Langage C#

Notes

Brice SIMEU, Adrien DELAURENS, Maxance MALLET, Elie SIDAWI
ISEN Lille, France

Sommaire

Lexique C#

1. Namespace : espaces de noms, équivalent du package en JAVA.
2. Nested Namespace : espace de noms imbriqués (imbrication de packages).
3. Console.ReadKey : maintenir la console ouverte après lancement et exécution du programme.
4. Console.Write : afficher des données sur la console avec retour à la ligne.
5. CLR : common langage runtime, définition du CLI (Common Language Infrastructure) par Microsoft qualifiant l'environnement d'exécution du programme.
6. Flux de control / Flux d'exécution : ordre dans lequel sont exécutées les instructions du programme et dépendant de la réaction de celui-ci à l'entrée qui lui est donnée.
7. Instruction de déclaration : déclaration d'une nouvelle variable ou constante. Assignation d'une valeur à celles-ci.
8. Instruction d'expression : expression qui calcule la valeur à stocker dans une variable.
9. Instruction de sélection : permettent de rediriger le flux d'exécution vers de nouvelles parties du programme selon certaines conditions.
10. Instruction d'itération : expression permettant d'exécuter une boucle d'instruction.
11. Instruction de saut : permettent de rediriger le flux d'exécution vers une autre section du programme.
12. Instruction de gestion des exceptions : gère les exceptions et rend possible leur récupération au cours de l'exécution.
13. Instruction de checked et unchecked : permet de vérifier les opérations autorisées à effectuer un dépassement de capacité ou non lors du stockage d'une valeur dans une variable trop petite.

I. Introduction

Le C# est un langage de programmation créé par Microsoft en 2002. Il peut être utilisé pour des applications Windows, Web, mobiles ou des jeux vidéo. La syntaxe du C# ressemble à celles du C, du C++ ou du JAVA. Le C# est un langage très populaire de Microsoft. Il fait partie d'un ensemble plus important, étant une brique de ce qu'on appelle le langage « .NET » qui regroupe trois langages : C#, F# et Visual Basic. « .NET » regroupe également 3 familles : le « Framework .NET », le «. NETCore » et le « Mono/Xamarian ». Le C# appartient à l'ensemble du « Framework .NET » qui est principalement utilisé pour la conception d'applications bureau, de sites web et de serveurs sur Windows. Le «. NETCore » est utilisé pour coder pour du multiplateformes (notamment sous Linux, MacOS et Windows), du web, des serveurs et des applications console et le « Mono/Xamarian » est le groupe permettant d'implémenter des applications sur le OS mobiles majeurs.

II. La structure d'un programme C#

Comme pour le C++, le JAVA ou les autres langages, le C# peut être composé d'un ou plusieurs fichiers. Chacun de ces fichiers peut contenir aucun ou plusieurs « Namespaces ». Un espace de nom (« Namespace ») peut contenir des types différents tels que des classes, des structures, des interfaces, des énumérations, des délégués ou d'autres « nested Namespaces ».

Example II-1 :

```
// La structure d'un programme C#
using System;
namespace EspaceDeNom1
{
    class Class1
    {… }
    struct Struct1
    {… }
    interface Interface1
    {…  }
    delegate int Delegue1();
    enum Enum1
    {… }
    namespace EspaceDeNomImbriquée
    {
        struct Struct2
        {… }
    }
    class MainClass
    {
        static void Main(string[] args)
        { //Le code principale... }
    }
}
```

III. Les types et les variables dans C#

<table>
<tr><th colspan="8">Type de valeur</th></tr>
<tr><td colspan="6">Type simple</td><td>Type d'enum</td><td>Type struct</td><td>Type valeur nullable</td></tr>
<tr><td>Entier Signé</td><td>Entier non signé</td><td>Charactéres unicode</td><td>Virgule flottante</td><td>Décimale haute</td><td>Boolean</td><td rowspan="5">enum E{…}</td><td rowspan="5">struct S{…}</td><td rowspan="5">null</td></tr>
<tr><td>sbyte</td><td>byte</td><td rowspan="4">char</td><td>float</td><td rowspan="4">decimal</td><td rowspan="4">bool</td></tr>
<tr><td>short</td><td>ushort</td><td rowspan="3">double</td></tr>
<tr><td>int</td><td>uint</td></tr>
<tr><td>long</td><td>ulong</td></tr>
</table>

<table>
<tr><th colspan="6">Type référence</th></tr>
<tr><td colspan="3">Type de classes</td><td>Types délégués</td><td>Type interface</td><td>Type de tableaux</td></tr>
<tr><td>Classe de base</td><td>Chaîne unicode</td><td>Type définis par</td><td rowspan="2">delegate int D()</td><td rowspan="2">interface I{…}</td><td rowspan="2">int[]
int[…,…]</td></tr>
<tr><td>object</td><td>string</td><td>class C{…}</td></tr>
</table>

Le type de valeur « decimal » indique 128-bits type de valeur. Le type décimal est plus précis que les type « float » ou « double ». Ce type est utilisé pour réaliser des calculs approximés où une précision plus importante est nécessaire, comme par exemple pour les calculs financiers.

Conversion possible :

- **sbyte** (signed byte) → short, int, long, float, double, decimal
- **byte** → short, ushort, int, uint, long, ulong, float, double, decimal
- **short** → int, long, float, double, decimal
- **ushort** (unsigned short) → int, uint, long, ulong, float, double, decimal
- **char** → ushort, int, uint, long, ulong, float, double, decimal
- **uint** (unsigned integer) → long, ulong, float, double, decimal
- **int** → long, float, double, decimal
- **long** → float, double, decimal
- **ulong** (unsigned long) → float, double, decimal
- **float** → double

Type C#	Type .NET Framework
byte	System.Byte
sbyte	System.SByte
char	System.Char
decimal	System.Decimal
double	System.Double
float	System.Single
int	System.Int32
uint	System.UInt32
long	System.Int64
ulong	System.UInt64
object	System.Object
short	System.Int16
ushort	System.UInt16
string	System.String
byte	System.Byte
sbyte	System.SByte

De plus, en C#, il existe le type « var » utilisé pour définir d'autres types de variables que l'on ne peut initialement pas déterminer ou connaître. Il est possible de l'utiliser pour les types de base tels que les « int », les « string », etc… Mais lorsqu'une valeur est assignée à ce type de variable, il devient impossible de la modifier. Par exemple, « var a=10 ; ».

Exemple III-1 « int-float » :

```
using System;
namespace Example
{
    class Variables
    {
        static void Main()
        {
            float pi=3.1416;
            int piInt=pi;
            Console.WriteLine("La valeur float");
            Console.Write(pi);//décimale
            Console.WriteLine("La valeur Int");
            Console.Write(piInt);//entier
        }
    }
}
```

Exemple III-2 « bool » :

```
            int a = 1;
            int b = 2;
            bool plusGrandAB = (a > b);
            bool egualeA1 = (a == 1);
            if (plusGrandAB)
             Console.WriteLine("A > B");
            else
             Console.WriteLine("A <= B");
```

```
            Console.WriteLine("plusGrandAB = " + plusGrandAB);
            Console.WriteLine("egualeA1 = " + egualeA1);
```

Exemple III-3 « char » :

```
            char ch = 'a';
            Console.WriteLine(
            "le code de '" + ch + "' est: " + (int)ch); // (int)ch pour lire la
valeur du char
            ch = 'b';
            Console.WriteLine(
            "le code de '" + ch + "' est: " + (int)ch);
            ch = 'A';
            Console.WriteLine(
            "le code de '" + ch + "' est: " + (int)ch);
            // Console output:
            // le code de 'a' est: 97
            // le code de 'b' est: 98
            // le code de 'A' est: 65
```

Exemple III-4 « null » :

Pour utiliser la valeur « null » en C#, il existe deux méthodes équivalentes :

```
            Nullable<int> i1 = null;//on peut utiliser int, double, bool
            int? i2 = null;
            int i=5;
            Console.WriteLine("i1" + i1);
            Console.WriteLine("i2" + i2);
            Console.WriteLine(i1.HasValue);// false
            i = i2.GetValueOrDefault();
            Console.WriteLine(i); // 0
```

Exemple III-5 conversion :

```
            double monDouble = 5.1d;
            Console.WriteLine(monDouble); // 5.1
            long monLong = (long)monDouble;
            Console.WriteLine(monLong); // 5
            monDouble = 5e9d; // 5 * 10^9
            Console.WriteLine(monDouble); // 5000000000
            int monInt = (int)monDouble;
            Console.WriteLine(monInt); // -2147483648
            Console.WriteLine(int.MinValue); // -2147483648
```

Example III-6 Concaténation de « string » :

```
            string text1 = "hello";
            string text2 = "world";
            Console.WriteLine(text1+" "+text2+"!!");//concatiner un String
```

IV. Lire et afficher des variables

Pour afficher des variables en C#, il existe plusieurs méthodes :

- Afficher une valeur sur une seule ligne.
  Exemple IV-1 :

```
int a = 7;
Console.Write(a);
```

- Afficher la première valeur puis passer à la ligne suivante pour afficher la seconde.
  Exemple IV-2 :

```
int a = 7;
int b = 1;
Console.WriteLine(a);
Console.WriteLine(b);
```

- Afficher des « string » et y ajouter des variables (équivalent du « printf » en C).
  Exemple IV-3 :

```
int a = 7;
int b = 4;
int c = 3;
int d = (a + b) / c;
int e = (a + b) % c;
Console.WriteLine($"quotient: {d}");
Console.WriteLine($"reste: {e}");
```

Afin de lire une entrée de valeur en C#, il est nécessaire d'utiliser la commande « `Console.ReadLine()` » qui renvoie la valeur en question au programme sous forme d'un type « string ». Ainsi, pour entrer une valeur numérique, il faut convertir la « string » obtenue en un type de valeur correspondant au type attendu « `int.Parse()` ».
Exemple IV-4 :

```
Console.Write("a = ");
int a = int.Parse(Console.ReadLine());
Console.Write("b = ");
int b = int.Parse(Console.ReadLine());
Console.WriteLine("{0} + {1} = {2}", a, b, a + b);
Console.WriteLine("{0} * {1} = {2}", a, b, a * b);
Console.Write("f = ");
double f = double.Parse(Console.ReadLine());
Console.WriteLine("{0} * {1} / {2} = {3}",a, b, f, a * b / f);
```

V. Les opérateurs

Cette catégorie regroupe l'ensemble des signes nécessaires pour exécuter des opérations ou des traitements sur les variables du script.

Catégorie	Operateurs
Arithmatique	-, +, *, /, %, ++, --
Logique	&&, ||, !, ^
Binaire	&, |, ^, ~, <>
Comparaison	==,!=, >, =, <=
Affectation	=, +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=
String et concatenation	
Conversion de type	(type), as, is, typeof, sizeof
Autre	., new, (), [], ?:, ??

L'ensemble de ces opérateurs possèdent un ordre de priorité lors de l'exécution du programme duquel ils font partie. Ainsi il n'y a pas de conflit lorsque plusieurs d'entre eux se retrouve dans la même instruction. Voici un tableau qui en dévoile une partie d'entre elles :

Priorité	Catégorie	Les opérateurs
1 (plus haute)	Unaire	! (non logique) nombre++ nombre-- -nombre
2	Arithmétique-multiplicative	* / % (modulo)
3	Arithmétique-additive	+ - somme et différence pour les types numériques + concaténation lorsqu'un ou 2 opérandes sont de type string
4	Relationnel	< > <= >=
5	Égalité	== !=
6	Logique	&& (ET) le ET est prioritaire au OU || (OU)
7	Assignation	= += -= *= /= %=

VI. Les instructions

- « if-else » (instructions conditionnelles) :
  Exemple VI-1 :

```
if (expression)
{
   si vrai
}
else
{
   si faux
}
```

- « switch » :
  Exemple VI-2 :

```
switch (caseSwitch)
```

```
{
    case 1:
        Console.WriteLine("Case 1");
    break;
    case 2:
        Console.WriteLine("Case 2");
    break;
    default:
        Console.WriteLine("Default case");
    break;
}
```

- Les boucles (instructions d'itérations) :
    - Les boucles « for » et « foreach » :
        - « for » :
          Exemple VI-3 :

```
for(int compteur = 0; compteur < 10; compteur++)
{
  Console.WriteLine($"Le compteur est {compteur}");
}
```

        - « foreach » :
          Exemple VI-4 :

```
var liste_ = new List<int> { 0, 1, 1, 2, 3, 5, 8, 13 };//liste
int compt = 0;
foreach (int element in liste_)
{
    compt++;
    Console.WriteLine($"Element #{compt}: {element}");
}
Console.WriteLine($"Number of elements: {compt}");
```

    - La boucle conditionnelles « do-while » et « while » :
        - « do-while » :
          Les instructions de la boucle sont réalisées une première fois avant de vérifier si la condition de boucle est bien vérifiée.
          Exemple VI-5 :

```
int compteur = 0;
do
{
  Console.WriteLine($"Le compteur est {compteur}");
  compteur++;
} while (compteur < 10);
```

        - La boucle « while » :
          Les instructions à l'intérieur de la boucle ne sont exécutées que si la condition de boucle est bien vérifiée.
          Exemple VI-6 :

```
int compteur = 0;
while (compteur < 10)
{
```

```
  Console.WriteLine($"Le compteur est {compteur}");
  compteur++;
}
```

- Les instructions de saut :
  - « break » :
    L'instruction « break » permet d'interrompre la boucle qui l'englobe ou l'instruction d'un switch dans laquelle elle figure. Le contrôle est passé à l'instruction qui suit l'instruction interrompue, le cas échéant.
    Exemple VI-7 :

```
for (int i = 1; i <= 100; i++)
{
    if (i == 5)
    {
        break;
    }
    Console.WriteLine(i);
}
```

  - « continue » :
    L'instruction « continue » permet de sauter l'exécution des instructions suivant celle-ci et de passer à une nouvelle itération de la boucle itérative qui l'englobe (for, foreach, while, do-while).
    Exemple VI-8 :

```
for (int i = 1; i <= 10; i++)
{
    if (i < 9)
    {
        continue;
    }
    Console.WriteLine(i);
}
```

  - « goto » :
    L'instruction « goto » transfère le contrôle du programme directement a une instruction ciblée. Une utilisation courante de « goto » consiste à transferer le contrôle a une étiquette « switch-case » ou à l'étiquette par défaut d'une instruction « switch ». Elle sert aussi à quitter des boucles fortement imbriquées.
    Exemple VI-9 :

```
switch (n)
{
    case 1:
        cost += 25;
    break;
    case 2:
        cost += 25;
        goto case 1;
    case 3:
        cost += 50;
    goto case 1;
    default:
```

```
        Console.WriteLine("Invalid selection.");
    break;
}
```

- « return » :
  L'instruction « return » met un terme à l'exécution de la méthode dans laquelle elle apparait et retourne le contrôle à la méthode d'appel. Elle permet également retourner une valeur (facultative ou non). Si la méthode est censée renvoyer une variable de type « void », l'instruction « return » peut être omise. Si l'instruction « return » est à l'intérieur d'un bloc « try », le bloc « finally » est exécuté avant que le contrôle retourne à la méthode d'appel.
  Exemple VI-10 :

```
static double calculer(int r)
{
    double a = r * r * Math.PI;
    return a;
}
```

VII. Les classes et les objets

Une classe est un type de référence. Lors de la création d'un objet, la variable à laquelle l'objet est affecté conserve uniquement une référence à son adresse mémoire. Lorsque l'adresse est affectée à une nouvelle variable, celle-ci fait toujours référence à l'objet d'origine. Les modifications apportées à une des deux variables sont répercutées sur la seconde puisqu'elles font référence au même objet.
Exemple VII-1 :

- Type « class » :

```
    public static void Main()
    {
        Etudiant etudiant1 = new Etudiant("Elie", "Sidawi");//declaration d'un etu-
diant
        Etudiant etudiant2 = new Etudiant//autre methode pour declarer un etudiant
        {
            Prenom = "Elie",
            Nom = "Sidawi2",
        };
        Etudiant etudiant3 = new Etudiant
        {
            ID = 12
        };
        Etudiant etudiant4 = new Etudiant
        {
            Prenom = "Elie",
            Nom = "Sidawi3",
            ID = 116
        };
        System.Console.WriteLine(etudiant1.ToString());//imprimer la sortie
```

```
        System.Console.WriteLine(etudiant2.ToString());
        System.Console.WriteLine(etudiant3.ToString());
        System.Console.WriteLine(etudiant4.ToString());
        etudiant1.ID = 112;
        System.Console.WriteLine("Etud 1 " + etudiant1.ToString());
        etudiant2.Prenom = "Jean";
        System.Console.WriteLine("Etud 2 " + etudiant2.ToString());
    }
}

public class Etudiant
{
    public Etudiant() //constructeur par default
    {
        Prenom = "";
        Nom = "";
    }
    public Etudiant(string Prenom, string Nom)//constructeur
    {
        Prenom = this.Prenom;
        Nom = this.Nom;
    }
    // Proprietes
    public string Prenom { get; set; }
    public string Nom { get; set; }
    public int ID { get; set; }

    public override string ToString()//override de ToString
    {
        return Prenom + "  " + ID;
    }
}
//Sortie:
//Elie  0
//Elie  0
//  12
//Elie  116
//Etud 1 Elie  112
//Etud 2 Jean  0
//Press any key to continue...
```

L'exemple ci-dessus est constitué de deux classes. La première est la classe programme qui contient le programme principal qui s'exécute à chaque fois. La deuxième est la classe « Etudiant » qui permet de créer une instance du type de cette classe (à l'aide du mot clé « new » et de ses constructeurs) dont les paramètres sont un prénom, un nom et un ID. Il est possible de convertir ces informations de types particuliers (« int », « float », …) en chaînes de caractères en utilisant la méthode de programme « ToString() ».

Les différences entre le C# et le JAVA sont les méthodes de « set » et « get » permettant d'initialiser les paramètres avec les valeurs souhaitées et de les récupérer malgré le caractère potentiellement privé de ces paramètres (seulement accessibles aux instances de la classe qui

les possède). En C#, il est plus simple de créer ces méthodes qu'en JAVA, en utilisant directement la syntaxe « `public string Prenom { get; set; }` ».

Il reste possible d'utiliser les mêmes méthodes que celle utilisées par JAVA comme la méthode « ToString() » permettant la conversion des variables en chaînes de caractères. Ces méthodes, déjà définie par défaut, peuvent être redéfinies en utilisant la mention « @Override » au-dessus de leur signature et en implémentant les nouvelles instructions qu'elles doivent exécuter.

Example VII-2 L'héritage :

```
using System;
public class Humain
{
    public Humain()
    {
        Console.WriteLine("Humain");
    }
}
public class Homme : Humain
{
    public Homme()
    {
        Console.WriteLine("Homme");
    }
}
public class ClasseProgramme
{
    static void Main()
    {
        new Homme();
    }
}
```

Dans cet exemple, on peut voir une classe fille nommée « Homme » qui hérite d'une classe mère nommée « Homme ». En C# il est possible d'utiliser le symbole « : » afin de remplacer le mot clé « extends » afin de manifester l'héritage d'une classe par une autre, ce qui diffère du JAVA où ce symbole n'existe pas dans cette situation. En C#, il est possible d'écrire « public class A extends B » mais aussi « public class A : B ».

Example VII-3 L'héritage :

```
using System;
public abstract class Forme
{
    protected int x;
    protected int y;
    public Forme()
    {
        Console.WriteLine("Forme est crée");
    }
```

```
    public Forme(int x, int y)
    {
        this.x = x;
        this.y = y;
    }
    public abstract void disp();
}
public class Cercle : Forme
{
    private int r;

    public Cercle(int r, int x, int y) : base(x, y)
    {
        this.r = r;
    }

    public override string ToString()
    {
        return String.Format("Cercle, r:{0}, x:{1}, y:{2}", r, x, y);
    }
    public override void disp()
    {
        Console.WriteLine("r={0}",r);
    }
}
public class Formes
{
    static void Main()
    {
        Cercle c = new Cercle(2, 5, 6);
        Console.WriteLine(c);
        c.disp();
    }
}
//Sortie:
//Cercle, r:2, x:5, y:6
//r=2
```

Dans cet exemple, on remarque l'utilisation d'une méthode de type « abstract ». La méthode de type « abstract » est déclarée dans la classe mère avec une signature de la forme suivante « `public abstract void disp();` ». Son fonctionnement doit ensuite être complètement défini dans les classes filles ne possédant pas le type abstract.

Exemple VII-4 Les interfaces :

```
using System;
public interface Int1
{
    void Info();
}
public class Class1 : Int1
```

```
{
    public void Info()
    {
        Console.WriteLine("Informer");
    }
}
public class TestInterface
{
    static void Main()
    {
        Class1 c = new Class1();
        c.Info();
    }
}
//Sortie
//Informer
```

Dans cet exemple, on aperçoit une classe nommée « Int1 ». Elle représente une interface qui contient les méthodes qui doivent être implémentées par la classe « class1 ». Une des méthodes de l'interface est la méthode « Info » qui est définie dans la classe fille « Class1 ». Lorsque celle-ci est appelée lors de l'exécution de la méthode « main », elle affiche le message : « Informer ». De plus, on remarque à nouveau une différence entre C# est JAVA avec la spécification de l'implémentation de l'interface par une classe en C# qui se note à l'aide du symbole « : » où en JAVA il est nécessaire d'utiliser le mot clé « implements ».

Exemple VII-5 Implémentation de plusieurs interfaces :

```
using System;
public interface Device
{
    void SwitchOn();
    void SwitchOff();
}
public interface Volume
{
    void VolumeUp();
    void VolumeDown();
}
public interface Pluggable
{
    void PlugIn();
    void PlugOff();
}
public class CellPhone : Device, Volume, Pluggable//pour faire interface avec plu-
sieurs interfaces
{
    public void SwitchOn()
    {
        Console.WriteLine("Switching on");
    }
    public void SwitchOff()
    {
```

```
        Console.WriteLine("Switching on");
    }
    public void VolumeUp()
    {
        Console.WriteLine("Volume up");
    }
    public void VolumeDown()
    {
        Console.WriteLine("Volume down");
    }
    public void PlugIn()
    {
        Console.WriteLine("Plugging In");
    }
    public void PlugOff()
    {
        Console.WriteLine("Plugging Off");
    }
}
public class MultipleInterfaces
{
    static void Main()
    {
        CellPhone cp = new CellPhone();
        cp.SwitchOn();
        cp.VolumeUp();
        cp.PlugIn();
    }
}
//Sortie
//Switching on
//Volume up
//Plugging In
```

Dans cet exemple, on remarque l'importance des interfaces dans les classes qui en implémentent plusieurs à la fois afin de définir une grande variété de méthodes. Dans ce cas, les interfaces facilitent notre travail et nous permettent de concevoir un programme évolutif.

Exemple VII-6 Les polymorphismes :

```
using System;
public abstract class Shape
{
    protected int x;
    protected int y;

    public abstract int Area();
}
public class Rectangle : Shape
{
    public Rectangle(int x, int y)
    {
```

```
        this.x = x;
        this.y = y;
    }
    public override int Area()
    {
        return this.x * this.y;
    }
}
public class Square : Shape
{
    public Square(int x)
    {
        this.x = x;
    }

    public override int Area()
    {
        return this.x * this.x;
    }
}
public class Polymorphism
{
    static void Main()
    {
        Shape[] shapes = { new Square(5),
            new Rectangle(9, 4), new Square(12) };

        foreach (Shape shape in shapes)
        {
            Console.WriteLine(shape.Area());
        }
    }
}
//Sortie
//25
//36
//144
```

Dans cet exemple dans lequel on définit une classe mère « Shape » dont héritent deux classes filles « Square » et « Rectangles », on peut s'apercevoir de l'importance des polymorphismes en C#. En effet, ces deux classes possèdent chacune une méthode de type « abstract » héritée de leur classe mère et nommée identiquement « area », permettant de déterminer l'aire de chaque figure. Cependant, celle-ci est implémentée différemment selon la classe à laquelle elle appartient. L'utilisation des polymorphismes que fait C# nous permet de les appeler indifféremment l'une ou l'autre et d'obtenir le bon résultat quel que soit l'objet ciblé.

Exemple VII-7 Les classes scellées :

On utilise une classe dite « sealed » afin définir des méthodes et des variables immuables (c'est le même principe que pour le type « final » en JAVA). Aucune classe ne peut alors hériter de cette classe. Ce mot clé est également applicable à des méthodes ou des propriétés afin d'éviter que des classes filles puissent modifier leur implémentation.

```
using System;
sealed class Math
{
    public static double GetPI()
    {
        return 3.141592;
    }
}
public class DerivedMath : Math
{
    public void Say()
    {
        Console.WriteLine("Derived class");
    }
}
public class CSharpApp
{
    public static void Main()
    {
        DerivedMath dm = new DerivedMath();
        Console.WriteLine(dm.Say());
    }
}
```

Dans l'exemple précédent, on a une classe de base « Math » typée sealed. L'intérêt d'utiliser cette classe est de donner des méthodes et des constantes au programmeur. Cette classe n'est pas créée pour hériter d'autres classes.
Quand on a essayé de dériver de cette classe on a reçu une erreur « 'DerivedMath' cannot derive from sealed class 'Math' ».
Alors pour bien utiliser ce type de classe, il faut que cette classe soit la fin de la chaîne (n'hérite d'aucune classe) comme l'example ci-dessous.

```
using System;
public class Mother
{
    public void printX()
    {
        Console.WriteLine("Mother class");
    }
}
sealed class Math : Mother
{
    public static double GetPI()
    {
        return 3.141592;
    }
}
public class CSharpApp
{
    public static void Main()
    {
        Math dm = new Math();
```

```
        Console.WriteLine(dm.GetPI());
    }
}
```

VIII. Les structures

Les structures sont des entités de type de valeur. Lors de la création d'une « struct », les valeurs réelles de de celle-ci sont assignées à la variable. Si l'on affecte la « struct » à une nouvelle variable, elle est copiée. Les deux variables contiennent donc deux copies distinctes de l'objet et peuvent être modifiées indépendamment l'une de l'autre.

Exemple-15 struct :

```
using System;
public struct Coordonne
{
    public int x, y;

    public Coordonne(int x, int y)
    {
        this.x=x;
        this.y=y;
    }
}
class TestCoords
{
    static void Main()
    {
        Coordonne coord1 = new Coordonne();
        Coordonne coord2 = new Coordonne(10, 10);
        Coordonne coord3 = coord2;
        Console.Write("Coord 1: ");
        Console.WriteLine("x = {0}, y = {1}", coord1.x, coord1.y);
        Console.Write("Coords 2: ");
        Console.WriteLine("x = {0}, y = {1}", coord2.x, coord2.y);
        Console.Write("Coords 3: ");
        Console.WriteLine("x = {0}, y = {1}", coord3.x, coord3.y);
        coord3.x = 2;
        coord3.y = 2;
        Console.Write("Coords 2 change: ");
        Console.WriteLine("x = {0}, y = {1}", coord2.x, coord2.y);
        Console.Write("Coords 3 change: ");
        Console.WriteLine("x = {0}, y = {1}", coord3.x, coord3.y);
        Console.ReadKey();
    }
}
//Sortie:
//Coord 1: x = 0, y = 0
//Coords 2: x = 10, y = 10
//Coords 3: x = 10, y = 10
//Coords 2 change: x = 10, y = 10
//Coords 3 change: x = 2, y = 2
```

Dans cet exemple, on a testé une « struct » appelée « Coordonne » dans un classe « Test ». Au début, trois instances de « Coordonnées », « coord1 », « coord2 » et « coord3 », ont été créées puis affichées. Pour finir, afin de tester l'application de la « struct » correctement, les valeurs de la variable « coord3 », qui est une copie de l'instance « coord2 », ont été modifiées. Cela a démontré que seules les valeurs de l'instance « coord3 » ont été changées.

IX. Les Tableaux

Les indices des éléments d'une chaîne de caractères débutent par l'indice 0.

- Tableaux à une seule dimension :
  int[] nomDuTableau = new int[nbDesIndices];
  Exemple IX-1 :

```
using System;
namespace ArrayApplication
{
    class Chaine
    {
        static void Main(string[] args)
        {
            int[] n = new int[10];
            for (int i = 0; i < 10; i++)
            {
                n[i] = i + 100;
            }
            foreach (int j in n)
            {
                int i = j - 100;
                Console.WriteLine("Element[{0}] = {1}", i, j);
            }
            Console.ReadKey();
        }
    }
}
//sortie
//Element[0] = 100
//Element[1] = 101
//Element[2] = 102
//Element[3] = 103
//Element[4] = 104
//Element[5] = 105
//Element[6] = 106
//Element[7] = 107
//Element[8] = 108
//Element[9] = 109
```

- Tableaux multidimensionnels (jusqu'a 32 dimension) :
  int[, ,] nomDuTableau = new int[nbDesIndices, nbDesIndices, nbDesIndices];
  Exemple IX-2 :

```
using System;
namespace ArrayApplication
{
```

```
    class Chaine
    {
        static void Main(string[] args)
        {
            int[,] a = new int[5, 2] { { 0, 0 }, { 1, 2 }, { 2, 4 }, { 3, 6 }, { 4,
8 } };
            int i, j;
            for (i = 0; i < 5; i++)
            {

                for (j = 0; j < 2; j++)
                {
                    Console.WriteLine("a[{0},{1}] = {2}", i, j, a[i, j]);
                }
            }
            Console.ReadKey();
        }
    }
}
//sortie
//a[0, 0] = 0
//a[0, 1] = 0
//a[1, 0] = 1
//a[1, 1] = 2
//a[2, 0] = 2
//a[2, 1] = 4
//a[3, 0] = 3
//a[3, 1] = 6
//a[4, 0] = 4
//a[4, 1] = 8
```

- Tableaux multidimensionnels en escaliers :
  C'est un tableau de tableaux. Ils peuvent tous posséder des tailles différentes.
  int[][] nomDeLaChaine = new int[10][];
  jaggedArray[0]= new int[15];
  jaggedArray[1] = new int[20];
  Exemple IX-2 :

```
using System;
namespace ArrayApplication
{
    class Chaine
    {
        static void Main(string[] args)
        {
            int[][] a = new int[][]{new int[]{0,0},new int[]{1,2},
            new int[]{2,4},new int[]{ 3, 6 }, new int[]{ 4, 8 } };
            int i, j;
            for (i = 0; i < 5; i++)
            {
                for (j = 0; j < 2; j++)
                {
```

```
                    Console.WriteLine("a[{0}][{1}] = {2}", i, j, a[i][j]);
                }
            }
            Console.ReadKey();
        }
    }
}
//Sortie:
//a[0][0] = 0
//a[0][1] = 0
//a[1][0] = 1
//a[1][1] = 2
//a[2][0] = 2
//a[2][1] = 4
//a[3][0] = 3
//a[3][1] = 6
//a[4][0] = 4
//a[4][1] = 8
```

X. Les méthodes

Une méthode est une suite d'instructions regroupées sous une signature qui comprend la visibilité de celle-ci, son nom, le type de variable qu'elle renvoie ainsi que ses arguments d'entrée avec leurs types. Notez qu'une méthode peut ne prendre aucun paramètre en entrée ou ne retourner aucun résultat. Pour créer une méthode sans retourner un résultat on utilise le type de renvoi « void ».

Exemple X-1 :

```
void nomDeLaMéthode(typeParamètre nomParamètre1, typeParamètre nomParamètre2)
{
    // Code à exécuter quand la méthode est appelée.
}
```

Pour créer une méthode qui retourne un résultat on donne le type de la variable que la fonction doit renvoyer.

Exemple X-2 :

```
typeDuRésultat nomDeLaMéthode()
{
    // Code à exécuter quand la méthode est appelée.
    return xxx;
}
```

Exemple X-3 Les méthodes :

```
using System;
class Test
{
    static void Main(string[] args)
    {
        int number1 = 3;
        int number2 = 2;
```

```
        int number3 = Multiply(number1, number2);
        Console.WriteLine(number3);
    }
    static int Multiply(int x, int y)
    {
        return x * y;
    }
}
//Sortie:
//6
```

Pour une classe de type « static », il est possible de faire appel à ses méthodes en invoquant simplement le nom de la classe en question, nul besoin d'en posséder une instance. Une seule copie d'un membre statique existe : quel que soit le nombre d'instances de la classe qui ont été créées, elles seront toutes rattachées à la même adresse des valeurs de l'instance initiale. Les méthodes et propriétés statiques ne peuvent pas accéder à des champs ou des événements non statiques dans leur type contenant. Elles ne peuvent pas non plus accéder à une variable d'instance d'un objet quelconque à moins qu'elle soit passée explicitement dans un paramètre de méthode. Les méthodes statiques peuvent être surchargées mais pas substituées, car elles appartiennent à la classe et non pas à une instance de la classe.

XI. Le Type enum

On utilise le type « enum » pour les énumérations. Il est préférable de définir un « enum » dans un espace de noms pour que toutes les classes dans cet espace puissent l'utiliser. Un « enum » peut être utiliser dans une classe ou une structure. L'indice de la première « enum » est 0 par défaut mais il est possible de le modifier.

Exemple XI-1 Définition d'un « enum » :

```
enum Jour {L, M, Me, J, V, S, D};
```

Cependant, il est donc possible de changer l'indice du premier élément constant de la liste de l'énumération. Il suffit d'ajouter le symbole « = » puis de définir la valeur du premier indice auquel on souhaite débuter le décompte.

Exemple XI-2 Changement d'indice :

```
enum Jour {L=1, M, Me, J, V, S, D };
```

Par défaut, un « enum » est de type « int » mas il est modifiable. Il est ainsi possible de lui attribuer d'autre types numériques tels que « byte », « sbyte », etc… Le type « char » n'est pas utilisable dans cette situation.

Exemple XI-3 Changement de type :

```
enum Jour : byte { L=1, M, Me, J, V, S, D };
```

Exemple XI-4 « enum » :

```
public class Test
{
    enum Jour { L=1, M, Me, J, V, S, D };
    static void Main()
```

```
    {
        int x = (int)Jour.L;
        int y = (int)Jour.J;
        Console.WriteLine("L = {0}", x);
        Console.WriteLine("J = {0}", y);
    }
}
/* Output:
   L = 1
   J = 4
*/
```

Selon cet exemple, afin de lire la valeur d'un « enum » on utilise la commande «`int x = (int)Jour.L;` », où le type de la variable d'accueil correspond au type des valeur de l'énumération.

XII. Les délégués

Le type délégué permet d'encapsuler des méthodes dans une variable qu'il est possible de passer en argument d'autres méthodes. Cela est possible si la signature de la méthode en question possède les mêmes arguments d'entrée et le même type de sortie que celle du délégué. Ce type se rapproche des caractéristiques des pointeurs du C++ mais sont orientés objets.
Pour déclarer un délègue, on peut utiliser la syntaxe suivante :

```
public delegate void Del(string message);
```

Exemple XII-1 Utilisation de délégués :

```
public delegate void Delegue(string message);
public static void methodeDelegue(string x)
{
    Console.WriteLine(x);
}
Delegue y = DelegateMethod;
y("Hello World");
```

Dans cet exemple, on crée initialement une variable de type délégué nommée « Delegue ».
On crée ensuite une fonction « methodeDelegue » qui a le même type d'argument d'entrée « String » nommé « x » et qui l'affiche. On crée alors une variable déléguée de type « Delegue » nommée « y » à laquelle on attribue la fonction « `methodeDelegue` ». Pour finir, on passe la chaîne de caractère « `"Hello World"` » en argument de la variable « y » contenant la fonction précédemment implémentée et qui l'affiche sur la console.

XIII. Les exceptions

En C#, on utilise les exceptions pour prévenir le développeur des erreurs qui peuvent survenir au moment de la compilation du programme. C# utilise les mêmes blocs (« try », « catch » et « finally ») que JAVA pour gérer les exceptions. Les exceptions peuvent être générées lors du CLR, par un .NET Framework, par d'autres bibliothèques ou par des instructions. La commande « throw » permet de créer et de faire remonter les potentielles erreur à travers le code.

Exemple XIV-1 Les exceptions :

```
class Test
{
    static double div(double x, double y)
    {
        if (y == 0)
            throw new System.DivideByZeroException();
        return x / y;
    }
    static void Main()
    {
        double a = 1, b = 0;
        double c = 0;

        try
        {
            result = div(a, b);
            Console.WriteLine("{0} / {1} = {2}", a, b, c);
        }
        catch (DivideByZeroException e)
        {
            Console.WriteLine("Tu as divisé par 0.");
        }
    }
}
```

Quelques exemples des types d'exceptions :

Microsoft.SqlServer.Server.InvalidUdtException
System.AccessViolationException
System.AppDomainUnloadedException
System.ArgumentException
System.ArithmeticException
System.ArrayTypeMismatchException
System.BadImageFormatException

XIV. Les Instructions LINQ

Les instructions LINQ sont des instructions qui servent à réaliser les requêtes nécessaires à une base de données afin d'en obtenir le contenu attendu afin de le manipuler et le traiter dans le reste du programme. Voici un exemple de requête en base de données effectué grâce à une instruction LINQ :

```
var linqExperts = from p in programmers
                  where p.IsNewToLINQ
                  select new LINQExpert(p);

Il est également possible de réduire la rédaction de ces requêtes grâce à des API,
telles que IEnumerable<T>, de C# qui permettent de condenser ces appels :

var linqExperts = programmers.Where(p => p.IsNewToLINQ)
```

```
                                .Select(p => new LINQExpert(p));
```

Cependant, il est conseiller d'adopter l'une où l'autre de ces deux méthodes de requêtes en base de données en fonction des critères suivants :

- Si le code de base utilise l'une ou l'autre de ces méthodes, il est conseillé de continuer à utiliser la même méthode afin de ne pas créer de rupture ou de confusion lors de la lecture de celui-ci ;
- S'il est nécessaire de définir l'étendue des variables dans les requêtes en fonction de la complexité, il est préconisé d'utiliser le format de base des instructions LINQ. Dans le cas contraire, il vaut mieux utiliser la syntaxe des APIs ;
- En fonction de l'aisance que l'utilisateur a dans l'un ou l'autre des formats possibles ;

Il est souvent plus simple d'utiliser la syntaxe de base des instructions LINQs car l'apprentissages de l'utilisation des différentes APIs peut être assez fastidieux. De plus la syntaxe des requêtes LINQ ne varient pas ou peu, peu importe le type de base de données à accéder, grâce à son mappeur ORM (Objet-Relationnal Mapper) minimaliste. Il est aussi possible d'écrire son propre fournisseur LINQ.
De plus, LINQ étant expressif, il est facilement convertible en d'autres objets comme par exemple en dictionnaire :

```
var petLookup = pets.ToDictionary(pet => pet.RFID);
```

XV. Base de Donnée – Excel

Nous avons réalisé notre base de données sur un tableur « Excel » car celle-ci était trop petite pour nécessiter une réelle utilisation de base de données sous « MySQL ». Elle ne compte que deux attributs et une seule table.

Afin d'utiliser Excel, il est nécessaire d'importer :

```
using Microsoft.Office.Interop.Excel;
```

Puis, il faut définir un objet _Excel :

```
using _Excel = Microsoft.Office.Interop.Excel;

string path = "";
_Application excel = new _Excel.Application();
Workbook wb;
Worksheet ws;
```

Il est ensuite nécessaire de créer une nouvelle instance de la classe. Il faut donc la définir.

```
Program excel = new Program(@"C:\Users\Utilisateur\Desktop\Score_Excel\Tst.xlsx", 1);
```

Les fonctions utiles pour Excel sont les suivantes :

```
excel.WriteToCell(0, 0, "Nom");//row,column,text
excel.ReadCell(0, 0)
excel.Save();

public Program(string path, int sheet)
```

```
        {
            this.path = path;
            wb = excel.Workbooks.Open(path);
            ws = wb.Worksheets[sheet];
        }
        public string ReadCell(int i, int j)
        {
            i++;
            j++;
            if (ws.Cells[i, j].Value2 != null)
            {
                return ws.Cells[i, j].Value2;
            }
            else { return ""; }
        }
        public void WriteToCell(int i, int j, string s)
        {
            i++;
            j++;
            ws.Cells[i, j].Value2 = s;
        }
        public void Save()
        {
            wb.Save();
            wb.Close();
        }
```

XVI. Description du jeu Unity

Le jeu que nous avons développé pendant ce cours de C# est un jeu de plateforme dans lequel le joueur incarne un personnage dont le but est de s'échapper vers un vaisseau spatial. Le jeu dispose d'une une vue à la 3e personne et à la 1ère personne. Les contrôles pour se déplacer sont relatifs à la position de la caméra et l'appui de la touche espace permet au joueur de sauter et ainsi de se déplacer de plateforme en plateforme. Celle-ci peuvent être immobiles ou en mouvement pour corser un petit peu le jeu. Le jeu dispose de plus d'une horloge pour évaluer la performance de temps du joueur pour atteindre l'arrivée, et les meilleurs temps sont enregistrés et conservés.

Mouvement d'obstacles :

Ce script est utilisé pour faire quelques obstacles 3D présents dans le jeu vidéo se déplacer d'un point à un autre de façon répétitive.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Diagnostics;

public class Move_1 : MonoBehaviour
{
    public GameObject[] waypoints;//we create an empty objects to use them as a way
points, used as a public to be changed from the unity
    int current = 0;    //see which way point we are using -->way point numb
    public float speed;//speed of move used as a public to be changed from the uni-
ty
    float WPradius = 1; //way point radius
    void Update()
    {
        if (Vector3.Distance(waypoints[current].transform.position, trans-
form.position) < WPradius)//we check the distance from the way point
        {
            current++;//increment the current position
            if (current >= waypoints.Length) //to calculate if we arrived to the
desired position
                current = 0; //reset
        }
        transform.position = Vector3.MoveTowards(transform.position, way-
points[current].transform.position, Time.deltaTime * speed); //moving the obstacle
    }
}
```

Temps :

Ce script est utilisé pour calculer le temps du jeu et l'afficher sur l'écran. Dans le cas où le personnage tombe, il frappe un cube transparent et le temps s'arrête et devient écrit en rouge.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class Timer : MonoBehaviour
{
    public Text timerText; //text to be displayed
    private float startTime;
    private bool finnished = false;

    // Start is called before the first frame update
    void Start()
    {
```

```
        timerText.color = Color.green; //initial timer text color is green
        startTime = Time.time; //initial time
    }
    // Update is called once per frame
    void Update()
    {
        if (finnished)//to finish time
            return;
        float t = Time.time - startTime; //current time
        string minutes = ((int)t / 60).ToString();//calculate minutes
        string seconds = (t % 60).ToString("f0");//calculate seconds
        timerText.text = "Timer : "+minutes + ":" + seconds;//display string
    }
    public void Finnish()
    {
        timerText.color = Color.red;//if finish set the timer text color red
        finnished = true;//and set finished as true

    }
}
```

Cube Transparent d'arrêt :

Ce code est utilisé pour voir s'il y a une collision entre le personnage et le cube d'arrêt.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;

public class StopCube : MonoBehaviour
{
    void OnTriggerEnter(Collider other)
    {
        try
        {
            GameObject.Find("TimeText").SendMessage("Finnish");//if the player hit
the transparent cube it will display finnish
        }
        catch (System.Exception e)
        {
            Console.WriteLine(e);//to see the error if it occurs
        }
    }
}
```

Base de donnée -.txt:

Ce script est utilisé pour enregistrer le temps et le nom du joueur sous deux fichiers : un pour le temps et l'autre pour les noms. Ce script va lire et enregistrer les valeurs par ordre croissant.

```
using System;
```

```
namespace TXT
{
    class WriteTextFile
    {
        static void Main()
        {
            //Get name and time
            string nom = "ElieSSS";    //TO BE ADDED FROM UNITY
            string temps = "7,1";    //TO BE ADDED FROM UNITY

            string[] nN = { " ", " ", " ", " ", " " };//String array for names
            string[] nT = { "9999999,0", "9999999,0", "9999999,0", "9999999,0",
"9999999,0" };//String array for times
            int len = 0;
            try
            {
                string[] nnT = System.IO.File.ReadAllLines(@"Times.txt");//used to
get the length of the saved data
                len = nnT.Length;
                Console.WriteLine("length: " + len);
            }
            catch(Exception e)
            {
                Console.WriteLine(e);
            }

            for (int i = 0; i < len; i++)
            {
                nT[i] = System.IO.File.ReadAllLines(@"Times.txt")[i];
                nN[i] = System.IO.File.ReadAllLines(@"Names.txt")[i];
            }

            int indice = 0;//Pour savoir notre indice
            for(int i=0;i<5;i++)// pour calculer l'indice
            {
                if (nN[i] == " ")
                {
                    indice = i;
                    break;
                }
            }

            if (nN[4]!=" ")//si elle est complete on la reset
            {
                nN = new string[5] { " ", " ", " ", " ", " " };
                nT = new string[5] { "9999999,0", "9999999,0", "9999999,0",
"9999999,0", "9999999,0" };
            }
            else// si elle n'est pas complete on ajout les nouveau valeurs
            {
                if (indice != 0)
```

```
                {
                    nN[indice] = nom;
                    nT[indice] = temps;
                    Console.WriteLine("indice: " + indice);
                }
            }

            if (nN[0]!=" ")//si il y a deja des valeurs
            {
                string temp1;
                string temp2;
                //Bubble sort to sort the two arrays
                for (int j = 0; j < 4; j++)
                {
                    for (int i = 0; i < 4; i++)
                    {
                        if (Convert.ToDouble(nT[i]) > Convert.ToDouble(nT[i +
1]))//si le temps i+1 est plus grand que le temps i
                        {
                            temp1 = nT[i + 1];
                            nT[i + 1] = nT[i];
                            nT[i] = temp1;

                            temp2 = nN[i + 1];
                            nN[i + 1] = nN[i];
                            nN[i] = temp2;
                        }
                    }
                }
                System.IO.File.WriteAllLines(@"Names.txt", nN);//set new values for
the names
                System.IO.File.WriteAllLines(@"Times.txt", nT);//set new values for
the times
                Console.WriteLine("entered in if cond");
            }
            else if(nN[0] == " ")// si la liste est initialement vide
            {
                nN[0] = nom;
                nT[0] = temps;
                System.IO.File.WriteAllLines(@"Names.txt", nN);
                System.IO.File.WriteAllLines(@"Times.txt", nT);
                Console.WriteLine("entered in else cond");
            }
        }
    }
}
```

Base de donnée -Excel:

Ce script est écrit au début du projet pour utiliser Excel comme une base de donnée. A la fin du projet, l'équipe a décidé d'utiliser un fichier « .txt » où le code est plus simple, plus court et plus efficace.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Microsoft.Office.Interop.Excel;

using _Excel = Microsoft.Office.Interop.Excel;
namespace Excel_Score
{
    class Program
    {
        private static int nbDeJoueurEnregistree = 0;

        static string[,] tToExcel2d = new string[20, 2];

        string path = "";
        _Application excel = new _Excel.Application();
        Workbook wb;
        Worksheet ws;
        public static void Main(string[] args)
        {
            //OpenFile();
            Program excel = new Pro-
gram(@"C:\Users\Utilisateur\Desktop\Score_Excel\Score_Excel\Tst.xlsx", 1);

            excel.WriteToCell(0, 0, "Nom");//row,column,text
            excel.WriteToCell(0, 1, "Temps");//row,column,text


            if(excel.ReadCell(0, 0)=="")
            {
                Console.WriteLine("BdD vide");
            }
            else
            {
                for(int i=1;i<20;i++)
                {
                    if(excel.ReadCell(i, 0) != "")
                    {
                        nbDeJoueurEnregistree++;    //compter le nombre des joueur
enregistrer dans notre bd
                    }
                    Console.WriteLine("Il y a " + nbDeJoueurEnregistree +
"joueur(s)");
                }
            }

            // Ici on doit entre les valeurs de Unity

            Console.WriteLine("Nom: ");
```

```
            string nom = Console.ReadLine();

            Console.WriteLine("temps");//il doit etre envoyer ou converti en double
            double temps= Convert.ToDouble(Console.ReadLine());

            //Valeurs a ajouter dans le bd: Nom et temps

            if(nbDeJoueurEnregistree>0)//si il y a des joueurs dans la table
            {
                for(int i=0;i< nbDeJoueurEnregistree;i++)
                {
                    tToExcel2d[i, 0] = excel.ReadCell(i, 0);
                    tToExcel2d[i, 1] = excel.ReadCell(i, 0);
                }
            }

            tToExcel2d[nbDeJoueurEnregistree, 0] = nom ;
            tToExcel2d[nbDeJoueurEnregistree, 1] = temps.ToString();

            ////////////////BUBBLE SORT ALGORITHM///////////

            double temp;
            string temp2;

            for (int j = 0; j <= 20 - 2; j++)//Bubble sort to sort the arrays from
the lowest time to the highest
            {
                for (int i = 0; i <= 20 - 2; i++)
                {
                    if (Convert.ToDouble(tToExcel2d[i, 1]) > Con-
vert.ToDouble(tToExcel2d[i+1,1]))
                    {
                        temp = Convert.ToDouble(tToExcel2d[i + 1, 0]);
                        tToExcel2d[i + 1, 0]= tToExcel2d[i, 0];
                        tToExcel2d[i, 0] = temp.ToString();

                        temp2 = tToExcel2d[i + 1, 1];
                        tToExcel2d[i + 1, 1] = tToExcel2d[i, 1];
                        tToExcel2d[i, 1] = temp2;
                    }
                }
            }

            ////////////////////////////////////////////////

            for (int j = 1; j <= nbDeJoueurEnregistree+1; j++)//inserting values
            {
                excel.WriteToCell(j, 0, tToExcel2d[j-1, 0]);//row,column,text
                excel.WriteToCell(j, 1, tToExcel2d[j-1, 1]);//row,column,text
            }
```

```
            excel.Save();//saving the results on the excel file
        }
        public Program(string path, int sheet) //crete excel class
        {
            this.path = path;
            wb = excel.Workbooks.Open(path);
            ws = wb.Worksheets[sheet];

        }

        public string ReadCell(int i, int j) //reading from a cell
        {
            i++;
            j++;
            if (ws.Cells[i, j].Value2 != null)
            {
                return ws.Cells[i, j].Value2;
            }
            else { return ""; }
        }

        public void WriteToCell(int i, int j, string s) //inserting values to cell
        {
            i++;
            j++;
            ws.Cells[i, j].Value2 = s;
        }
        public void Save()// saving function
        {
            wb.Save();
            wb.Close();
        }
    }
}
```

Rotation du bridget:

Code pour mettre le pont en rotation.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class bridgerotation : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
```

```
    {
        transform.Rotate(Vector3.up * Time.deltaTime * 10);
    }
}
```

Camera switch :

Pour changer de vue sur le personnage du jeu.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Camera_Switch : MonoBehaviour
{
    //Camera vue on first person;
    public GameObject vueFPS;
    //Camera vue on third person;
    public GameObject vueDeDos;
    /**
     * Initializing of the camera used at the beginning of the game;
     */
    private void Start()
    {

    }

    /**
     * Update is called once per frame;
     * This method enables the player to change the view of the game and switch
between the first and third person view;
     */
    void Update()
    {
        if (vueFPS.active == true)
        {
            vueDeDos.transform.position = vueFPS.transform.position;
            vueDeDos.GetComponent<Perso_Info_Printing>().nbOfLives =
vueFPS.GetComponent<Perso_Info_Printing>().nbOfLives;

        }
        else if (vueDeDos.active == true)
        {
            vueFPS.transform.position = vueDeDos.transform.position;
            vueFPS.GetComponent<Perso_Info_Printing>().nbOfLives =
vueDeDos.GetComponent<Perso_Info_Printing>().nbOfLives;
        }

        /*
         * Waits for the using of the "1" key of the left top of the keyboard to
enable the first person view and disable the third one;
         */
```

```
        if (Input.GetKeyDown(KeyCode.Alpha1))
        {
            //FPS
            //vueFPS.SetActive(true);
            vueFPS.active = true;
            //vueDeDos.SetActive(false);
            vueDeDos.active = false;

            /*vueFPS.depth = 0;
            vueDeDos.depth = -1;*/
        }


        /*
         * Waits for the using of the "3" key of the left top of the keyboard to
enable the third person view and disable the first one;
         */
        if (Input.GetKeyDown(KeyCode.Alpha3))
        {
            //TPS
            //vueFPS.SetActive(false);
            //vueDeDos.SetActive(true);
            vueDeDos.active = true;
            vueFPS.active = false;
            /*vueFPS.depth = -1;
            vueDeDos.depth = 0;*/
        }

    }
}
```

Phrase de départ :

Pour afficher la phrase de départ.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;

public class Context_sentence_expiration : MonoBehaviour
{
    //Variable of the sentence at the beginning of the game;
    private string sentence = "Vous êtes actuellement en fuite. Une myriade de
vaisseaux ennemis sont à vos trousse. Tentez de les semer!";
    //Panel of the context sentence;
    public GameObject beginnig_sentence;
    //Panel of the end of the game;
    public GameObject end_panel;
    //Text gameobject that display the sentence during the game;
    public GameObject sentenceText;
    //Beginning of the timer;
```

```
    private float start_time;


    // Start is called before the first frame update
    void Start()
    {
        //Start of the timer of the beginning;
        start_time = Time.time;
        //Printing of the panel with the sentence;
        beginnig_sentence.SetActive(true);
        //Initialization of the sentence of the game;
        sentenceText.GetComponent<TextMeshProUGUI>().SetText(sentence);
    }

    // Update is called once per frame
    void Update()
    {
        //Current time;
        float current_time = Time.time;
        //Calculation of the time spent;
        float period = current_time - start_time;
        /*
         * If the period of printing of the panel is higher than 5 seconds, it de-
sapears;
         */
        if(period>5 || end_panel.active)
        {
            beginnig_sentence.SetActive(false);
        }
    }
}
```

Main menu :

L'interface graphique initiale pour choisir « new game », les options, le tableau des scores ou « quit ».

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;


/**
 * Class of the Main Menu scripts of the game;
 */
public class MainMenu_Script : MonoBehaviour
{
    /**
     * This method enables the player to begin a new game form the Main Menu;
     */
    public void newGame()
    {
```

```
        //Call the next scene with the following build index of the scenes set-
tings;
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
        /*
         * Also possible to call with the name of the scene :
         * SceneManager.LoadScene("Level01");
         *
         * Also possible to call the scene with the buildindex :
         * SceneManager.LoadScene(1);
         */

    }


    /**
     * This method enables the player to completely quit the game from the Main
Menu;
     */
    public void quitGame()
    {
        //"Quit" function doesn't work natively in unity IDE, so a log is printed
on the shell to advertise us;
        Debug.Log("QUIT!");
        //The application is shut down;
        Application.Quit();
    }

}
```

Pause menu :

Interface graphique qui s'ouvre en pause.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;
using TMPro;
using System;
using UnityStandardAssets.Characters.FirstPerson;

public class Pause_Menu_Perso : MonoBehaviour
{
    //Boolean variable to idicate the state of the game, paused or not;
    public static bool gameIsPaused = false;
    //Targeted menu to print when "Pause" is requested;
    public GameObject pauseMenuUI;
    //Targeted field for player name;
    public GameObject namePlayer;
    public GameObject vueFPS;
    public GameObject first_sentence_panel;
```

```
    string word = null;
    int wordIndex = 0;
    string alpha;


    /**
     * Update is called once per frame;
     * It waits for the use of the "Escape" key to print the "Pause" menu or go out
of it;
     */
    void Update()
    {
        /*
         * The game wait for the use of the "Escape" key;
         */
        if(Input.GetKeyDown(KeyCode.Escape))
        {
            /*
             * It tests if the game is already on "Pause";
             */
            if(gameIsPaused==true)
            {
                // If it is the case, it get back to the game;
                Resume();
            }
            else
            {
                // If it is not the case, it prints the "Pause" menu;
                Pause();
            }
        }
    }


    /**
     * Resume method enables to get back to the game at the moment of the pause;
     */
    public void Resume()
    {
        //It changes the state of visibility and usability of the "Pause" Menu dur-
ing the game to make it unusable;
        pauseMenuUI.SetActive(false);
        //It gives back a normal speed to the game time;
        Time.timeScale = 1f;
        //It changes the state value of the game;
        gameIsPaused = false;
        Cursor.lockState = CursorLockMode.Locked;
        Cursor.visible = false;
        vueFPS.GetComponent<FirstPersonController>().enabled = true;
    }
```

```
    /**
     * This method enables to set the "Pause" Menu active;
     */
    public void Pause()
    {
        //It changes the state of visibility and usability of the "Pause" Menu dur-
ing the game to make it usable;
        pauseMenuUI.SetActive(true);
        //It stops the time speed;
        Time.timeScale = 0f;
        //It changes the state value of the game;
        gameIsPaused = true;
        Cursor.lockState = CursorLockMode.None;
        Cursor.visible = true;
        vueFPS.GetComponent<FirstPersonController>().enabled = false;
        first_sentence_panel.SetActive(false);
    }


    /**
     * This method enables to come back to the Main Menu;
     */
    public void LoadMenu()
    {
        //It calls a scene manager to come back to the scene called "Mai_Menu";
        SceneManager.LoadScene("Main_Menu");
        //The time runs normally;
        Time.timeScale = 1f;
    }


    /**
     * This method enables to quit the game from the "Pause" menu;
     */
    public void QuiTGame()
    {
        //"Quit" function doesn't work natively in unity IDE, so a log is printed
on the shell to advertise us;
        Debug.Log("Quit");
        //The application is shut down;
        Application.Quit();
    }


  /*  public void writeName()
    {
        while (Input.GetKeyDown(KeyCode.Escape)==false && In-
put.GetKeyDown(KeyCode.Break)==false)
        {
            String alphabet="_";
            foreach (KeyCode kcode in Enum.GetValues(typeof(KeyCode)))
            {
```

```
                if (Input.GetKeyDown(kcode))
                {
                    alphabet = kcode.ToString();
                }
            }
            wordIndex++;
            word += alphabet;
            namePlayer.GetComponent<TextMeshProUGUI>().SetText(word);
        }
    }*/


    /**
     * The "Replay" function enables the player to come back to the start of the
game;
     */
    public void Replay()
    {
        //It calls back the same scene to reload it from the start;
        SceneManager.LoadScene("Level_1");
        //The time runs normally;
        Time.timeScale = 1f;
        Cursor.lockState = CursorLockMode.Locked;
        Cursor.visible = false;
        vueFPS.GetComponent<FirstPersonController>().enabled = false;
    }
}
```

Impression des informations du personnage :

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using TMPro;
using UnityStandardAssets.Characters.FirstPerson;

public class Perso_Info_Printing : MonoBehaviour
{
    //Targeted Text Printer on the game screen;
    public GameObject lifeText;
    //Targeted Death Menu of the game;
    public GameObject failMenu;
    //Number of lives of the character;
    public GameObject vueFPS;
    public int nbOfLives;


    /**
     * Start is called before the first frame update;
     * Initialization of the variables of the game at its beginning;
```

```
     */
    void Start()
    {
        //The "Death" Menu is set inactive from the start of the game;
        failMenu.gameObject.SetActive(false);
        //The time runs normally;
        Time.timeScale = 1f;
        //The text printing the number of lives is set with the correct values;
        lifeText.GetComponent<TextMeshProUGUI>().text = "x"+nbOfLives.ToString();
    }


    /**
     * This function detects the entry of the player in a collider zone to decrease
the number of lives and print the "Death" Menu;
     */
    public void OnTriggerEnter(Collider other)
    {
        /*
         * Checks if the player enters a collider zone tagged "Vide" and if it re-
mains some lives to him;
         * If the player enters in the collider zone tagged "Vide" and its number
of lives is higher than 0, its number of lives decreases;
         */
        if(other.gameObject.tag=="Vide" && nbOfLives>0)
        {
            nbOfLives--;
            //The Lives text is updated;
            lifeText.GetComponent<TextMeshProUGUI>().text =
"x"+nbOfLives.ToString();
            //This line enables to play the Wilhem shout saved on the project as a
ressource;
            GetComponent<AudioSource>().Play();
        }
        /*
         * If the player enters in the collider zone tagged "Vide" and its number
of lives is 0, the "Death" Menu is printed;
         */
        else if (other.gameObject.tag=="Vide"&& nbOfLives==0)
        {
            //The "Death" Menu is enabled;
            failMenu.gameObject.SetActive(true);
            //It enables to take back the control of the cursor which was locked
because of the FPSController used in the scene;
            Cursor.lockState = CursorLockMode.None;
            //It makes the cursor visible for the player;
            Cursor.visible = true;
            //It desables the script of the FPSController to avoid moving the
screen while the menu is displayed;
            vueFPS.GetComponent<FirstPersonController>().enabled = false;
            //The time stops;
```

```
            Time.timeScale = 0f;
        }
    }

    // Update is called once per frame
    void Update()
    {

    }
}
```

Rotation du platforme:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class platformrotation : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        transform.Rotate(Vector3.up * Time.deltaTime*20, Space.World);
    }
}
```

Portail arrivée :

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using TMPro;
using UnityStandardAssets.Characters.FirstPerson;

public class Portail_arrivée : MonoBehaviour
{
    //Targeted Finish Menu to print at the end of the game;
    public GameObject FinishMenu;
    //Targeted Text to print the time score of the gamer;
    public GameObject timeScore;
    //GameObject of the FPS view;
    public GameObject vueFPS;
    //Timer where to take the value of the time of the player at the end of the
game;
```

```
    public Text chrono;
    //Counter to know the number of time the player go through the "Finish Portal";
    public int compteur = 0;


    /**
     * Method to dectect the entry of the player in the collider zone of the portal
and launch the "Finish" Menu only once;
     */
    public void OnTriggerEnter(Collider other)
    {
        /**
         * Checks if the object that is entered in the zone of the portal is tagged
as "Player" or not and if the counter is under 1;
         */
        if(other.gameObject.tag=="Player" && compteur<1)
        {
            //If the conditions are true, the "Finish" menu is enabled;
            FinishMenu.gameObject.SetActive(true);
            //The time speed of the game is reduced;
            Time.timeScale = 0.3f;
            //The time score text is updated with the time the player passed
through the "Finish" portal;
            timeScore.GetComponent<TextMeshProUGUI>().text= chrono.text+" s";
            //The timer is disabled;
            chrono.gameObject.SetActive(false);
            //The cursor is unlocked to be able to interact with the different but-
tons of the menu;
            Cursor.lockState = CursorLockMode.None;
            //It makes the cursor visible to the user;
            Cursor.visible = true;
            //It disables the FPSController script to avoid the FPS movements while
the menu display;
            vueFPS.GetComponent<FirstPersonController>().enabled = false;
            //The counter is increased of one;
            compteur++;

        }
    }

    /**
     * Start is called before the first frame update;
     * Initialization of the variables of the game;
     */
    void Start()
    {
        //The "Finish" Menu is desabled from the beginning of the game;
        FinishMenu.gameObject.SetActive(false);
        //The time runs normally;
        Time.timeScale = 1f;
        //The timer is enabled;
```

```
        chrono.gameObject.SetActive(true);
    }

    // Update is called once per frame
    void Update()
    {

    }
}
```

Zone death :

C'est le code de détection entre le cube transparent et le personnage du jeu.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using TMPro;
using UnityStandardAssets.Characters.FirstPerson;

public class Portail_arrivée : MonoBehaviour
{
    //Targeted Finish Menu to print at the end of the game;
    public GameObject FinishMenu;
    //Targeted Text to print the time score of the gamer;
    public GameObject timeScore;
    //GameObject of the FPS view;
    public GameObject vueFPS;
    //Timer where to take the value of the time of the player at the end of the
game;
    public Text chrono;
    //Counter to know the number of time the player go through the "Finish Portal";
    public int compteur = 0;


    /**
     * Method to dectect the entry of the player in the collider zone of the portal
and launch the "Finish" Menu only once;
     */
    public void OnTriggerEnter(Collider other)
    {
        /**
         * Checks if the object that is entered in the zone of the portal is tagged
as "Player" or not and if the counter is under 1;
         */
        if(other.gameObject.tag=="Player" && compteur<1)
        {
            //If the conditions are true, the "Finish" menu is enabled;
            FinishMenu.gameObject.SetActive(true);
            //The time speed of the game is reduced;
            Time.timeScale = 0.3f;
```

```
            //The time score text is updated with the time the player passed
through the "Finish" portal;
            timeScore.GetComponent<TextMeshProUGUI>().text= chrono.text+" s";
            //The timer is disabled;
            chrono.gameObject.SetActive(false);
            //The cursor is unlocked to be able to interact with the different but-
tons of the menu;
            Cursor.lockState = CursorLockMode.None;
            //It makes the cursor visible to the user;
            Cursor.visible = true;
            //It disables the FPSController script to avoid the FPS movements while
the menu display;
            vueFPS.GetComponent<FirstPersonController>().enabled = false;
            //The counter is increased of one;
            compteur++;

        }
    }

    /**
     * Start is called before the first frame update;
     * Initialization of the variables of the game;
     */
    void Start()
    {
        //The "Finish" Menu is desabled from the beginning of the game;
        FinishMenu.gameObject.SetActive(false);
        //The time runs normally;
        Time.timeScale = 1f;
        //The timer is enabled;
        chrono.gameObject.SetActive(true);
    }

    // Update is called once per frame
    void Update()
    {

    }
}
```

Timer script :

Script utiliser pour calculer le temps.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class Timer_Script : MonoBehaviour
{
    //Text intance enabling the count;
```

```
    public Text timerText;
    //Value of time at the beginning of the count;
    private float startTime;


    /**
     * Start is called before the first frame update;
     */
    void Start()
    {
        //Initializing of the first date of the time;
        startTime = Time.time;
    }

    /**
     * Update is called once per frame;
     * On each frame with take the time and we make the calculation of the time
spent;
     */
    void Update()
    {
        //Taking of current time;
        float t = Time.time - startTime;

        //Conversion in seconds and minutes with 2 numbers before the coma;
        string minutes = ((int)t / 60).ToString("D2");
        string seconds = ((int)t % 60).ToString("D2");

        //Printing of the time value obtained;
        timerText.text = minutes + " : " + seconds;
    }
}
```

Références

1. C#, Hilaire, (2019). *Introduction au C#*. [online] OpenClassrooms. Valable sur: https://openclassrooms.com/fr/courses/1526901-apprenez-a-developper-en-c/1527058-introduction-au-c [Accessed 7 Jan. 2019].
2. C# Guide, https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/
3. Forum C#, StackOverflow
4. Docs.unity3d
5. Unity3d-france
6. Youtube