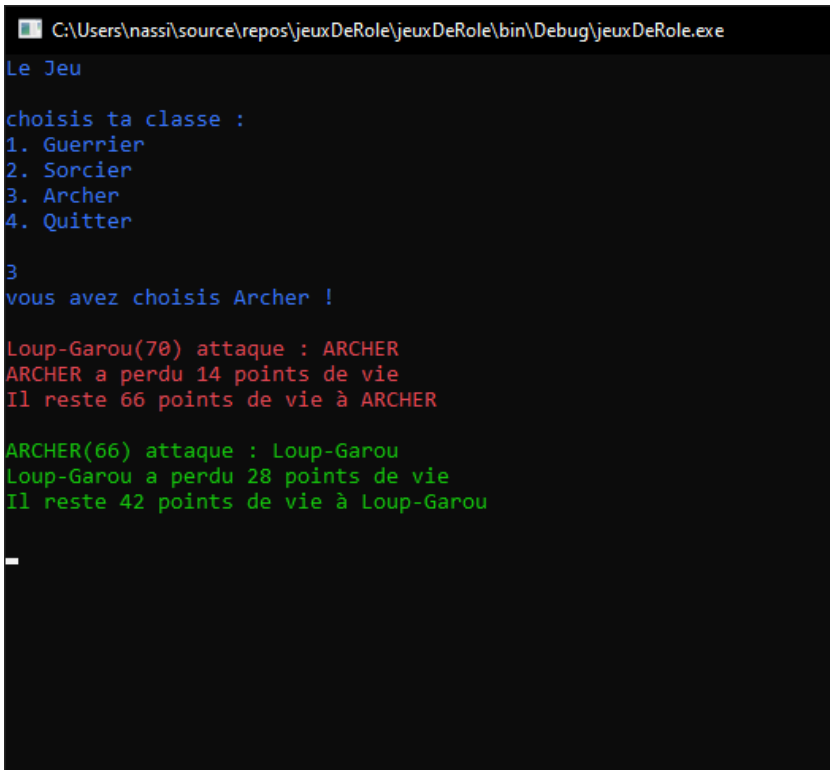
Documentation apprentisage de la programation orienter objet

Créer un jeu de rôle



Réaliser le 2/10/2021

Table des matières

Introduction 3
Shéma du projet 4
Représentation du projet sous forme d'un schéma 4
Projet en C# 5
Création de la classe Entite 5
Création de la classe Personnage 6
Création des Personnage 7
Création du Monstre 8
Classe program 8

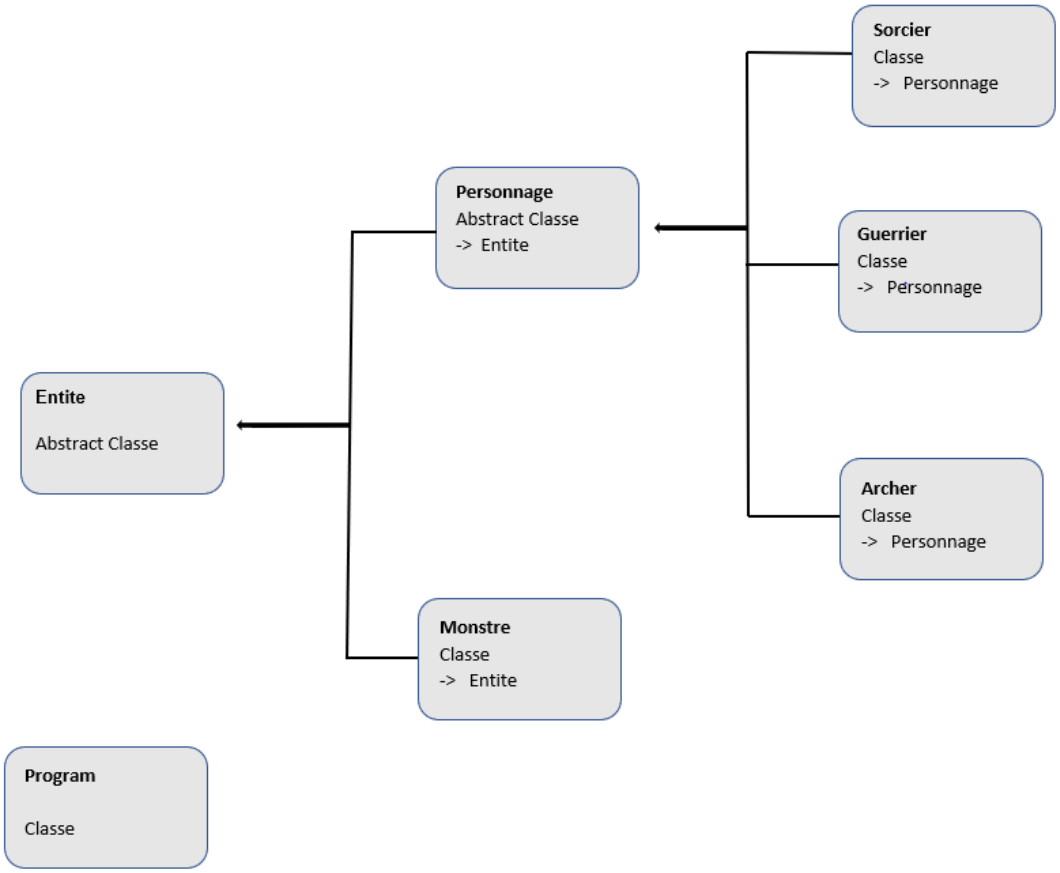
Introduction

En cette deuxième année de BTS SIO nous avons comme directive d'apprendre et d'utiliser la programmation orienter objet. Je me suis alors entrainé à suivre des projets comme celui-ci.

Ce projet ma permit d'en apprendre plus sur la création de classe ainsi que certain de ses caractéristiques comme Abstract. La création de fonction dont les éléments qui la forme son récupérer par des objets que l'on va chercher à récupérer et qui son préalablement crée.

Shéma du projet

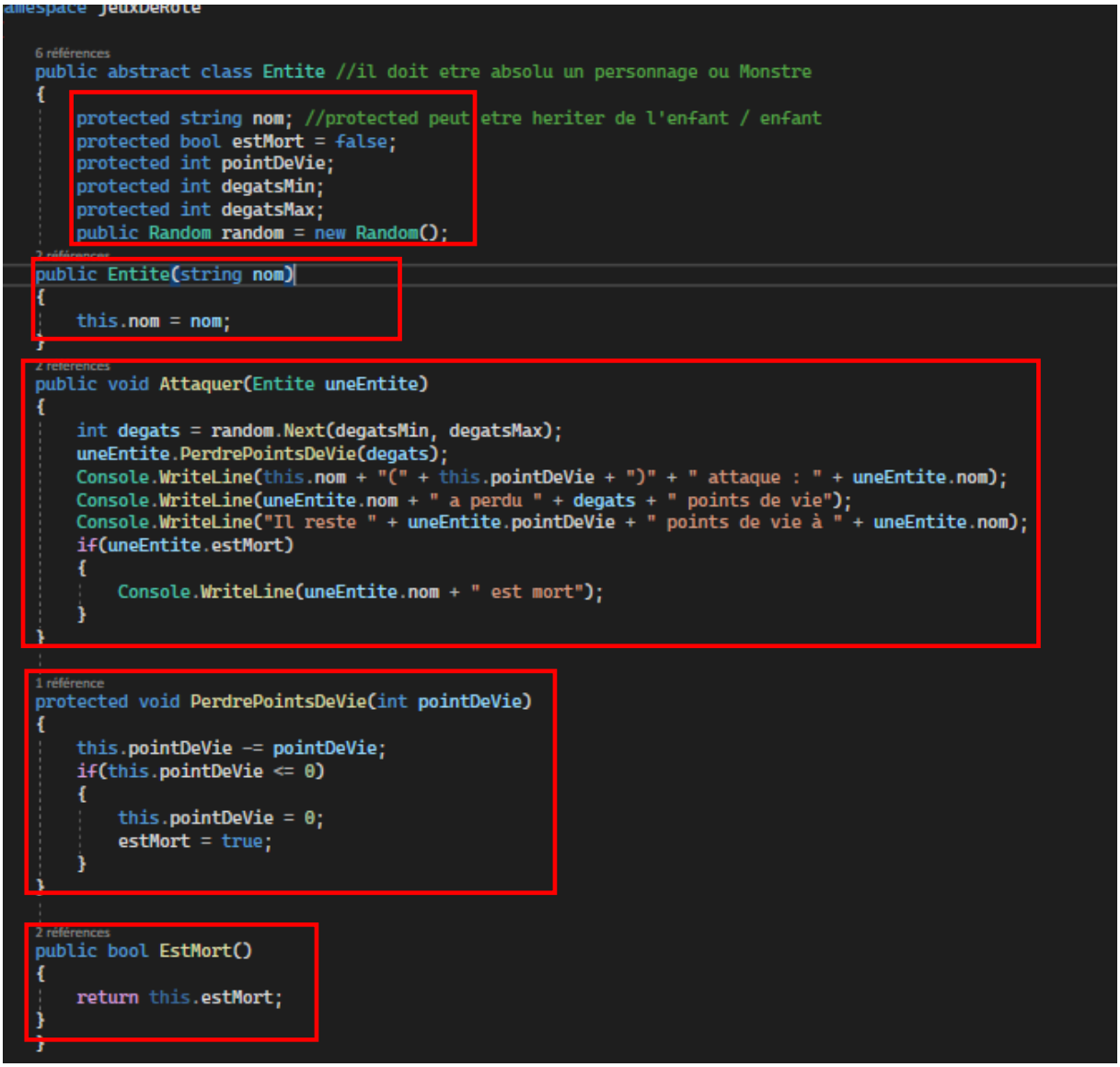
Représentation du projet sous forme d'un schéma



La -> représenter L'héritage
Abstract classe on oblige la classe à posséder des information exemple Personnage doit obligatoirement être soit un sorcier soit un Guerrier ou un Archer

Projet en C#

Création de la classe Entite



```
namespace JeuxDeRole

6 références
public abstract class Entite //il doit etre absolu un personnage ou Monstre
{
    protected string nom; //protected peut etre heriter de l'enfant / enfant
    protected bool estMort = false;
    protected int pointDeVie;
    protected int degatsMin;
    protected int degatsMax;
    public Random random = new Random();

2 références
public Entite(string nom)
{
    this.nom = nom;
}

2 références
public void Attaquer(Entite uneEntite)
{
    int degats = random.Next(degatsMin, degatsMax);
    uneEntite.PerdrePointsDeVie(degats);
    Console.WriteLine(this.nom + "(" + this.pointDeVie + ")" + " attaque : " + uneEntite.nom);
    Console.WriteLine(uneEntite.nom + " a perdu " + degats + " points de vie");
    Console.WriteLine("Il reste " + uneEntite.pointDeVie + " points de vie à " + uneEntite.nom);
    if(uneEntite.estMort)
    {
        Console.WriteLine(uneEntite.nom + " est mort");
    }
}

1 référence
protected void PerdrePointsDeVie(int pointDeVie)
{
    this.pointDeVie -= pointDeVie;
    if(this.pointDeVie <= 0)
    {
        this.pointDeVie = 0;
        estMort = true;
    }
}

2 références
public bool EstMort()
{
    return this.estMort;
}
}
```

1) On crée 6 propriétés qui définisse l'entite on les met en protected pour accéder à ses propriétés par les classe fille. Le public Random nous permettra de générer des nombres aléatoires
2) On crée le constructeur de la classe avec comme paramètre le nom. Le nom de la classe affectera les paramètres de l'entite
3) On crée une méthode attaquer qui représenter les dégâts infliger entre chaque tour.
   On crée une variable dégâts qui récupère un chiffre aléatoire entre les dégâts max et min d'une entité. Ensuite on appelle la fonction PerdrePointDeVie.
   On intègre une condition qui permet de connaitre l'état de vie des joueurs à partir de la méthode PerdrePointDeVie et affiche un message que le joueur est mort.

4) La méthode perdre point de vie réduit les point de vie de l'Entite par des points de vie en paramètre et on met la condition que si les point de vie sont égaux ou inferieur a 0 ont défini la méthode estMort a true.
5) On crée la méthode estMort que l'on utilisera pour arrêter le programme lorsqu'un des 2 joueur meurt.

Création de la classe Personnage

```
8 références
public abstract class Personnage : Entite // de entite
{
    private int niveau;
    public int experience;

    3 références
    public Personnage(string nom):base(nom)
    {
        this.nom = nom;
        niveau = 1;
        experience = 0;
    }

    1 référence
    public void gagnerExperience(int experience)
    {
        this.experience += experience;
        while(experience >= experienceRequise())
        {
            niveau += 1;

            Console.ForegroundColor = ConsoleColor.Yellow;
            Console.WriteLine("Bravo : vous avez atteint le niveau " + niveau + " !");


            pointDeVie += 15;
            degatsMin += random.Next(1, 4);
            degatsMax += random.Next(3, 8);
        }
    }

    2 références
    public double experienceRequise() //methode d'un personnage
    {
        return Math.Round(4 * (Math.Pow(niveau, 3) / 5));//Roud arrondi le résulta 4 * niveau
    }

    1 référence
    public string Caracteristiques()
    {
        Console.WriteLine("Bravo : vous avez atteint le niveau " + niveau + " !");
        return this.nom + "\n" +
            "Points de vie : " + pointDeVie + "\n" +
            "Niveau : " + niveau + "\n" +
            "Points d'experience : (" + experience + "/" + experienceRequise() + ")\n" +
            "Dégats : [" + degatsMin + ";" + degatsMax + "]";
    }
}
```

1) Il s'agit d'une classe abstraite donc nécessite un personnage elle hérite de hérite
En propriété on a mis le niveaux et l'expérience

2) Le constructeur qui hérite de entite doit aussi avoir un nom donc on l'appel avec : base(nom)
   Dans le constructeur on initialise le niveaux et l'expérience
3) Méthode gagnerExpérience augmente l'expérience.
   Dans la boucle while on regarde si l'expérience est supérieure ou égale à l'experienceRequise. Si oui elle va alors augmenter le niveau de 1 ainsi que les point de vie, dégâts min et max.
4) La méthode expérienceRequise calcule l'expérience requise en fonction du niveau.
   Ont appelé le module math que l'on arrondi au chiffre supérieure ou inférieure le calcule : (4 x le niveaux actuel puissance 3) diviser par 5. Ce calcule est un calcule récupère le Pokémon.
5) La méthode Caractéristique nous renvoie les informations du personnage du personnage une fois un tour gagner contre un monstre en affichent le niveaux actuelle et l'expérience requise pour le prochain niveau

Création des Personnage

Classe sorcier

```
{
    2 références
    internal class Sorcier : Personnage
    {
        1 référence
        public Sorcier(string nom) : base(nom)
        {
            pointDeVie = 90;
            degatsMin = 10;
            degatsMax = 25;
        }
    }
}
```

Classe Archer

```
    internal class Archer :Personnage
    {
        1 référence
        public Archer(string nom) : base(nom)
        {
            pointDeVie = 80;
            degatsMin = 15;
            degatsMax = 30;
        }
    }
}
```

Classe Guerrier

```
2 références
public class Guerrier : Personnage // public pour pouvoir a
{
    1 référence
    public Guerrier(string nom) : base(nom)
    {
        pointDeVie = 120;
        degatsMin = 10;
        degatsMax = 15;
    }

}
}
```

La classe hérite de Personnage qui hérite de Entite donc elle possède les méthodes de Entite
On crée juste un constructeur qui permet d'initialiser nos variables.
On initialise pointDeVie, degarsMin et degatsMax ceci est dû aux l'attribut qui sont en protected et donc les entités fille peuvent les utiliser.

Création du Monstre

```
3 références
internal class Monstre : Entite
{
    1 référence
    public  Monstre(string nom) : base(nom)
    {
        this.nom = nom;
        pointDeVie = 70;
        degatsMin = 7;
        degatsMax = 15;
    }
```

Monstre n'hérite pas de Personnage il faut donc crée le même constructeur que les personnages en rajoutent nom au constructeur

Classe program

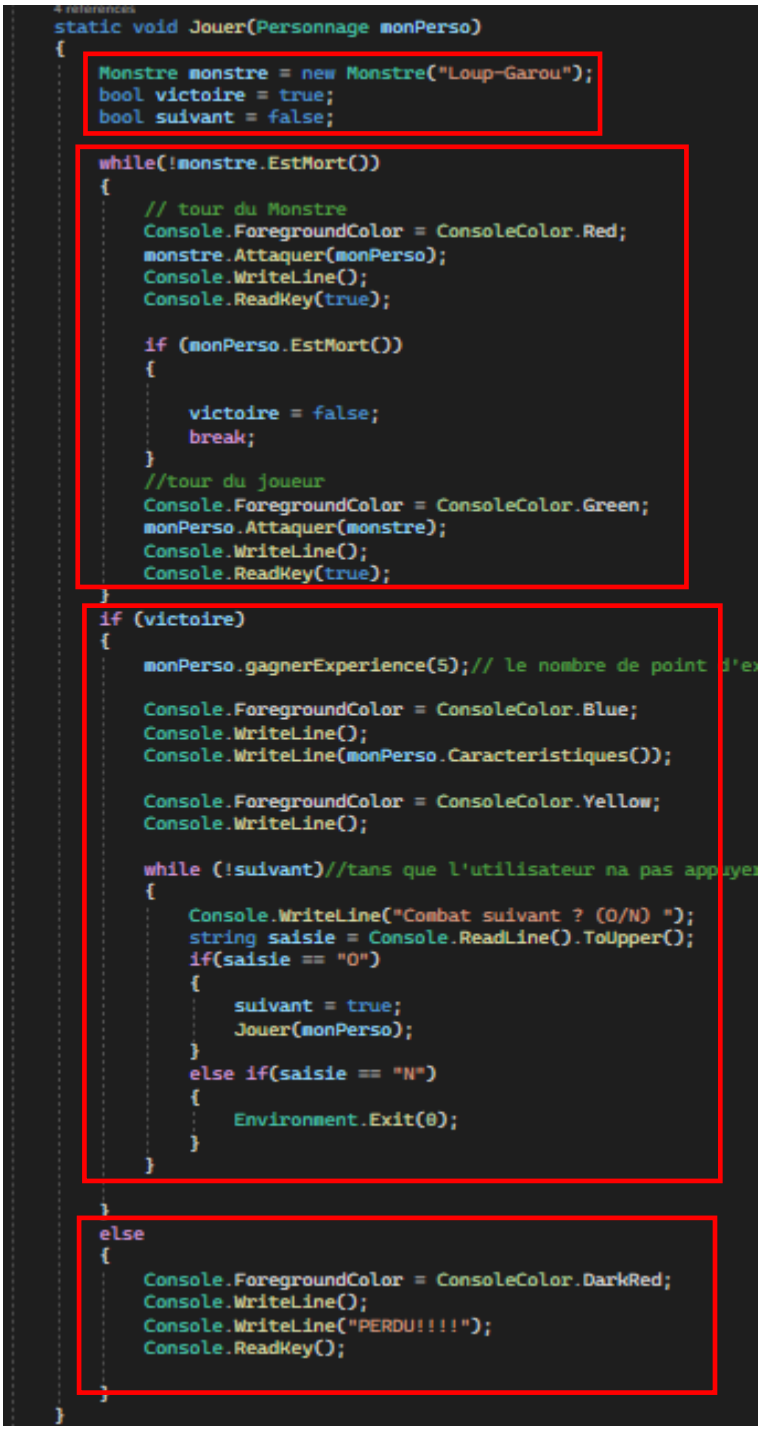
```
0 références
internal class Program
{
    0 références
    static void Main(string[] args)
    {
        Menu();
    }
```

```
static void Menu()
{
    Console.ForegroundColor = ConsoleColor.Blue;
    Console.WriteLine("Le Jeu");
    Console.WriteLine();
    Console.WriteLine("choisis ta classe : ");
    Console.WriteLine("1. Guerrier");
    Console.WriteLine("2. Sorcier");
    Console.WriteLine("3. Archer");
    Console.WriteLine("4. Quitter");
    Console.WriteLine();

    switch(Console.ReadLine())//lire le caractere que l'utilisateur tape
    {

        case "1":
            Console.WriteLine("vous avez choisis Guerrier !");
            Console.WriteLine();
            Jouer(new Guerrier("GUERRIER"));
            break;
        case "2":
            Console.WriteLine("vous avez choisis Sorcier !");
            Console.WriteLine();
            Jouer(new Sorcier("SORCIER"));
            break;
        case"3":
            Console.WriteLine("vous avez choisis Archer !");
            Console.WriteLine();
            Jouer(new Archer("ARCHER"));
            break;
        case"4":
            break;
    }
}
```

Cree une méthode statice menu car l'exécutable est en statice void Main ()
On appel la méthode Menu () dans Main()
Dans le méthode menu on affiche les règles en console avec « console. WriteLine ». Le joueur devra taper une des touches proposer pour pouvoir commencer ou arrêter le jeu. Entre 1 à 3 on attribut a chaque touche une entite personnage comme Guerrier ou l'on va mettre en paramètre le nom de celui-ci. On attribut ses informations à la fonction Jouer. Qui a en paramètre Personnage.



```
static void Jouer(Personnage monPerso)
{
    Monstre monstre = new Monstre("Loup-Garou");
    bool victoire = true;
    bool suivant = false;

    while(!monstre.EstMort())
    {
        // tour du Monstre
        Console.ForegroundColor = ConsoleColor.Red;
        monstre.Attaquer(monPerso);
        Console.WriteLine();
        Console.ReadKey(true);

        if (monPerso.EstMort())
        {

            victoire = false;
            break;
        }
        //tour du joueur
        Console.ForegroundColor = ConsoleColor.Green;
        monPerso.Attaquer(monstre);
        Console.WriteLine();
        Console.ReadKey(true);
    }
    if (victoire)
    {
        monPerso.gagnerExperience(5);// le nombre de point d'ex

        Console.ForegroundColor = ConsoleColor.Blue;
        Console.WriteLine();
        Console.WriteLine(monPerso.Caracteristiques());

        Console.ForegroundColor = ConsoleColor.Yellow;
        Console.WriteLine();

        while (!suivant)//tans que l'utilisateur na pas appuye
        {
            Console.WriteLine("Combat suivant ? (O/N) ");
            string saisie = Console.ReadLine().ToUpper();
            if(saisie == "O")
            {
                suivant = true;
                Jouer(monPerso);
            }
            else if(saisie == "N")
            {
                Environment.Exit(0);
            }
        }

    }
    else
    {
        Console.ForegroundColor = ConsoleColor.DarkRed;
        Console.WriteLine();
        Console.WriteLine("PERDU!!!!");
        Console.ReadKey();

    }
}
```

La méthode Jouer prend en paramètre un nouveau personnage sélectionner préalablement par l'utilisateur.

1) On crée des variables :
   On instancie le monstre avec comme nom (« Loup enragé »)
   On crée une variable victoire si le Personnage gagne contre le monstre et une suivant si l'utilisateur veut continuer après chaque tour.
2) On crée une boucle while qui prend en paramètre la vie du monstre, si celui-ci est toujours en vie.

Cette fonction fonctionne par tour d'attaque le premier représente le monstre qui attaque le Personnage.
Il appel la fonction Attaque et a comme paramètre le nom de l'utilisateur (soit archer, sorcier, guerrier)
Le ReadKey permet d'obliger le Personnage à toucher une touche pour passer au tour suivant.
Apres l'attaque du loup on vérifie avec la condition « if » si Personnage est toujours vivant.
Si non victoire est à false et arrête la boucle avec un « Break »
Si le personnage st toujours vivant c'est au tour de celui-ci et il va alors attaquer le loup.

3) Une fois la boucle terminer c'est-à-dire que le Personnage est toujours en vie et le Monstre mort. On attribut 5 point d'expérience au Personnage avec monParso.gagneExperience(5).
On va par la suite afficher les informations du personnage en appellent la fonction Caractéristique ().
Après l'affichage on demande à l'utilisateur d'appuyer sur une touche pour continuer ou arrêter de jouer. La touche est automatiquement en majuscule avec « toUpper » pour éviter certain problème.
4) Si la condition que le Personnage est toujours en vie est fausse alors le programme s'arrête et affiche le message « PERDU »