 2 et 4** se répète également.



Ici on observe que les zones bleues se répètent selon les numéros **impairs 1 ; 3 et 5** et les zones oranges par les numéros **pairs 0 ; 2 ; 4 et 6**



Ici également comme pour la 1ère ligne, à part les colonnes 0 et 6 il y a 2 zones qui se répètent **impairs** → bleues et **pairs** → orange.

Cette ligne se répète également en **2 et 4** → **pairs**



Il reste la dernière ligne le principe !!

mais là vous avez compris

Au niveau des colonnes on a le même principe

Se déplacer dans le plateau de jeu

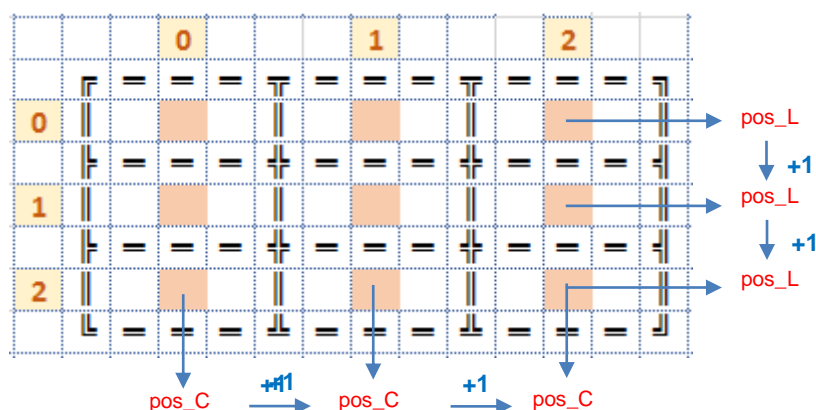
Il vous faut mettre en place un code vous permettant de vous déplacer dans les cellules grâce aux touches de



direction

Pour se faire il vous faut utiliser la méthode **ReadKey** de la console et une **boucle tant que** !

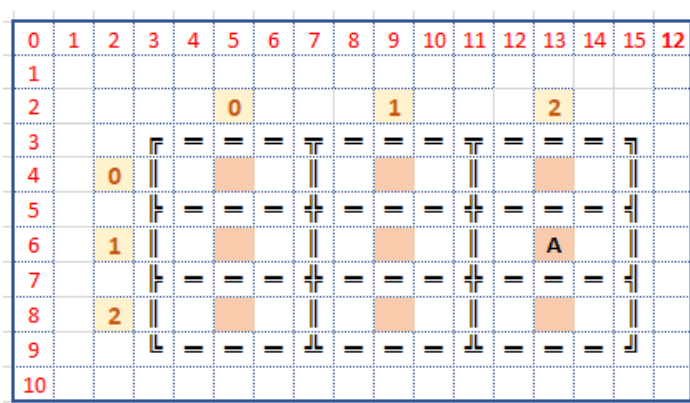
Vous pouvez également utiliser la position en **ligne** et **colonne** du tableau de jeu. En effet, on remarque sur le plateau de jeux.



On remarque bien une similitude avec un tableau à 2 dimensions de 3 x 3

Ce qu'on recherche ici c'est de se déplacer selon les **lignes** et **colonnes** en jouant avec les indices d'un tableau.

Ex : si **pos_C=2** et **pos_L=1** je me trouverais donc au **point A**



Les coordonnées selon la référence de la console seront :

Position Left :

$\text{MARGIN_LEFT} + \text{STEP_X} / 2 + \text{pos_C} * \text{STEP_X}$

Position Top :

$\text{MARGIN_TOP} + \text{STEP_Y} / 2 + \text{pos_L} * \text{STEP_Y}$

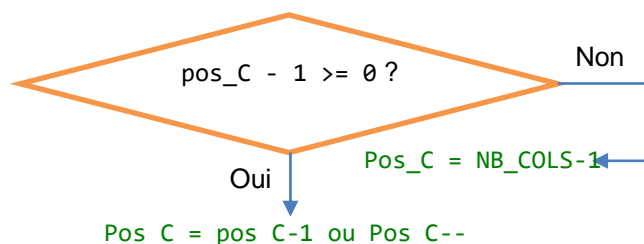
Non seulement on peut se positionner facilement sur le plateau de jeu mais on peut également facilement se déplacer dans un tableau à 2 dimensions grâce à **pos_C** et **pos_L**.

Il est également facile de gérer les bords de mon jeu en jouant également avec les valeurs du nbre de lignes et colonnes de mon plateau qui correspondront à **pos_C** et **pos_L**.

Exemple pour la touche



$\text{pos_C} = (\text{pos_C} - 1 \geq 0) ? \text{pos_C} - 1 : \text{NB_COLS} - 1;$



Rappel

Comment savoir la touche qui a été sélectionnée ? grâce à la propriété. Key de la commande `Console.ReadKey()`.

`ConsoleKey ConsoleKeyInfo.Key { get; }`

Obtient la touche de console représentée par le `ConsoleKeyInfo` objet.

Cette propriété retourne une valeur de type `ConsoleKey`. Il s'agit donc d'une énumération permettant de connaître à travers un nom la touche sélectionnée.

Exemple :

`Console.ReadKey().Key` peut avoir la valeur : `ConsoleKey.UpArrow`

`Console.ReadKey().KeyChar` Objet qui correspond à la touche de console

Par exemple si j'appuie sur **Z** `Console.ReadKey().KeyChar` retourne **'Z'**

Je peux donc écrire P.ex `char usrChoice = Console.ReadKey(true).KeyChar;`

Le paramètre `true` permet de rendre invisible la touche pressée et `usrChoice` vaudra **'Z'**

Le tableau à 2 dimensions

Suite en construction !!

A vous de compléter et d'ajouter ce qu'il faut pour réaliser votre jeu.

Je rappelle qu'il vous faut utiliser à bon escient les méthodes, les variables et les constantes. Que votre code doit être propre et facilement lisible et compréhensible. Qu'il soit bien commenté et dans les normes de l'ETML.

Quelques rappels théoriques

Les méthodes

Il y a deux grandes familles de méthode celle qui ne **retourne rien** et celles qui **retourne une information** !

Prenons la 1^{ère} méthode sans retour.

Méthode qui ne retourne rien. On la reconnaît grâce au mot clé **void**.

Exemple

Si je veux afficher un titre à travers une méthode.

```
/// <summary>
/// Affiche un titre
/// </summary>
static void DisplayTile()
{
    Console.WriteLine("Jeu du démineur");
}
```

Vous remarquez le respect des normes

- Commentaire d'entête de méthode
- Nom de la méthode avec première lettre en majuscule

Cette méthode ne retourne **rien** d'où le type **void** !

Pour appeler la méthode, il suffit de la nommer dans votre code : `DisplayTile()`;

Ajoutons-lui des paramètres !

Par exemple, je veux afficher un titre à une position précise sur la console.

Il suffit pour cela de mettre 2 variables entre les parenthèses de la déclaration de ma méthode
→ c'est ce **qu'on appelle des paramètres**.

```
/// <summary>
/// Affiche un titre à une position donnée
/// </summary>
/// <param name="posLeft">position gauche du titre</param>
/// <param name="posTop">position depuis le haut du titre</param>
static void DisplayTile(int posLeft, int posTop )
{
    Console.SetCursorPosition(posLeft, posTop);
    Console.WriteLine("Jeu du démineur");
}
```

Vous remarquez qu'on a ajouté des commentaires pour les paramètres ! Cela facilite la compréhension. En effet, lors de l'appel de la méthode Visual studio vous donne des indications quant à l'utilisation de la méthode.

Si vous mettez le curseur au début de la parenthèse VS vous aide en affichant le commentaire de votre paramètre.



```
DisplayTile();
void Program.DisplayTile(int posLeft, int posTop)
    Affiche un titre à une position donnée
    posLeft: position gauche du titre
```

Vous l'avez compris pour appeler la méthode, il suffit d'écrire la méthode et de lui fournir des valeurs pour les paramètres.

`DisplayTile(5,8);`

→ placera le titre à 5 caractères depuis la gauche de la console et 8 depuis le haut de la console.

Modifions cette méthode en lui indiquant que les paramètres sont optionnels ! ce qui veut dire que lors de l'appel de la méthode nous ne sommes pas obligés de fournir des valeurs aux paramètres ! parce que ceux-ci auront en fait une valeur par défaut.

Comment fait-on ? simplement en **initialisant les variables** qu'on a introduit comme paramètres.

```

/// <summary>
/// Affiche un titre à une position donnée
/// </summary>
/// <param name="posLeft">position gauche du titre</param>
/// <param name="posTop">position depuis le haut du titre</param>
static void DisplayTile(int posLeft=5, int posTop=8 )
{
    Console.SetCursorPosition(posLeft, posTop);
    Console.WriteLine("Jeu du démineur");
}

```

Lors de l'appel vous savez qu'il n'est pas indispensable de renseigner le paramètre, car celui-ci **est entre []**

```

DisplayTile()
void Program.DisplayTile([int posLeft = 5], [int posTop = 8])
Affiche un titre à une position donnée
posLeft: position gauche du titre

```

Donc si vous appelez la méthode DisplayTile();
vos paramètres voudront les valeurs que vous leur avez donné à l'initialisation soit 5 pour posLeft et 8 pour posTop

Une autre caractéristique très répandue pour les méthodes et le **passage des paramètres par référence**.

Le passage par référence s'obtient avec le mot clé **ref**.

Prenons un exemple

```

static void Main(string[] args)
{
    int b = 4;
    TestRef(ref b);
    Console.WriteLine("b= "+b);
}

static void TestRef(ref int value)
{
    value = 6;
}

```

Au démarrage de l'application b vaut 4
On appelle la méthode TestRef en donnant comme valeur à son paramètre **value** la valeur de la variable b, mais du fait de mettre le mot clé **ref** → on passe cette valeur en référence ! C'est comme si je passais **b**.

Du coup dans la méthode TestRef si je modifie **value** c'est comme si je modifiais **b**. C'est pour cela qu'à la sortie de la méthode ma variable b a été modifiée !

b= 6

Voyons la 2ème méthode avec retour.

Il suffit d'indiquer le type de retour en remplaçant le mot clé **void** par le type de donnée qu'on veut retourner et de ne pas oublier **le Return**

```

static int GetNbColonne()
{
    int nbCol = 0;

    Console.Write("Nombre de colonne : ");
    int.TryParse(Console.ReadLine(), out nbCol);

    return nbCol;
}

```

Ici GetNbColonne retournera la valeur entrée par l'utilisateur !

Une méthode avec retour s'assimile à une fonction, comme elle nous retourne quelque chose ce quelque chose je peux l'affecter directement à une variable.

Par Exemple → `int col = GetNbColonne();`

Ou l'utiliser directement → `Console.WriteLine(GetNbColonne());`

On peut également utiliser des paramètres avec ou sans ref. Son emploi est la même chose qu'avec des méthodes sans retour.

Les énumérations

Une **énumération** est utilisée pour définir un **ensemble de valeurs constantes**.

Il s'agit d'un **type** ! qui peut s'utiliser comme les autres types (string, int, etc..).

On peut donc déclarer une variable de type énumération.

Ces différentes valeurs représentent différents cas ; on les nomme **énumérateurs**.

Lorsqu'une variable est de type énuméré, elle peut avoir comme valeur **n'importe quel cas de ce type énuméré**. Pour définir un type d'énumération, utilisez le **enum** mot clé et spécifiez les noms **des membres enum**:

Exemple :

```
enum DifficultGame
{
    gameEasy = 1,
    gameMedium = 2,
    gameHard = 3
}
```

Cette déclaration **se fera avant le main** dans votre application console.

```
enum Doigt
{
    Pouce,
    Index,
    Majeur,
    Annulaire,
    Auriculaire
}
```

Par défaut, le type sous-jacent de chaque élément dans l'énumération est **int**
Donc un numérique



Avantages qu'offre l'utilisation d'un **enum** par rapport à un type numérique :

- Vous spécifiez clairement pour le code client les valeurs qui sont valides pour la variable.
- Dans Visual Studio, IntelliSense répertorie les valeurs définies.



Quand vous ne spécifiez pas de valeurs pour les éléments dans la liste d'énumérateurs, les valeurs sont **automatiquement** incrémentées de 1

```
enum MachineState
{
    PowerOff = 0,
    Running = 5,
    Sleeping = 10,
    Hibernating = Sleeping + 5
}
```

Vous pouvez affecter n'importe quelle valeur aux éléments dans la liste d'énumérateurs d'un type énumération, et vous pouvez également utiliser des valeurs calculées

Pour utiliser une **enum**, il suffit de déclarer une variable de ce type :

Je vous invite à relire le support de cours théorique sur les **enum**.

`//difficulté entré par l'utilisateur`

`DifficultGame gameDifficult=DifficultGame.gameEasy;`

Ça ne vous rappelle rien ?? → C'est la même chose avec les couleurs !

`Console.ForegroundColor= ConsoleColor.Green;`



`enum System.ConsoleColor`

Specifies constants that define foreground and background colors for the console.

ConsoleColor est un **enum** !! lorsque je click sur ConsoleColor j'obtiens :

```

... public enum ConsoleColor
{
    ... Black = 0,
    ... DarkBlue = 1,
    ... DarkGreen = 2,
    ... DarkCyan = 3,
    ... DarkRed = 4,
    ... DarkMagenta = 5,
    ... DarkYellow = 6,
    ... Gray = 7,
    ... DarkGray = 8,
    ... Blue = 9,
    ... Green = 10,
    ... Cyan = 11,
    ... Red = 12,
    ... Magenta = 13,
    ... Yellow = 14,
    ... White = 15
}

```

Une belle énumération !

Le ReadKey

La méthode **Console.ReadKey()** fait attendre au programme l'appui sur une touche.

La touche enfoncée peut **soit être affichée ou non** dans la fenêtre de la console.

Il existe deux méthodes dans la liste de **surcharge** de cette méthode :

▲ 1 sur 2 ▼ **ConsoleKeyInfo Console.ReadKey()**
Obtains the next character or function key pressed by the user. The pressed key is displayed in the console window.

▲ 2 sur 2 ▼ **ConsoleKeyInfo Console.ReadKey(bool intercept)**
Obtains the next character or function key pressed by the user. The pressed key is optionally displayed in the console window.
intercept: Determines whether to display the pressed key in the console window. true to not display the pressed key; otherwise, false.

Avec la 2^{ème} méthode en donnant comme valeur true au paramètre intercept, le caractère ne sera pas affiché.

Valeur de retour : cette méthode renvoie un objet qui décrit la constante **ConsoleKey** et le caractère Unicode (le cas échéant), il correspond à la touche enfoncée.

Il s'agit d'un type structure **ConsoleKeyInfo**

```

ConsoleKeyInfo usrEntry;

usrEntry = readonly struct System.ConsoleKeyInfo
    Describes the console key that was pressed, including the character represented by the console key and the state of the SHIFT, ALT, and CTRL modifier keys.

```

Exemple :

```

/* on définit une variable de type ConsoleKeyInfo
 * qui va récupérer le caractère appuyé par l'utilisateur */

ConsoleKeyInfo usrEntry;

// on attend que l'utilisateur appuie sur une touche
usrEntry = Console.ReadKey();

/* on affiche les caractéristiques de la touche appuyée
 * KeyChar retourne le code unicode du caractère appuyé il s'agit d'un char
 * Key lui représente la constante (un enum) qui décrit la touche appuyée
 * pour récupérer le code ascii de la touche il suffit de faire une conversion un CAST */
Console.WriteLine("\nKeyChar : " + usrEntry.KeyChar + " Key : " + usrEntry.Key);
Console.WriteLine("ASCII : " + (int)usrEntry.KeyChar );

```

// on peut donc effectuer toutes sortes de tests

```
if (usrEntry.KeyChar == 102)
    Console.WriteLine("Bravo");

if (usrEntry.KeyChar == 'f')
    Console.WriteLine("Re Bravo");

if (usrEntry.Key == ConsoleKey.F)
    Console.WriteLine("Re Re Bravo");
```

```
f
KeyChar : f Key : F
ASCII : 102
Bravo
Re Bravo
Re Re Bravo
```

Conclusion si je veux tester les touches fléchées



/.Key

Evaluation :

L'évaluation de ce projet se fera par la remise d'un document qui suit le temps alloué à ce projet.

Le document doit contenir :

- Un organigramme général du déroulement de votre programme avec des structo ou pseudo code quand cela est nécessaire.
- Le code source Visual Studio de votre solution

Il vous faut imprimer la grille d'évaluation présentée dans ce chapitre et la présenter au correcteur.

Au correcteur d'estimer et de définir la pertinence et la forme d'une remédiation si le résultat et pas acquis.

