Cyriac Dupouey 1A

Rapport Cyriac Dupouey - Simulateur de Potager

1. Univers du jeu et spécificités

Ce projet est né de l'idée de concevoir un simulateur de potager à la fois réaliste, éducatif et interactif. Le joueur incarne un jardinier dans un pays fictif, ici le Brésil, et doit gérer un potager dans des conditions climatiques variables, le tout en respectant les cycles de la nature, les préférences des plantes, et en réagissant à des événements aléatoires comme la grêle ou l'arrivée de nuisibles.

Ce qui rend ce jeu intéressant, c'est la combinaison de plusieurs mécaniques :

la croissance végétale basée sur des conditions environnementales, la gestion de ressources (place, santé des plantes), et la prise de décision face à des urgences.

L'univers mélange donc aspects réalistes et éléments de gameplay dynamiques.

---

2. Déroulement d’une partie et possibilités offertes au joueur

Le jeu s'articule autour de deux modes :

2.1. Mode classique

Chaque tour simule une semaine :

- Une météo est générée (température, humidité, luminosité)
- Le joueur observe l’impact de ces conditions sur son potager
- Il choisit des actions d’entretien pour chaque plante (arroser, pailler, traiter, etc.)
- Il peut planter de nouvelles graines si la capacité du terrain le permet

2.2. Mode urgence

Déclenché aléatoirement si grêle, rongeurs, ou pluie :

- Le joueur doit réagir rapidement avec des choix spécifiques : épouvantail, bâche, etc.
- Ces choix ont un impact direct sur la survie des plantes

---

3. Exemple de déroulé complet en console

Voici un aperçu de ce que le joueur voit lors d'une session type. Cela illustre clairement le flux de jeu et les décisions à prendre.

3.1. Lancement du jeu

```
=== SIMULATEUR DE POTAGER - ProgAV 2025 ===

--- TERRAIN ACTUEL ---
Type: Terre | Espacement: 0,4 | Capacité: 20


==================================================
```

Un terrain est choisi aléatoirement parmi Terre, Sable, ou Argile. Le joueur est plongé immédiatement dans l'environnement avec un affichage clair.

3.2. Première semaine

```
=== SEMAINE 1 ===
Saison: Saison sèche | Température: 18°C | Humidité: 49% | Luminosité: 93%
Pluie: Non | Grêle: Non | Rongeur: Non
```

La météo donne le ton : les paramètres sont importants pour la survie des plantes.

3.3 Inventaire du potager

```
--- INVENTAIRE DU POTAGER ---
Capacité utilisée: 0/20 (Terrain: Terre)
```

Le joueur n'a encore rien planté, il peut alors décider de planter une graine.

3.4. Choix de graine à planter

```
Planter une nouvelle graine ? (oui/non)
oui
1. Tomate – Terrain: Terre, Espacement: 0,5, Temp: 15-30°C, Humidité min: 60%, Lumi min: 80%, Vie: 6 semaines, Maladie: mildiou (20%)
2. Carotte – Terrain: Sable, Espacement: 0,3, Temp: 10-25°C, Humidité min: 50%, Lumi min: 70%, Vie: 7 semaines, Maladie: vers (10%)
3. Pomme de terre – Terrain: Argile, Espacement: 0,6, Temp: 5-20°C, Humidité min: 65%, Lumi min: 60%, Vie: 6 semaines, Maladie: mildiou (25%)
```

Le joueur choisit ici la plante à ajouter, qui est ensuite ajoutée au potager.

3.5. Tour suivant avec plante en croissance

```
1
--------------------------------------------------
Appuyer sur Entrée pour passer à la semaine suivante...


==================================================
=== SEMAINE 2 ===
Saison: Saison sèche | Température: 15°C | Humidité: 68% | Luminosité: 84%
Pluie: Oui | Grêle: Non | Rongeur: Non
 Tomate | Santé: 100/100 | Vie: 6 | Croissance: 0,8 | Rendement: 4
Sur Tomate, quelle action souhaitez-vous effectuer ?
1=Arroser
2=Pailler
3=Désherber
4=Traiter
5=Installer serre
6=Barrière
7=Pare-soleil
8=Rien
```

```
--- INVENTAIRE DU POTAGER ---
1.  Tomate | Santé: 100/100 | Vie restante: 5
Capacité utilisée: 1/20 (Terrain: Terre)
```

Le joueur choisit une action hebdomadaire. Cela influence directement la santé et la croissance de la plante.

3.6. Apparition d'une urgence

```
=== SEMAINE 10 ===
Saison: Saison sèche | Température: 15°C | Humidité: 67% | Luminosité: 85%
Pluie: Oui | Grêle: Oui | Rongeur: Non

--- MODE URGENCE ACTIVÉ ---
Action d'urgence ?
1=Faire du bruit
2=Déployer bâche
3=Fermer serre
4=Installer épouvantail
5=Reboucher trous
6=Creuser tranchée
7=Ignorer
2
 Dommages limités par mesure d'urgence.
```

L'événement aléatoire ajoute de la tension. Le joueur peut sauver ses plantes avec la bonne action.

3.7. Mort de la plante (si conditions non respectées)

mort (conditions non remplies ou vieillesse):

```
Carotte est morte (conditions non remplies ou vieillesse).
```

Ou maladie:

```
 Tomate est tombée malade (mildiou) et a été détruite.
La plante est malade.
```

La simulation gère la disparition des plantes de façon claire et visuelle.

3.8. Fin de jeu

Toutes les plantes sont mortes. Fin du jeu.

Si toutes les plantes disparaissent, la partie se termine. L'utilisateur peut relancer une nouvelle session avec de nouvelles conditions.

---

4. Modélisation Objet et commentaires détaillés du code

Ce projet repose entièrement sur les principes de la programmation orientée objet, avec une séparation claire entre les entités (plantes, terrain, météo, urgences) et les mécanismes du jeu (simulation, interactions, conditions). Voici une revue complète de chaque classe utilisée dans le projet, commentée avec mon raisonnement personnel.

4.1. Classe Program (point d'entrée du jeu)

```
{
    /*Classe principale qui sert à démarrer notre simulateur de potager*/
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("=== SIMULATEUR DE POTAGER - ProgAV 2025 ===\n");

            /*On crée une instance de simulateur et on lance la simulation*/
            Simulateur simulateur = new Simulateur();
            simulateur.LancerSimulation();
        }
    }
```

Cette classe `Program` contient le point d'entrée principal de l'application. Elle affiche un message de bienvenue puis crée une instance du simulateur de potager et lance la boucle de jeu.

Utilité : elle démarre le jeu en initialisant tous les éléments nécessaires à la simulation, jouant ainsi un rôle fondamental pour exécuter l'application depuis la console.

4.2. Classe Simulateur

```
/*Cette classe gère toute la logique du simulateur de potager*/
class Simulateur
{
    /*Déclaration des champs nécessaires à la simulation*/
    private Pays pays;
    private Terrain terrain;
    private List<Terrain> terrainsDisponibles;
    private List<Plante> potager;
    private List<Plante> plantesDisponibles;
    private Random rand;
    private string saisonActuelle;
    private int semainesRestantesDansSaison;
```

Cette classe `Simulateur` est le cœur du jeu : elle regroupe tous les éléments nécessaires à la gestion du potager (pays, terrains, plantes, météo, saisons, etc.). Les champs déclarés permettent de suivre l'état du jeu au fil des semaines.

Utilité : elle centralise toute la logique de simulation, rendant le système modulable, cohérent et facile à maintenir. C'est elle qui pilote l'évolution du potager semaine après semaine

4.2.1. Initialisation de tous les éléments du jeu

```
/*Constructeur pour initialiser tous les éléments du jeu*/
public Simulateur()
{
    pays = new Pays("Brésil");
    rand = new Random();
    terrain = CreerTerrainAleatoire();
    terrainsDisponibles = new List<Terrain> { new Terrain("Terre", 0.4, 20), new Terrain("Argile", 0.6, 20), new Terrain("Sable", 0.3, 20)
    potager = new List<Plante>();
```

Ce constructeur initialise le simulateur de potager en configurant le pays, un terrain de départ aléatoire, et les terrains disponibles. Il prépare également une liste vide pour stocker les plantes que le joueur cultivera.

Utilité : il pose les bases du jeu en créant l'environnement initial dans lequel la simulation va évoluer.

4.2.2. Liste des plantes qu'on pourra planter

```
/*Liste des plantes qu'on pourra planter*/
plantesDisponibles = new List<Plante>
{
    new Plante("Tomate", "Terre", 0.5, 15, 30, 60, 80, 6, "annuelle", true, new List<string>{"Saison seche"}, 1.0, new List<s
    new Plante("Carotte", "Sable", 0.3, 10, 25, 50, 70, 7, "annuelle", true, new List<string>{"Saison des pluies"}, 0.8, new
    new Plante("Pomme de terre", "Argile", 0.6, 5, 20, 65, 60, 6, "annuelle", true, new List<string>{"Saison des pluies"}, 1.
};
```

Ce bloc initialise la liste des plantes disponibles que le joueur pourra planter dans le potager. Chaque plante est définie avec ses préférences de sol, ses besoins climatiques, sa durée de vie, sa maladie associée et son rendement.

Utilité : il fournit un catalogue de plantes que le joueur peut choisir, chacune ayant des caractéristiques uniques influençant sa croissance et sa survie dans le jeu

4.2.3. Initialisation des premières informations de simulation

```
terrain.AfficherCaracteristiques();
saisonActuelle = pays.GetSaisonAleatoire();
semainesRestantesDansSaison = 15;
```

Ce bloc initialise les premières informations de simulation juste après la configuration du terrain et des plantes. Il affiche les caractéristiques du terrain actuel, sélectionne une saison aléatoire pour démarrer la partie, puis fixe la durée d'une saison à 15 semaines.

Utilité : il permet d'informer le joueur sur son environnement de départ et de définir le contexte climatique dans lequel les premières plantes vont pousser.

4.2.4. Créer un terrain

```
/*Fonction pour créer un terrain au hasard*/
private Terrain CreerTerrainAleatoire()
{
    List<Terrain> choix = new List<Terrain> {
        new Terrain("Terre", 0.4, 20),
        new Terrain("Sable", 0.3, 20),
        new Terrain("Argile", 0.6, 20)
    };
    return choix[rand.Next(choix.Count)];
}
```

Cette fonction crée un terrain aléatoire parmi trois types : Terre, Sable ou Argile, chacun ayant un espacement et une capacité spécifique. Le choix est fait grâce à un générateur aléatoire.

Utilité : elle permet de varier les parties en proposant à chaque lancement un terrain de départ différent, ce qui influence directement les stratégies de plantation du joueur.

4.2.5. Affiche l'état actuel du potager

```
/*Affiche l'état actuel du potager*/
private void AfficherInventaire()
{
    Console.WriteLine("\n--- INVENTAIRE DU POTAGER ---");
    for (int i = 0; i < potager.Count; i++)
    {
        Plante p = potager[i];
        Console.WriteLine($"{i + 1}. 🌿 {p.Nom} | Santé: {p.Sante}/100 | Vie restante: {p.EsperanceDeVie}");
    }
    Console.WriteLine($"Capacité utilisée: {potager.Count}/{terrain.Capacite} (Terrain: {terrain.Type})\n");
}
```

Cette méthode affiche dans la console un résumé de l'état actuel du potager : les plantes présentes, leur santé, leur espérance de vie, ainsi que la capacité restante du terrain.

Utilité : elle permet au joueur de suivre facilement l'évolution de ses plantations et de prendre des décisions stratégiques (planter, soigner, changer de terrain, etc.)

4.2.6. Simulation semaine après semaine

```
/*Fonction principale qui simule semaine après semaine*/
public void LancerSimulation()
{
    int semaine = 1;

    while (true)
    {
        Console.WriteLine("\n================================================");
        Console.WriteLine($"=== SEMAINE {semaine} ===");

        /*Changement de saison toutes les 15 semaines*/
        if (semainesRestantesDansSaison == 0)
        {
            saisonActuelle = pays.GetSaisonAleatoire();
            semainesRestantesDansSaison = 15;
        }
```

Cette méthode lance la boucle principale du jeu, où chaque itération représente une nouvelle semaine de simulation. Elle affiche la progression temporelle et gère le changement de saison toutes les 15 semaines.

Utilité : c'est le moteur du gameplay : il fait avancer le temps, permet les actions du joueur, met à jour l'environnement et les plantes, et définit le rythme de la partie.

4.2.7. Génération des conditions météo de la semaine

```
/*On génère les conditions météo de la semaine*/
double temperature = pays.GenererTemperatureSelonSaison(saisonActuelle);
double humidite = pays.GenererHumiditeSelonSaison(saisonActuelle);
double luminosite = pays.GenererLuminositeSelonSaison(saisonActuelle);
bool pluie = rand.NextDouble() < 0.4;
bool grele = rand.NextDouble() < 0.1;
bool rongeur = rand.NextDouble() < 0.1;
```

Ce bloc génère les conditions climatiques de la semaine en cours selon la saison actuelle : température, humidité et luminosité sont calculées, et des événements aléatoires comme la pluie, la grêle ou les rongeurs peuvent apparaître.

Utilité : il permet de créer un environnement dynamique influençant directement la croissance, la santé ou la survie des plantes du joueur.

4.2.8. Affichage des conditions météo

```
/*Affichage des conditions météo*/
Console.WriteLine($"Saison: {saisonActuelle} | Température: {temperature}°C | Humidité: {humidite}% | Luminosité: {luminosite}%");
Console.WriteLine($"Pluie: {(pluie ? "Oui" : "Non")} | Grêle: {(grele ? "Oui" : "Non")} | Rongeur: {(rongeur ? "Oui" : "Non")}");
```

Ce bloc affiche à l'écran les conditions météorologiques de la semaine, ainsi que la présence ou non d'événements exceptionnels comme la pluie, la grêle ou les rongeurs.

Utilité : il informe le joueur de l’environnement actuel, essentiel pour adapter ses actions (soin des plantes, anticipation des urgences, choix des plantations).

4.2.9. Cas d'urgence

```
/*Si urgence, on déclenche le menu d'actions*/
if (grele || rongeur)
    Urgence.GererUrgence();
```

Ce code vérifie si une urgence climatique ou animale (grêle ou rongeur) a lieu cette semaine. Si c'est le cas, il appelle la méthode `GererUrgence()` qui affiche un menu d'actions au joueur.

Utilité : il permet d'intégrer des événements imprévus dans la simulation, obligeant le joueur à réagir rapidement pour protéger son potager.

4.2.10. Vérification de l'état de la plante

```
/*Pour chaque plante du potager, on vérifie son état*/
for (int i = potager.Count - 1; i >= 0; i--)
{
    Plante plante = potager[i];
    int conditionsOK = plante.NombreConditionsRespectees(terrain, temperature, luminosite, humidite);
    double proportion = conditionsOK / 5.0;

    if (proportion < 0.5 || plante.EsperanceDeVie <= 0)
    {
        Console.WriteLine($"❌ {plante.Nom} est morte (conditions non remplies ou vieillesse).\n");
        potager.RemoveAt(i);
        continue;
    }

    if (rand.NextDouble() < plante.ProbabiliteContamination)
    {
        Console.WriteLine($"⚠️ {plante.Nom} est tombée malade ({plante.Maladies[0]}) et a été détruite.\nLa plante est malade.");
        potager.RemoveAt(i);
        continue;
    }
```

Ce bloc parcourt toutes les plantes du potager pour vérifier leur état de santé. Il calcule combien de conditions sont respectées (sol, climat, lumière...) et applique les règles de survie : mort si moins de 50% des conditions sont remplies ou si la plante est trop vieille, ou si elle tombe malade aléatoirement.

Utilité : il permet de simuler la croissance, la mortalité et les maladies des plantes, en rendant le potager vivant et sensible aux choix du joueur comme aux aléas climatiques.

4.2.11. Affichage de l'état de la plante

```
/*Affichage de l'état de la plante*/
Console.WriteLine($"{plante.Nom} | Santé: {plante.Sante}/100 | Vie: {plante.EsperanceDeVie} | Croissance: {plante.VitesseCroi
plante.EffectuerAction(pluie);
plante.EsperanceDeVie--;
```

Ce bloc affiche un résumé de l'état de chaque plante : son nom, sa santé, son espérance de vie restante, sa vitesse de croissance ajustée et son rendement prévu. Ensuite, le joueur effectue une action d'entretien, puis la durée de vie de la plante diminue d'une semaine.

Utilité : il permet au joueur de suivre l'évolution de chaque plante en détail et de choisir une action hebdomadaire pour en maximiser la croissance et la durée de vie.

4.2.12. Inventaire

```
AfficherInventaire();

/*Si le terrain a de la place, on propose de planter une nouvelle graine*/
if (potager.Count < terrain.Capacite)
{
    Console.WriteLine("Planter une nouvelle graine ? (oui/non)");
    if (Console.ReadLine()!.ToLower() == "oui")
    {
        for (int i = 0; i < plantesDisponibles.Count; i++)
        {
            Plante p = plantesDisponibles[i];
            Console.WriteLine($"{i + 1}. {p.Nom} - Terrain: {p.TerrainPrefere}, Espacement: {p.EspacementMin},
        }

        int choix = int.Parse(Console.ReadLine()!);
        Plante planteChoisie = plantesDisponibles[choix - 1];
```

Ce bloc affiche d'abord l'état actuel du potager avec `AfficherInventaire()`, puis vérifie si le terrain a encore de la place. Si c'est le cas, il propose au joueur de planter une nouvelle graine. Toutes les plantes disponibles sont listées avec leurs caractéristiques, et le joueur peut en sélectionner une.

Utilité : il permet de gérer dynamiquement l'évolution du potager, en ajoutant de nouvelles plantes en fonction de la capacité disponible et des préférences stratégiques du joueur.

4.2.13. Nouvelle graine à planter par la suite

```
/*On crée une vraie nouvelle plante à partir de celle choisie (copie indépendante)*/
potager.Add(new Plante(
    planteChoisie.Nom,
    planteChoisie.TerrainPrefere,
    planteChoisie.EspacementMin,
    planteChoisie.TempMin,
    planteChoisie.TempMax,
    planteChoisie.HumMin,
    planteChoisie.LumiMin,
    planteChoisie.EsperanceDeVie,
    planteChoisie.Nature,
    planteChoisie.Comestible,
    new List<string>(planteChoisie.SaisonsSemis),
    planteChoisie.VitesseCroissance,
    new List<string>(planteChoisie.Maladies),
    planteChoisie.ProbabiliteContamination,
    planteChoisie.Rendement
```

Ce bloc ajoute une nouvelle plante dans le potager en créant une copie indépendante de la plante choisie par le joueur. Chaque attribut est dupliqué pour éviter toute modification indirecte de la plante originale dans la liste des options.

Utilité : il garantit que chaque plante cultivée est une instance unique, avec sa propre évolution, santé et durée de vie, ce qui est essentiel pour suivre individuellement chaque pousse dans le jeu

4.2.14. Si terrain plein?

```
else
{
    Console.WriteLine("🌾 Terrain plein ! Changement de terrain...\n");
    terrain = CreerTerrainAleatoire();
    potager.Clear();
}
```

Ce bloc s'exécute lorsque le terrain est plein (plus de place pour de nouvelles plantations). Il affiche un message d'alerte, génère un nouveau terrain aléatoire, puis vide complètement le potager pour repartir sur de nouvelles bases.

Utilité : il permet d'assurer la continuité du jeu en renouvelant automatiquement l'environnement de culture, tout en forçant le joueur à réadapter sa stratégie à un nouveau type de sol.

4.2.15. Passage a la semaine d'après

```
/*Fin de semaine : on passe à la suivante*/
Console.WriteLine("-------------------------------------------------");
Console.WriteLine("Appuyer sur Entrée pour passer à la semaine suivante...");
Console.ReadLine();
semaine++;
semainesRestantesDansSaison--;
```

Ce bloc marque la fin d'une semaine dans la simulation. Il affiche une séparation visuelle, attend que le joueur appuie sur Entrée pour continuer, puis passe à la semaine suivante et décrémente le compteur de semaines restantes dans la saison.

Utilité : il permet de rythmer la progression du jeu, en donnant au joueur un moment de pause avant d'enchaîner, tout en maintenant le suivi du cycle saisonnier.

4.2.16. Si plus aucune plante n'est vivante, on signe la fin du jeu

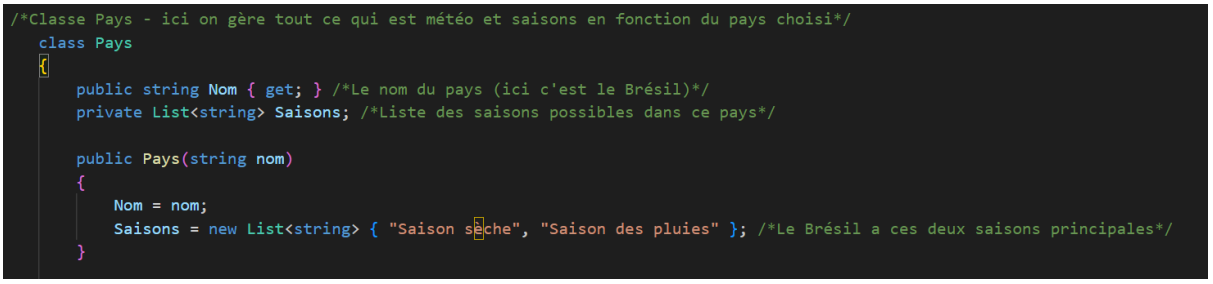
```
/*Si plus aucune plante n'est vivante, on termine le jeu*/
if (potager.Count == 0)
{
    Console.WriteLine("🌾 Toutes les plantes sont mortes. Fin du jeu.");
    break;
}
```

Ce bloc vérifie si le potager est vide, c'est-à-dire que toutes les plantes sont mortes. Si c'est le cas, un message de fin est affiché et la boucle principale du jeu est interrompue avec `break`.

Utilité : il permet de mettre fin logiquement à la partie lorsque le joueur a perdu toutes ses cultures, renforçant le côté réaliste et stratégique du simulateur.

4.3. Classe Pays

Elle permet de définir les saisons et de générer une météo adaptée au climat du pays sélectionné.



```
/*Classe Pays - ici on gère tout ce qui est météo et saisons en fonction du pays choisi*/
class Pays
{
    public string Nom { get; } /*Le nom du pays (ici c'est le Brésil)*/
    private List<string> Saisons; /*Liste des saisons possibles dans ce pays*/

    public Pays(string nom)
    {
        Nom = nom;
        Saisons = new List<string> { "Saison seche", "Saison des pluies" }; /*Le Brésil a ces deux saisons principales*/
    }
```

Cette classe Pays permet de gérer le climat et les saisons dans le jeu. Elle stocke le nom du pays (ici le Brésil) et les saisons possibles selon ce pays.

Utilité : elle permet de simuler des conditions météorologiques cohérentes avec l'environnement choisi, ce qui influence directement la croissance des plantes et les stratégies du joueur.

4.3.1. Saison, Température, Humidité, Luminosité

```
/*Cette méthode choisit une saison aléatoirement pour débuter ou après 15 semaines*/
public string GetSaisonAleatoire() => Saisons[new Random().Next(Saisons.Count)];

/*On génère une température réaliste selon la saison (saison sèche = plus chaud, saison des pluies = plus frais)*/
public double GenererTemperatureSelonSaison(string saison) =>
    saison == "Saison sèche" ? new Random().Next(15, 31) : new Random().Next(0, 16);

/*L'humidité aussi dépend de la saison : saison des pluies = très humide !*/
public double GenererHumiditeSelonSaison(string saison) =>
    saison == "Saison sèche" ? new Random().Next(40, 70) : new Random().Next(70, 100);

/*Et pareil pour la luminosité : plus de soleil en saison sèche, plus nuageux pendant les pluies*/
public double GenererLuminositeSelonSaison(string saison) =>
    saison == "Saison sèche" ? new Random().Next(60, 100) : new Random().Next(30, 60);
```

Ce bloc contient plusieurs méthodes qui permettent de générer aléatoirement les conditions climatiques du jeu en fonction de la saison actuelle (saison sèche ou saison des pluies).

Utilité : ces méthodes assurent que la température, l'humidité et la luminosité varient de manière réaliste selon la saison, ce qui rend chaque partie unique et impacte directement les chances de survie des plantes

4.4. Classe Terrain

Cette classe décrit les types de sols disponibles : leur espacement et leur capacité.

```
/*Classe Terrain - chaque potager est associé à un type de terrain*/
   class Terrain
   {
       public string Type { get; set; } /*Le type de sol : Terre, Argile, Sable...*/
       public double Espacement { get; set; } /*Distance nécessaire entre les plantes sur ce terrain*/
       public int Capacite { get; set; } /*Nombre maximum de plantes que ce terrain peut accueillir*/

       public Terrain(string type, double espacement, int capacite)
       {
           Type = type;
           Espacement = espacement;
           Capacite = capacite;
       }
```

Cette classe `Terrain` représente un type de sol disponible dans le jeu, comme la Terre, l'Argile ou le Sable. Chaque terrain a un espacement minimal entre plantes et une capacité maximale de plantation.

Utilité : elle permet de définir les contraintes physiques du potager, influençant combien de plantes peuvent être cultivées et dans quelles conditions elles peuvent pousser efficacement.

4.4.1. Affichage des caractéristiques actuelles du terrain

```
/*Affiche les caractéristiques actuelles du terrain à l'écran, utile à chaque changement de terrain*/
public void AfficherCaracteristiques()
{
    Console.WriteLine($"--- TERRAIN ACTUEL ---\nType: {Type} | Espacement: {Espacement} | Capacité: {Capacite}\n");
}
```

Cette méthode affiche dans la console les caractéristiques du terrain actif, notamment son type, l'espacement requis entre les plantes, et sa capacité maximale.

Utilité : elle permet au joueur de visualiser les propriétés du sol sur lequel il cultive, afin d'adapter ses choix de plantation en fonction des contraintes du terrain.

4.5. Classe Plante

```
/*Classe Plante - chaque plante a ses préférences et ses vulnérabilités*/
    class Plante
    {
        public string Nom { get; set; } /*Nom de la plante (ex: Tomate, Carotte...)*/
        public string TerrainPrefere { get; set; } /*Le type de sol idéal pour cette plante*/
        public double EspacementMin { get; set; } /*Espace minimum requis entre les plants*/
        public double TempMin { get; set; } /*Température minimale supportée*/
        public double TempMax { get; set; } /*Température maximale supportée*/
        public double HumMin { get; set; } /*Humidité minimale requise*/
        public double LumiMin { get; set; } /*Luminosité minimale requise*/
        public int Sante { get; set; } = 100; /*Santé actuelle de la plante (max 100)*/
        public int EsperanceDeVie { get; set; } /*Durée de vie en semaines*/
        public string Nature { get; set; } /*Annuelle, vivace, etc.*/
        public bool Comestible { get; set; } /*Est-ce une plante que l'on peut manger ?*/
        public List<string> SaisonsSemis { get; set; } /*Saisons pendant lesquelles on peut la semer*/
        public double VitesseCroissance { get; set; } /*Vitesse à laquelle elle pousse*/
        public List<string> Maladies { get; set; } /*Liste des maladies connues*/
        public double ProbabiliteContamination { get; set; } /*Chance qu'elle tombe malade*/
        public int Rendement { get; set; } /*Nombre de fruits ou légumes produits*/
```

La classe `Plante` décrit en détail les caractéristiques biologiques et environnementales d'une plante dans le jeu. Chaque plante a un nom, des préférences de sol, des seuils climatiques, une santé, une espérance de vie, et un rendement.

Utilité : elle permet de modéliser chaque espèce végétale avec ses propres besoins et vulnérabilités, influençant sa croissance, sa survie et sa productivité dans le potager du joueur.

C'est la plus détaillée. Chaque plante a :

- Des conditions idéales (température, humidité, lumière)
- Des maladies possibles avec une probabilité
- Une espérance de vie et un rendement

4.5.1. Amélioration de la santé des plantes

```
/*Améliore la santé de la plante, sans dépasser 100*/
public void AugmenterSante(int points) => Sante = Math.Min(Sante + points, 100);
```

Cette méthode augmente la santé de la plante d'un nombre de points donné, tout en s'assurant que la valeur ne dépasse pas 100 (valeur maximale).

Utilité : elle permet de récompenser les actions positives du joueur (comme arroser ou traiter) tout en gardant une limite réaliste à la vitalité de la plante

4.5.2. Réduction de la santé des plantes

```
/*Réduit la santé, mais elle ne descend jamais sous 0*/
public void DiminuerSante(int points) => Sante = Math.Max(Sante - points, 0);
```

Cette méthode diminue la santé de la plante d'un certain nombre de points, sans jamais aller en dessous de 0 pour éviter les valeurs négatives.

Utilité : elle permet de gérer les effets négatifs (manque de soin, conditions défavorables...) tout en assurant une cohérence logique dans la gestion des points de vie de la plante.

4.5.3. Vérification de la santé de la plante

```
/*Vérifie si la plante est morte (santé à 0)*/
public bool EstMorte() => Sante <= 0;
```

Cette méthode vérifie si la santé de la plante est nulle ou inférieure à zéro, ce qui signifie qu'elle est considérée comme morte dans le jeu.

Utilité : elle permet de détecter rapidement l'état vital d'une plante, afin de la retirer du potager si elle ne peut plus être sauvée.

4.5.4. Action pour entretenir la plante

```
/*Propose à l'utilisateur une action pour entretenir la plante cette semaine*/
public void EffectuerAction(bool pluie)
{
    Console.WriteLine($"Sur {Nom}, quelle action souhaitez-vous effectuer ?\n1=Arroser\n2=Pailler\n3=Dés
    string action = Console.ReadLine()!;

    switch (action)
    {
        case "1": /*Arroser (si pas déjà de la pluie)*/
            if (!pluie) AugmenterSante(10);
            else Console.WriteLine("💧 Trop d'eau!");
            break;

        case "2": /*Pailler*/
        case "3": /*Désherber*/
            AugmenterSante(5);
            break;

        case "4": /*Traiter*/
            AugmenterSante(8);
            break;

        case "5": /*Installer une serre*/
            AugmenterSante(6);
            break;
```

Cette méthode propose au joueur de choisir une action hebdomadaire pour entretenir la plante sélectionnée. Chaque choix (arroser, pailler, traiter...) a un effet positif spécifique sur la santé de la plante, sauf si le joueur ne fait rien, auquel cas elle perd de la vitalité.

Utilité : elle rend le jeu interactif et stratégique, en obligeant le joueur à prendre des décisions d'entretien adaptées aux conditions (ex. éviter d'arroser s'il pleut). Cela influence directement la survie et la productivité des plantes.

On vérifie que la plante a au moins 3/5 conditions réunies pour survivre.

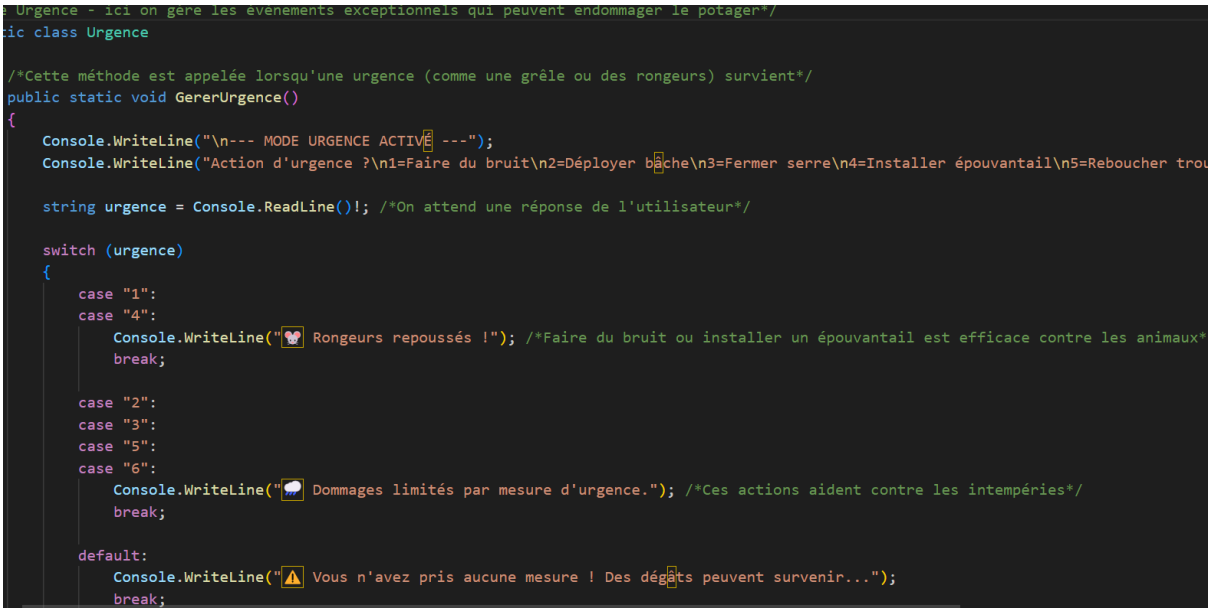
Interaction utilisateur :

plante.EffectuerAction(pluie);

Un menu s'affiche pour agir sur chaque plante. Les actions possibles ont des effets positifs (ex: pailler +5 santé), sauf si le joueur ignore la plante, auquel cas elle perd de la santé.

4.6. Classe Urgence

Cette classe contient une méthode unique :



```
e Urgence - ici on gère les évènements exceptionnels qui peuvent endommager le potager*/
tic class Urgence

/*Cette méthode est appelée lorsqu'une urgence (comme une grêle ou des rongeurs) survient*/
public static void GererUrgence()
{
    Console.WriteLine("\n--- MODE URGENCE ACTIVÉ ---");
    Console.WriteLine("Action d'urgence ?\n1=Faire du bruit\n2=Déployer bâche\n3=Fermer serre\n4=Installer épouvantail\n5=Reboucher trou

    string urgence = Console.ReadLine()!; /*On attend une réponse de l'utilisateur*/

    switch (urgence)
    {
        case "1":
        case "4":
            Console.WriteLine("🐭 Rongeurs repoussés !"); /*Faire du bruit ou installer un épouvantail est efficace contre les animaux*
            break;

        case "2":
        case "3":
        case "5":
        case "6":
            Console.WriteLine("🌧️ Dommages limités par mesure d'urgence."); /*Ces actions aident contre les intempéries*/
            break;

        default:
            Console.WriteLine("⚠️ Vous n'avez pris aucune mesure ! Des dégâts peuvent survenir...");
            break;
```

Cette classe `Urgence` gère les événements exceptionnels du jeu, comme les attaques de rongeurs ou les intempéries. La méthode `GererUrgence()` affiche un menu d'actions que le joueur peut utiliser pour protéger son potager en cas de danger.

Utilité : elle introduit une dimension imprévisible et réactive dans le jeu, obligeant le joueur à faire des choix rapides pour limiter les dégâts causés par des menaces extérieures (grêle, animaux, etc.).

En résumé, l'architecture objet est bien segmentée, chaque classe ayant un rôle précis. Cela a facilité le débogage, la maintenance du code et permettrait facilement d'ajouter de nouvelles fonctionnalités comme de nouveaux types de plantes ou d'urgences.

5. Tests réalisés

Plusieurs tests ont été effectués à différentes étapes du développement pour garantir un bon fonctionnement du simulateur :

5.1. Cycle des saisons : vérifié avec succès. Le changement de saison toutes les 15 semaines fonctionne bien, avec mise à jour de la météo.

- **5.2.Apparition des urgences** : validé. Des événements comme la grêle ou les rongeurs surgissent aléatoirement et déclenchent le menu approprié.
- **5.3. Interactions joueur-plante** : chaque action (arroser, pailler, etc.) a un effet mesuré sur la santé de la plante.
- **5.4. Conditions de mort** : si une plante a moins de 3 conditions favorables, elle meurt, ce qui est bien géré par le programme.
- **5.5. Tests d'interface console** : l'affichage est propre, clair, et permet de suivre facilement l'évolution du potager semaine après semaine.

Ces tests m'ont permis de corriger plusieurs oublis et d'améliorer la robustesse du simulateur.

6. Gestion de projet

Le projet a été réalisé en autonomie, en respectant une planification en trois grandes étapes

- Phase 1 : Conception et architecture
  - Rédaction des classes de base (Plante, Terrain, Pays)
  - Définition de la logique globale de jeu
- Phase 2 : Implémentation
  - Développement des interactions entre les classes
  - Gestion des cycles de jeu et des urgences
- Phase 3 : Tests et ajustements
  - Simulations longues (plusieurs saisons)
  - Ajustement des probabilités et des rendements

---

7. Bilan critique et perspectives d'amélioration

Réussites

- Le jeu est fonctionnel, stable, sans crash ni erreur de logique.
- L'architecture orientée objet est bien exploitée, avec une séparation claire des responsabilités.
- Les interactions sont riches, et les conditions de croissance des plantes sont respectées.

Limitations

- Je n'ai pas fait de quadrillage comme beaucoup d'autres ont fait car mon niveau actuel d'informatique et le temps dont je disposais ne le permettaient pas.
- Il n'existe pas encore de système de sauvegarde ou de chargement de partie.
- Le joueur n'a pas encore d'objectifs clairs ou de progression à long terme.

Problèmes rencontrés

Pendant le développement du simulateur de potager, j'ai rencontré plusieurs difficultés techniques qui m'ont demandé des ajustements progressifs. Un des premiers enjeux a été de gérer correctement la suppression des plantes mortes dans la boucle de simulation. J'ai dû opter pour un parcours inversé de la liste afin d'éviter des erreurs d'indexation, ce qui est un point souvent délicat lorsqu'on modifie une collection pendant son parcours.

J'ai aussi dû faire attention à ce que chaque plante instanciée soit bien indépendante, notamment au niveau de l'espérance de vie. Cela m'a obligé à créer des copies distinctes des objets pour éviter que plusieurs plantes partagent par erreur les mêmes références et donc les mêmes valeurs de vie.

Un autre point de réflexion a été l'équilibrage de l'aléatoire. Les conditions météorologiques devaient rester cohérentes avec les saisons sans pour autant rendre le jeu trop imprévisible ou injouable. J'ai donc ajusté les plages de valeurs pour conserver une part de difficulté tout en maintenant une certaine logique climatique.

La vérification des conditions de croissance des plantes a aussi demandé un peu de travail. J'ai mis en place une méthode pour évaluer si au moins 50 % des conditions nécessaires sont respectées, ce qui permet de rendre le système plus réaliste sans être trop punitif.

L'intégration des événements d'urgence a été un vrai ajout en termes de gameplay, mais il a fallu veiller à ce que leur déclenchement n'interrompt pas de manière incohérente le déroulement classique des tours de jeu.

Enfin, j'ai aussi passé du temps à soigner l'affichage console, car un retour clair pour l'utilisateur est essentiel dans ce type de simulation. J'ai utilisé des séparateurs, des récapitulatifs et un affichage structuré pour que le joueur puisse suivre l'évolution de ses plantes facilement et sans confusion.

Ces différentes étapes m'ont permis de consolider ma compréhension des listes, des objets, des boucles, mais aussi de la structuration générale d'un programme orienté utilisateur.