Rapport Projet Ensemenc

Première année Groupe 4

Auteur : Abdessamia AIT RAI / aarai@ensc.fr
2024/2025

1. Introduction

Ce projet de programmation avancée m'a été présenté dans le cadre du module CO6SFMA0 et devait être rendu pour le vendredi 23 mai 2025. L'objectif était de concevoir et développer un simulateur de potager en mode console utilisant le langage C# avec une approche résolument orientée objet.

J'ai opté pour une organisation agile :

- Définition des spécifications et architecture globale
- Développement par composants (plantes, animaux, météo)
- Intégration continue : via GitHub avec revue de code mutuelle

Le jeu que j'ai développé, baptisé **ENSemenC**, plonge le joueur dans la gestion d'un potager virtuel où il doit :

- Cultiver diverses plantes (comestibles, ornementales, imaginaires)
- Gérer les aléas climatiques et les interactions animales
- Développer son jardin grâce à un système économique

Contraintes techniques respectées :

- Développement en C# avec .NET 8
- Application console avec interface ergonomique
- Architecture POO poussée (abstraction, héritage, polymorphisme)
- Gestion de projet via GitHub (dépôt: https://github.com/PROGAV-PRJ25/projet-ensemenc-aarai03.git)
- Code documenté suivant les conventions camelCase
- Tests unitaires et d'intégration systématiques

Particularité notable : notre simulation intègre deux modes de jeu distincts :
1. Un **mode classique** au tour par tour pour la gestion stratégique
2. Un **mode urgence** en temps réel pour gérer les événements imprévus

Ce rapport détaillera ma démarche, depuis la conception objet jusqu'aux défis techniques rencontrés, en passant par des choix d'implémentation originaux.

2. Gestion de Projet

2.1. Organisation et méthodologie de travail

En tant que seul membre sur ce projet, j'ai dû adopter une approche structurée pour gérer efficacement la complexité du développement orienté objet (POO) et l'étendue des fonctionnalités à implémenter.

Pour optimiser mon temps et garantir une progression fluide, j'ai :
1. Établir un planning détaillé en découpant le projet en tâches claires (conception, développement des classes principales, implémentation des fonctionnalités, tests).
2. Travaillé par composants en me concentrant successivement sur les plantes, les animaux, la météo et l'interface, afin d'éviter la dispersion.
3. Prioriser la conception avant le codage : j'ai d'abord défini l'architecture objet (diagrammes UML, relations entre classes) pour anticiper les dépendances et faciliter l'intégration.

Cette rigueur m'a permis de :
- Maintenir une vue d'ensemble malgré l'absence de binôme.
- Minimiser les retours en arrière grâce à une structure de code bien planifiée dès le départ.
- Rester efficace en suivant un cadre méthodique adapté au travail en solo.

Exemple concret : avant d'implémenter la croissance des plantes, j'ai formalisé leurs attributs et comportements communs dans une classe abstraite (`Plante`), ce qui a accéléré le développement des espèces spécifiques (tomates, carottes, etc.).

Outils utilisés :
- **GitHub** pour le versioning et le suivi des tâches (issues, milestones).
- **Visual Studio Code** pour coder.

Cette autonomie exigeante a renforcé mes compétences en gestion de projet individuel tout en respectant les bonnes pratiques POO.

Tâche	Temps estimé	Temps réel
Conception des classes	10h	12h

Implémentation des plantes	6h	7h
Système météo	8h	9h
Simulation	5h	3h
test et débogage	7h	9h

3. Présentation du Jeu

3.1. Concept et objectifs

ENSemenC est un simulateur de jardinage éducatif où le joueur doit cultiver des plantes, gérer des ressources et affronter des événements météorologiques. Les objectifs incluent :
- Atteindre l'autosuffisance en ressources.
- Agrandir le jardin en achetant de nouveaux terrains.
- Collectionner toutes les variétés de plantes.

3.2. Fonctionnalités principales

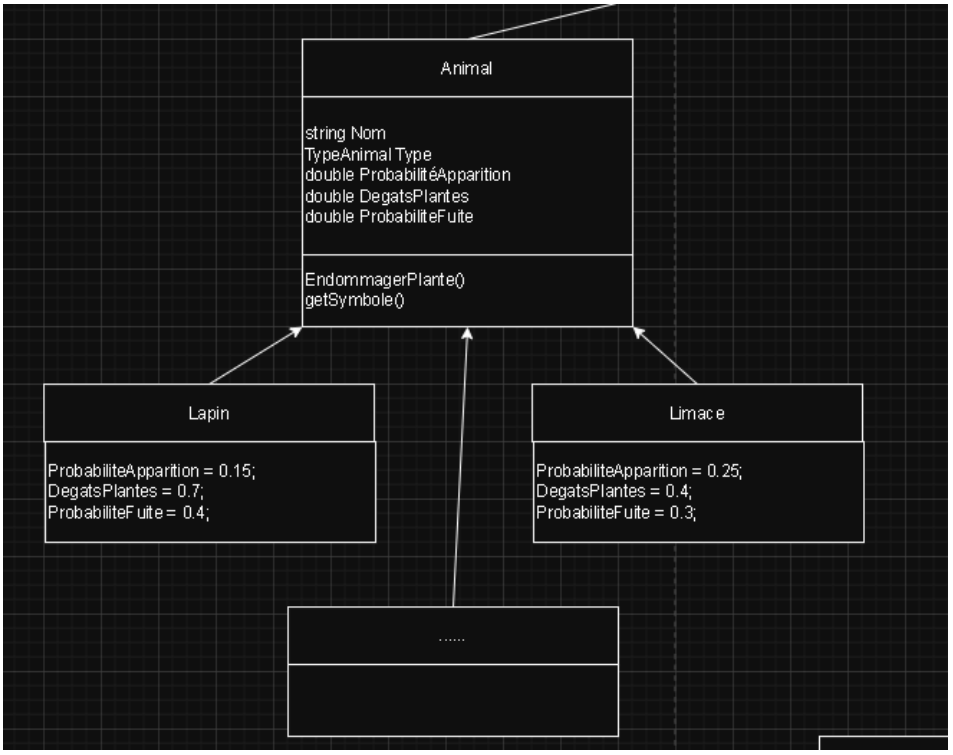
- **10 variétés de plantes** avec des besoins spécifiques (eau, lumière, température).
- **5 espèces animales** interactives (lapins, corbeaux, etc.).
- **Météo dynamique** influencée par la localisation et la saison.
- **Système économique** pour acheter/vendre des semis et récoltes.

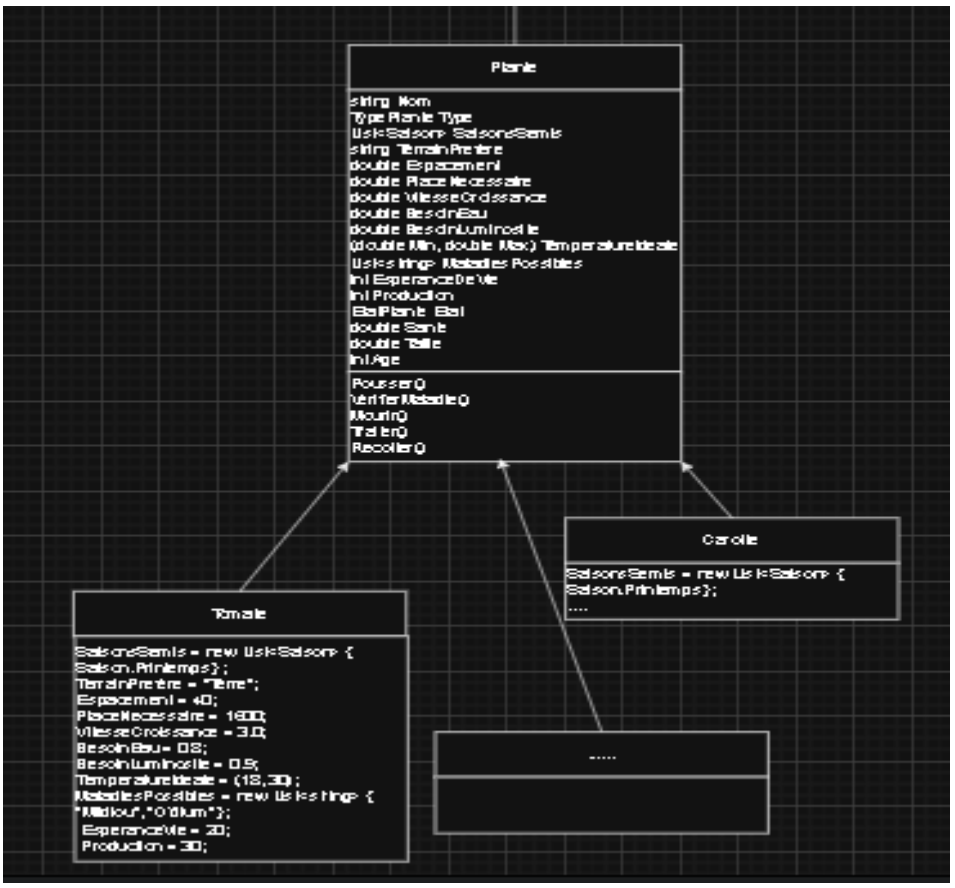
3.3. Déroulement d'une partie

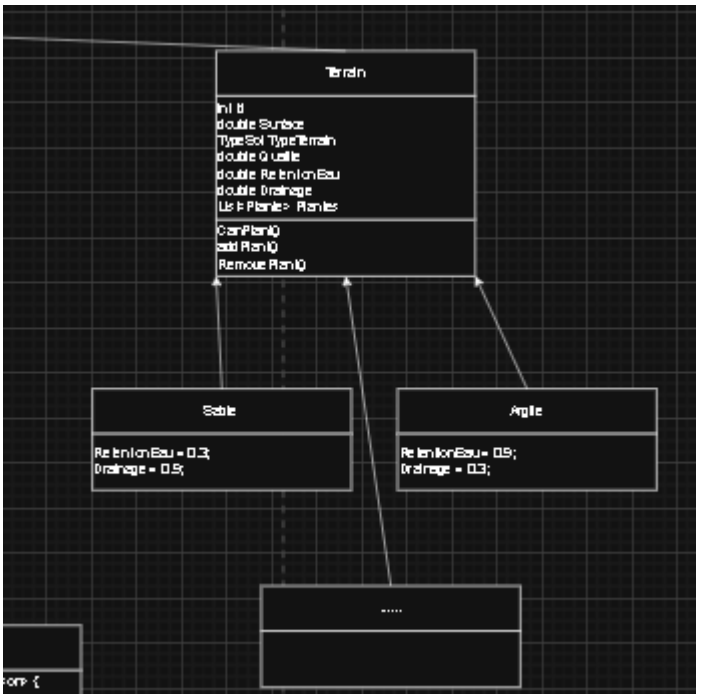
1. **Planification** : Choix des cultures en fonction de la saison.
2. **Installation** : Semis et gestion des terrains.
3. **Entretien** : Traitement des maladies, Effrayer les animaux.
4. **Récolte** : Vente de produits pour agrandir le jardin.

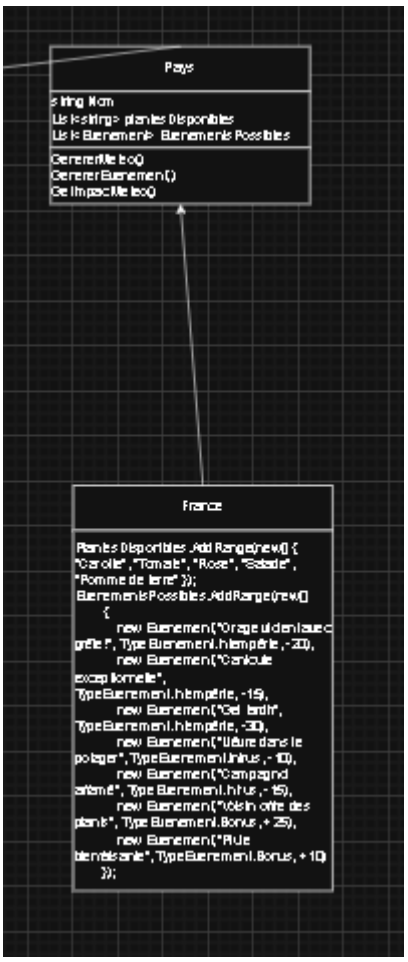
4. Modélisation Objet

4.1. Diagramme de classes UML




Plante
string Nom
TypePlante Type
List<Saison> SaisonsSemis
string TerrainPrefere
double Espacement
double PlaceNecessaire
double VitesseCroissance
double BesoinEau
double BesoinLuminosite
(double Min, double Max) TemperatureIdeale
List<string> MaladiesPossibles
int EsperanceDeVie
int Production
EtatPlante Etat
double Sante
double Taille
int Age
Pousser()
VerifierMaladie()
Mourir()
Tailler()
Recolter()
Carotte
SaisonsSemis = new List<Saison> {
Saison.Printemps};
....
Tomate
SaisonsSemis = new List<Saison> {
Saison.Printemps};
TerrainPrefere = "Terre";
Espacement = 40;
PlaceNecessaire = 1600;
VitesseCroissance = 3.0;
BesoinEau = 0.8;
BesoinLuminosite = 0.9;
TemperatureIdeale = (18,30);
MaladiesPossibles = new List<string> {
"Mildiou","Oïdium"};
EsperanceVie = 20;
Production = 30;
.....


Terrain
int Id
double Surface
TypeSol TypeTerrain
double Qualite
double RetentionEau
double Drainage
List<Plante> Plantes
CanPlant()
addPlant()
Remove Plant()
Sable
RetentionEau = 0.3;
Drainage = 0.9;
Argile
RetentionEau = 0.9;
Drainage = 0.3;
.....



4.2. Explication des classes principales

- **Jardin** : Classe centrale gérant les terrains, les plantes et les animaux.

```
public class Jardin
{
    13 references
    public List<Terrain> Terrains { get; private set; }
    5 references
    public Pays Localisation { get; private set; }
    7 references
    public DateTime DateCourante { get; private set; }
    13 references
    public Dictionary<string, int> InventaireSemis { get; private set; }
    10 references
    public Dictionary<string, int> InventaireRecoltes { get; private set; }
    9 references
    public int Argent { get; set; }
    10 references
    public List<Animal> Animaux { get; private set; }
    2 references
    private List<Animal> AnimauxDisponibles { get; set; }
```

- **Plante** : Classe abstraite avec des implémentations concrètes (Tomate, Carotte, etc.).

```
public abstract class Plante
{
    11 references
    public string Nom { get; protected set; }
    1 reference
    public TypePlante Type { get; protected set; }
    11 references
    public List<Saison> SaisonsSemis { get; protected set; }
    10 references
    public string TerrainPrefere { get; protected set; }
    10 references
    public double Espacement { get; protected set; } // en cm
    13 references
    public double PlaceNecessaire { get; protected set; } // en cm²
    11 references
    public double VitesseCroissance { get; protected set; } // cm par semaine
    11 references
    public double BesoinEau { get; protected set; } // de 0 à 1
    11 references
    public double BesoinLuminosite { get; protected set; } // de 0 à 1
    15 references
    public (double Min, double Max) TemperatureIdeale { get; protected set; } // en °C
    12 references
    public List<string> MaladiesPossibles { get; protected set; }
    11 references
    public int EsperanceVie { get; protected set; } // en semaines
    11 references
    public int Production { get; protected set; } // nombre de fruits/légumes
    21 references
    public EtatPlante Etat { get; protected set; }
    16 references
    public double Sante { get;  set; } // de 0 à 1
    5 references
    public double Taille { get; protected set; } // en cm
    3 references
    public int Age { get; protected set; } // en semaines
```

- **Animal** : Gère les interactions avec les plantes (dégâts, fuite).

```
public enum TypeAnimal
{
    1 reference
    Rongeur,
    1 reference
    Oiseau,
    1 reference
    Insecte,
    0 references
    Reptile,
    1 reference
    Lagomorphe
}

14 references
public abstract class Animal
{
    7 references
    public string Nom { get; protected set; }
    1 reference
    public TypeAnimal Type { get; protected set; }
    5 references
    public double ProbabiliteApparition { get; protected set; } // 0 à 1
    5 references
    public double DegatsPlantes { get; protected set; } // 0 à 1
    5 references
    public double ProbabiliteFuite { get; protected set; } // 0 à 1

    4 references
    protected Animal(string nom, TypeAnimal type)
    {
        Nom = nom;
        Type = type;
    }
```

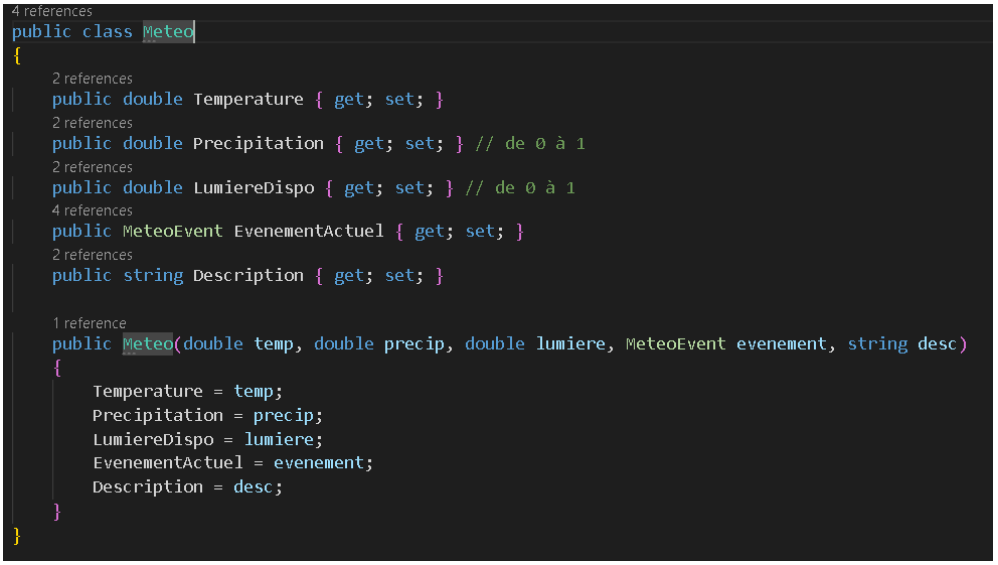
- **Terrain** : Différents types de sols (sable, argile) avec des propriétés spécifiques.

```
public enum TypeSol {Sable, Argile, Terre, Mixte}

14 references
public abstract class Terrain{
    5 references
    public int Id { get; }
    5 references
    public double Surface { get; } // en m²
    3 references
    public TypeSol TypeTerrain {get; protected set;}
    2 references
    public double Qualite { get;  set; } // de 0 à 1
    4 references
    public double RetentionEau { get;  set; } // de 0 à 1
    4 references
    public double Drainage { get;  set; } // de 0 à 1
    13 references
    public List<Plante> Plantes { get; protected set; }

    1 reference
    protected Terrain()
    {
        Plantes = new List<Plante>(); // Initialisation de la liste
        Qualite = 0.7; // valeur par défaut
    }

    4 references
    protected Terrain(int id, double surface, TypeSol type) : this() // Appel au constructeur par défaut
    {
        Id = id;
        Surface = surface;
        TypeTerrain = type;
    }
```

- **Simulation :** Pour gérer la simulation du potager.

```
2 references
public class Simulateur
{
    23 references
    private Jardin Jardin { get; set; }
    1 reference
    private List<Plante> Catalogue { get; set; }
    1 reference
    private Magasin Magasin { get; set; }

    1 reference
    public Simulateur(Jardin jardin, List<Plante> catalogue)
    {
        Jardin = jardin;
        Catalogue = catalogue;
        Magasin = new Magasin();
    }
```

- Magasin : pour gérer l'ensembles des opérations que peut faire le joueur
- Météo : pour gérer la météo



```
4 references
public class Meteo
{
    2 references
    public double Temperature { get; set; }
    2 references
    public double Precipitation { get; set; } // de 0 à 1
    2 references
    public double LumiereDispo { get; set; } // de 0 à 1
    4 references
    public MeteoEvent EvenementActuel { get; set; }
    2 references
    public string Description { get; set; }

    1 reference
    public Meteo(double temp, double precip, double lumiere, MeteoEvent evenement, string desc)
    {
        Temperature = temp;
        Precipitation = precip;
        LumiereDispo = lumiere;
        EvenementActuel = evenement;
        Description = desc;
    }
}
```

4.3. Explications des classes dérivées

- Plantes : pour classer l'ensembles des plantes dérivées de la classe plante.
- animaux : pour classer l'ensemble des animaux dérivés de la classe animal.
- Terrains : pour classer l'ensemble des animaux dérivés de la classe terrain.

4.4. Principes SOLID appliqués

- **Single Responsibility** : Chaque classe a une responsabilité unique.
- **Open/Closed** : Extensible via l'héritage (ex : nouvelles plantes).

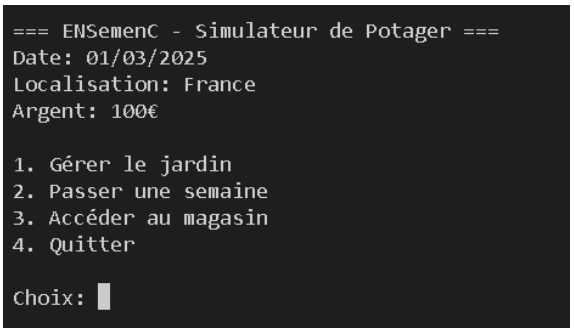
5. Tests et Validation

5.1. Tests unitaires

```
var france = new France();
var jardin = new Jardin(france);




// Initialisation des données
var catalogue = new List<Plante>
{
    new Carotte(),
    new Tomate(),
    new Rose(),
    new Salade(),
    new PommeDeTerre(),
    new Tournesol(),
    new Cactus(),
    new Bambou(),
    new PlanteMagique(),
    new Mandragore()
};

var simulateur = new Simulateur(jardin, catalogue);
simulateur.AfficherMenuPrincipal();

// Création du simulateur
```

5.2. Tests d'intégration

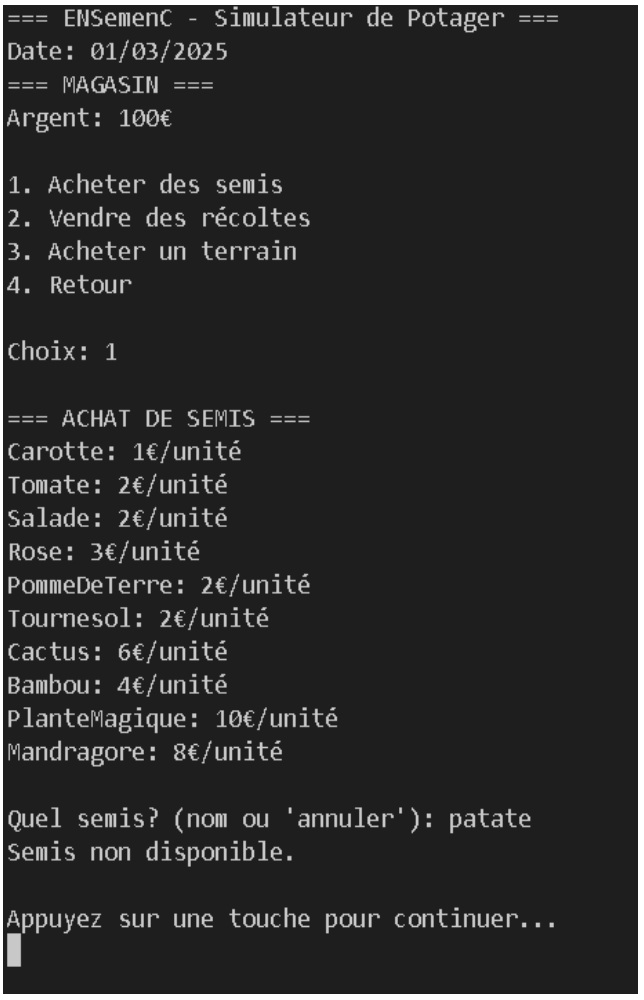
- Simulation d'une saison complète.
- Chaîne économique (semis → récolte → vente).



5.3. Robustesse

- Gestion des entrées d'utilisateur.

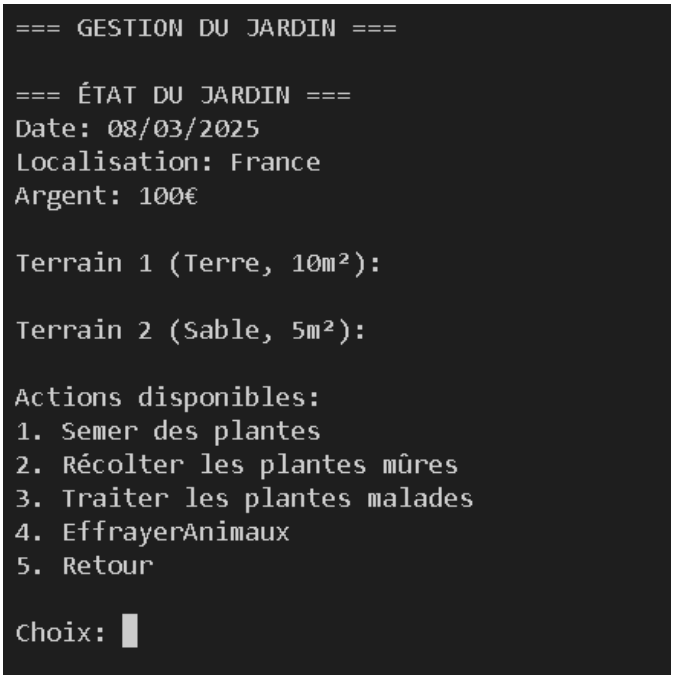
- Messages d'erreur explicites.



6. Interface Utilisateur

6.1. Conception ergonomique

- Affichage clair des ressources et des actions possibles.



7. Améliorations possibles

1. Interface graphique : Remplacer la console par une UI.
2. Sauvegarde cloud : Permettre de reprendre une partie.
3. Multijoueur : Échanges entre joueurs.

9. Conclusion

ENSemenC est un projet complet qui allie programmation orientée objet et modélisation d'un écosystème complexe. Les défis techniques ont été relevés grâce à une organisation rigoureuse et des tests approfondis. Le jeu offre une base solide pour des extensions futures, tout en restant accessible et éducatif.

Bilan : Un simulateur réaliste et extensible, idéal pour apprendre les bases du jardinage et de la POO.

Annexes :

- Code source commenté

Plante.cs:

```
// Plante.cs

public enum TypePlante
{
    Annuelle,
    Vivace,
    MauvaiseHerbe,
    Ornementale,
    Comestible,
    Imaginaire
}

public enum Saison
{
    Printemps,
    Ete,
    Automne,
    Hiver
}

public enum EtatPlante
{
    Graine,
    Pousse,
    Mature,
    Malade,
    Mort,
    Recoltable
}

public abstract class Plante
{
    public string Nom { get; protected set; }
    public TypePlante Type { get; protected set; }
    public List<Saison> SaisonsSemis { get; protected set; }
    public string TerrainPrefere { get; protected set; }
    public double Espacement { get; protected set; } // en cm
    public double PlaceNecessaire { get; protected set; } // en cm²
    public double VitesseCroissance { get; protected set; } // cm par
semaine
```

```
    public double BesoinEau { get; protected set; } // de 0 à 1
    public double BesoinLuminosite { get; protected set; } // de 0 à 1
    public (double Min, double Max) TemperatureIdeale { get; protected
set; } // en °C
    public List<string> MaladiesPossibles { get; protected set; }
    public int EsperanceVie { get; protected set; } // en semaines
    public int Production { get; protected set; } // nombre de
fruits/légumes
    public EtatPlante Etat { get; protected set; }
    public double Sante { get;  set; } // de 0 à 1
    public double Taille { get; protected set; } // en cm
    public int Age { get; protected set; } // en semaines


    protected Plante(string nom, TypePlante type)
    {
        Nom = nom;
        Type = type;
        SaisonsSemis = new List<Saison>();
        MaladiesPossibles = new List<string>();
        Etat = EtatPlante.Graine;
        Sante = 1.0;
        Taille = 0.1; // 1mm pour commencer
        Age = 0;
    }

    public virtual void Pousser(double qualiteTerrain, double
eauDisponible, double luminosite, double temperature)
    {
        if (Etat == EtatPlante.Mort) return;

        Age++;

        // Calcul de la satisfaction des besoins
        double satisfactionEau = 1 - Math.Abs(BesoinEau -
eauDisponible);
        double satisfactionLumiere = 1 - Math.Abs(BesoinLuminosite -
luminosite);
        double satisfactionTemp = temperature >= TemperatureIdeale.Min
&& temperature <= TemperatureIdeale.Max ? 1 :
                                  (temperature < TemperatureIdeale.Min ?
temperature / TemperatureIdeale.Min :
                                  TemperatureIdeale.Max / temperature);
```

```
        double satisfactionMoyenne = (satisfactionEau +
satisfactionLumiere + satisfactionTemp + qualiteTerrain) / 4;

        if (satisfactionMoyenne < 0.5)
        {
            Sante -= 0.2;
            if (Sante <= 0)
            {
                Mourir();
                return;
            }
        }
        else
        {
            Sante = Math.Min(1.0, Sante + 0.05);
        }

        // Croissance en fonction de la satisfaction
        Taille += VitesseCroissance * satisfactionMoyenne;

        // Changement d'état
        if (Etat == EtatPlante.Graine && Taille > 1.0)
            Etat = EtatPlante.Pousse;
        else if (Etat == EtatPlante.Pousse && Taille > TailleMature() *
0.5)
            Etat = EtatPlante.Mature;
        else if (Etat == EtatPlante.Mature && Age > EsperanceVie * 0.7)
            Etat = EtatPlante.Recoltable;

        // Vérifier les maladies
        VerifierMaladie();
    }

    protected virtual double TailleMature()
    {
        return 30.0; // taille moyenne par défaut en cm
    }

    protected virtual void VerifierMaladie()
    {
        if (Etat == EtatPlante.Mort) return;
```

```
        Random rand = new Random();
        foreach (var maladie in MaladiesPossibles)
        {
            if (rand.NextDouble() < 0.05) // 5% de chance par maladie
            {
                Etat = EtatPlante.Malade;
                Sante -= 0.3;
                if (Sante <= 0) Mourir();
                return;
            }
        }
    }

    public virtual void Mourir()
    {
        Etat = EtatPlante.Mort;
        Sante = 0;
    }

    public virtual void Traiter()
    {
        if (Etat == EtatPlante.Malade)
        {
            Etat = EtatPlante.Mature;
            Sante = Math.Min(1.0, Sante + 0.3);
        }
    }

    public virtual int Recolter()
    {
        if (Etat != EtatPlante.Recoltable) return 0;

        int recolte = (int)(Production * Sante);
        Mourir();
        return recolte;
    }


}
```

Plantes.cs:

```
public class Carotte : Plante
```

```
{
    public Carotte() : base("Carotte", TypePlante.Comestible)
    {
        SaisonsSemis = new List<Saison> { Saison.Printemps, Saison.Ete
};
        TerrainPrefere = "Terre";
        Espacement = 5;
        PlaceNecessaire = 25;
        VitesseCroissance = 2.5;
        BesoinEau = 0.7;
        BesoinLuminosite = 0.8;
        TemperatureIdeale = (15, 25);
        MaladiesPossibles = new List<string> { "Mildiou", "Rouille" };
        EsperanceVie = 12;
        Production = 10;
    }

    protected override double TailleMature()
    {
        return 20.0;
    }
}

public class Tomate : Plante
{
    public Tomate() : base("Tomate", TypePlante.Comestible)
    {
        SaisonsSemis = new List<Saison> { Saison.Printemps };
        TerrainPrefere = "Terre";
        Espacement = 40;
        PlaceNecessaire = 1600;
        VitesseCroissance = 3.0;
        BesoinEau = 0.8;
        BesoinLuminosite = 0.9;
        TemperatureIdeale = (18, 30);
        MaladiesPossibles = new List<string> { "Mildiou", "Oïdium" };
        EsperanceVie = 20;
        Production = 30;
    }

    protected override double TailleMature()
    {
        return 100.0; // plantes de tomates plus grandes
```

```
    }
}

public class Rose : Plante
{
    public Rose() : base("Rose", TypePlante.Ornementale)
    {
        SaisonsSemis = new List<Saison> { Saison.Printemps,
Saison.Automne };
        TerrainPrefere = "Argile";
        Espacement = 50;
        PlaceNecessaire = 2500;
        VitesseCroissance = 1.5;
        BesoinEau = 0.6;
        BesoinLuminosite = 0.7;
        TemperatureIdeale = (10, 28);
        MaladiesPossibles = new List<string> { "Oïdium", "Taches
noires" };
        EsperanceVie = 260; // 5 ans
        Production = 15;
    }
}

public class Salade : Plante
{
    public Salade() : base("Salade", TypePlante.Comestible)
    {
        SaisonsSemis = new List<Saison> { Saison.Printemps,
Saison.Automne };
        TerrainPrefere = "Terre";
        Espacement = 30;
        PlaceNecessaire = 900;
        VitesseCroissance = 2.0;
        BesoinEau = 0.8;
        BesoinLuminosite = 0.7;
        TemperatureIdeale = (10, 20);
        MaladiesPossibles = new List<string> { "Mildiou", "Pucerons" };
        EsperanceVie = 8;
        Production = 1; // Une salade par plant
    }
}

public class PommeDeTerre : Plante
```

```
{
    public PommeDeTerre() : base("PommeDeTerre", TypePlante.Comestible)
    {
        SaisonsSemis = new List<Saison> { Saison.Printemps };
        TerrainPrefere = "Terre";
        Espacement = 40;
        PlaceNecessaire = 1600;
        VitesseCroissance = 1.5;
        BesoinEau = 0.6;
        BesoinLuminosite = 0.8;
        TemperatureIdeale = (15, 25);
        MaladiesPossibles = new List<string> { "Doryphore", "Mildiou"
};
        EsperanceVie = 20;
        Production = 10; // Nombre de pommes de terre par plant
    }

    protected override double TailleMature()
    {
        return 60.0;
    }
}

public class Tournesol : Plante
{
    public Tournesol() : base("Tournesol", TypePlante.Ornementale)
    {
        SaisonsSemis = new List<Saison> { Saison.Printemps };
        TerrainPrefere = "Terre";
        Espacement = 50;
        PlaceNecessaire = 2500;
        VitesseCroissance = 3.0;
        BesoinEau = 0.5;
        BesoinLuminosite = 1.0; // Beaucoup de lumière
        TemperatureIdeale = (20, 30);
        MaladiesPossibles = new List<string> { "Oïdium", "Pucerons" };
        EsperanceVie = 16;
        Production = 1; // Une fleur
    }

    protected override double TailleMature()
    {
        return 200.0; // Très grand
```

```
    }

}

public class Cactus : Plante
{
    public Cactus() : base("Cactus", TypePlante.Ornementale)
    {
        SaisonsSemis = new List<Saison> { Saison.Printemps, Saison.Ete
};
        TerrainPrefere = "Sable";
        Espacement = 20;
        PlaceNecessaire = 400;
        VitesseCroissance = 0.5; // Croissance lente
        BesoinEau = 0.2; // Besoin en eau faible
        BesoinLuminosite = 0.9;
        TemperatureIdeale = (25, 40);
        MaladiesPossibles = new List<string> { "Pourriture" };
        EsperanceVie = 520; // 10 ans
        Production = 0; // Non récoltable
    }



    public override int Recolter()
    {
        Console.WriteLine("Les cactus ne sont pas récoltables!");
        return 0;
    }
}

public class Bambou : Plante
{
    public Bambou() : base("Bambou", TypePlante.Comestible) // Ou un
nouveau type Commerciale
    {
        SaisonsSemis = new List<Saison> { Saison.Printemps };
        TerrainPrefere = "Terre";
        Espacement = 100;
        PlaceNecessaire = 10000;
        VitesseCroissance = 5.0; // Croissance très rapide
        BesoinEau = 0.8;
        BesoinLuminosite = 0.7;
```

```
        TemperatureIdeale = (15, 30);
        MaladiesPossibles = new List<string>();
        EsperanceVie = 260; // 5 ans
        Production = 5; // Tiges de bambou
    }

    protected override double TailleMature()
    {
        return 500.0; // Très grand
    }



    public override void Pousser(double qualiteTerrain, double
eauDisponible, double luminosite, double temperature)
    {
        base.Pousser(qualiteTerrain, eauDisponible, luminosite,
temperature);

        // Le bambou peut envahir d'autres terrains
        if (Etat == EtatPlante.Mature && new Random().NextDouble() <
0.05)
        {
            Console.WriteLine("Le bambou s'étend à un terrain
voisin!");
            // Logique pour s'étendre...
        }
    }
}

public class PlanteMagique : Plante
{
    public PlanteMagique() : base("PlanteMagique",
TypePlante.Imaginaire)
    {
        SaisonsSemis = new List<Saison> { Saison.Printemps,
Saison.Automne };
        TerrainPrefere = "Argile";
        Espacement = 40;
        PlaceNecessaire = 1600;
        VitesseCroissance = 1.8;
        BesoinEau = 0.7;
        BesoinLuminosite = 0.5; // Peu de lumière
```

```
        TemperatureIdeale = (10, 25);
        MaladiesPossibles = new List<string> { "Malédiction", "Sort
raté" };
        EsperanceVie = 52; // 1 an
        Production = 3; // Potions magiques
    }




}

public class Mandragore : Plante
{
    public Mandragore() : base("Mandragore", TypePlante.Imaginaire)
    {
        SaisonsSemis = new List<Saison> { Saison.Automne };
        TerrainPrefere = "Argile";
        Espacement = 60;
        PlaceNecessaire = 3600;
        VitesseCroissance = 0.3; // Croissance très lente
        BesoinEau = 0.6;
        BesoinLuminosite = 0.3; // Ombre
        TemperatureIdeale = (5, 15);
        MaladiesPossibles = new List<string> { "Malédiction" };
        EsperanceVie = 104; // 2 ans
        Production = 1; // Très précieuse
    }

    protected override double TailleMature()
    {
        return 30.0;
    }



    public override int Recolter()
    {
        if (Etat != EtatPlante.Recoltable) return 0;

        // La mandragore crie quand on la récolte
        Console.WriteLine("AAAAAAAAAH! (La mandragore pousse un cri
perçant)");
```

```
        return base.Recolter();
    }
}
```

Terrain.cs:

```
public enum TypeSol {Sable, Argile, Terre, Mixte}

public abstract class Terrain{
    public int Id { get; }
    public double Surface { get; } // en m²
    public TypeSol TypeTerrain {get; protected set;}
    public double Qualite { get;  set; } // de 0 à 1
    public double RetentionEau { get;  set; } // de 0 à 1
    public double Drainage { get;  set; } // de 0 à 1
    public List<Plante> Plantes { get; protected set; }

    protected Terrain()
    {
        Plantes = new List<Plante>(); // Initialisation de la liste
        Qualite = 0.7; // valeur par défaut
    }

    protected Terrain(int id, double surface, TypeSol type) : this() //
Appel au constructeur par défaut
    {
        Id = id;
        Surface = surface;
        TypeTerrain = type;
    }

    public virtual bool CanPlant(Plante plante)
    {
        if (plante == null ) return false;
        double espaceOccupe = Plantes?.Sum(p => p?.PlaceNecessaire) ??
0; // Gestion du cas null
        return (Surface * 10000 - espaceOccupe) >=
plante.PlaceNecessaire; // Conversion m² en cm²
    }

    public virtual void AddPlant(Plante plante)
    {
        if (plante == null)
```

```
        {
            Console.WriteLine(" Erreur : plante null");
        }
        else if (CanPlant(plante))
        {
            Plantes.Add(plante);
        }
        else
        {
            Console.WriteLine("Pas assez d'espace pour cette plante");
        }
    }
    public virtual void RemovePlant(Plante plante)
    {
        Plantes?.Remove(plante);
    }
}
```

Terrains.cs:

```
public class TerrainSable : Terrain
{
    public TerrainSable(int id, double surface) : base(id, surface,
TypeSol.Sable)
    {
        RetentionEau = 0.3;
        Drainage = 0.9;
    }
}

public class TerrainArgile : Terrain
{
    public TerrainArgile(int id, double surface) : base(id, surface,
TypeSol.Argile)
    {
        RetentionEau = 0.9;
        Drainage = 0.3;
    }
}
```

```
public class TerrainTerre : Terrain
{
    public TerrainTerre(int id, double surface) : base(id, surface,
TypeSol.Terre)
    {
        RetentionEau = 0.7;
        Drainage = 0.6;
    }
}

public class TerrainMixte : Terrain
{
    public TerrainMixte(int id, double surface) : base(id, surface,
TypeSol.Mixte)
    {
        RetentionEau = 0.8;
        Drainage = 0.5;
    }
}
```

Animal.cs:

```
public enum TypeAnimal
{
    Rongeur,
    Oiseau,
    Insecte,
    Reptile,
    Lagomorphe
}

public abstract class Animal
{
    public string Nom { get; protected set; }
    public TypeAnimal Type { get; protected set; }
    public double ProbabiliteApparition { get; protected set; } // 0 à
1
    public double DegatsPlantes { get; protected set; } // 0 à 1
    public double ProbabiliteFuite { get; protected set; } // 0 à 1

    protected Animal(string nom, TypeAnimal type)
    {
        Nom = nom;
```

```
        Type = type;
    }

    public virtual void EndommagerPlante(Plante plante)
    {
        if (new Random().NextDouble() < DegatsPlantes)
        {
            double degats = 0.1 + (new Random().NextDouble() * 0.4); //
Entre 10% et 50% de dégâts
            plante.Sante -= degats;
            Console.WriteLine($"{Nom} a endommagé {plante.Nom}! Santé
-{(degats*100):F0}%");

            if (plante.Sante <= 0)
            {
                plante.Mourir();
                Console.WriteLine($"{plante.Nom} a été détruite par
{Nom}!");
            }
        }
        else
        {
            Console.WriteLine($"{Nom} a tenté d'attaquer {plante.Nom}
mais a échoué!");
        }
    }

    public abstract string GetSymbole();
}
```

Animaux.cs:

```
public class Lapin : Animal
{
    public Lapin() : base("Lapin", TypeAnimal.Lagomorphe)
    {
        ProbabiliteApparition = 0.15;
        DegatsPlantes = 0.7;
        ProbabiliteFuite = 0.4;
    }

    public override string GetSymbole() => "🐇";
}
```

```
public class Souris : Animal
{
    public Souris() : base("Souris", TypeAnimal.Rongeur)
    {
        ProbabiliteApparition = 0.2;
        DegatsPlantes = 0.5;
        ProbabiliteFuite = 0.6;
    }

    public override string GetSymbole() => "🐁";
}

public class Corbeau : Animal
{
    public Corbeau() : base("Corbeau", TypeAnimal.Oiseau)
    {
        ProbabiliteApparition = 0.1;
        DegatsPlantes = 0.3;
        ProbabiliteFuite = 0.8;
    }

    public override string GetSymbole() => "🐦";
}

public class Limace : Animal
{
    public Limace() : base("Limace", TypeAnimal.Insecte)
    {
        ProbabiliteApparition = 0.25;
        DegatsPlantes = 0.4;
        ProbabiliteFuite = 0.3;
    }

    public override string GetSymbole() => "🐌";
}
```

Jardin.cs:

```
public class Jardin
{
    public List<Terrain> Terrains { get; private set; }
    public Pays Localisation { get; private set; }
    public DateTime DateCourante { get; private set; }
```

```
    public Dictionary<string, int> InventaireSemis { get; private set;
}
    public Dictionary<string, int> InventaireRecoltes { get; private
set; }
    public int Argent { get; set; }
    public List<Animal> Animaux { get; private set; }
    private List<Animal> AnimauxDisponibles { get; set; }

    public Jardin(Pays pays)
    {
        Localisation = pays;
        Terrains = new List<Terrain>();
        InventaireSemis = new Dictionary<string, int>();
        InventaireRecoltes = new Dictionary<string, int>();
        DateCourante = new DateTime(2025, 3, 1); // Début printemps
        Argent = 100; // Argent de départ

        // Initialiser avec quelques terrains de base
        Terrains.Add(new TerrainTerre(1, 10)); // 10m²
        Terrains.Add(new TerrainSable(2, 5));  // 5m²

        // Initialiser avec quelques semis de base
        InventaireSemis.Add("Carotte", 10);
        InventaireSemis.Add("Tomate", 5);
        InventaireSemis.Add("Salade", 8);
        Animaux = new List<Animal>();
        InitialiserAnimaux();
    }

    private void InitialiserAnimaux()
    {
        AnimauxDisponibles = new List<Animal>
        {
            new Lapin(),
            new Souris(),
            new Corbeau(),
            new Limace()
        };
    }

    public void PasserSemaine()
    {
        DateCourante = DateCourante.AddDays(7);
```

```
        var meteo = Localisation.GenererMeteo(DateCourante);

        Console.WriteLine($"\n=== Semaine du {DateCourante:dd/MM/yyyy}
===");
        Console.WriteLine($"Météo: {meteo.Description}");

        foreach (var terrain in Terrains)
        {
            foreach (var plante in terrain.Plantes.ToList())
            {
                plante.Pousser(terrain.Qualite, meteo.Precipitation,
meteo.LumiereDispo, meteo.Temperature);

                // Appliquer l'impact des événements météo
                if (meteo.EvenementActuel != MeteoEvent.None)
                {
                    double impact =
Localisation.GetImpactMeteo(meteo.EvenementActuel);
                    plante.Sante += impact;

                    if (plante.Sante <= 0)
                    {
                        plante.Mourir();
                        terrain.RemovePlant(plante);
                        Console.WriteLine($"{plante.Nom} a été détruite
par {meteo.EvenementActuel}!");
                    }
                }
            }
        }
    }

    private void GererAnimaux()
    {
        // Faire fuir certains animaux existants
        foreach (var animal in Animaux.ToList())
        {
            if (new Random().NextDouble() < animal.ProbabiliteFuite)
            {
                Animaux.Remove(animal);
                Console.WriteLine($"{animal.Nom} a quitté le jardin.");
            }
        }
```

```
        // Ajouter de nouveaux animaux aléatoirement
        foreach (var animalType in AnimauxDisponibles)
        {
            if (new Random().NextDouble() <
animalType.ProbabiliteApparition)
            {

Animaux.Add(Activator.CreateInstance(animalType.GetType()) as Animal);
                Console.WriteLine($"Un {animalType.Nom} est apparu dans
le jardin! {animalType.GetSymbole()}");
            }
        }

        // Les animaux endommagent les plantes
        foreach (var animal in Animaux)
        {
            foreach (var terrain in Terrains)
            {
                if (terrain.Plantes.Any() && new Random().NextDouble()
> 0.5) // 50% de chance d'attaquer ce terrain
                {
                    var planteCible = terrain.Plantes[new
Random().Next(terrain.Plantes.Count)];
                    animal.EndommagerPlante(planteCible);
                }
            }
        }
    }

    public void EffrayerAnimaux()
    {
        if (!Animaux.Any())
        {
            Console.WriteLine("Il n'y a aucun animal à effrayer.");
            return;
        }

        int animauxEffrayes = 0;
        foreach (var animal in Animaux.ToList())
        {
            if (new Random().NextDouble() < 0.7) // 70% de chance de
fuite
```

```
            {
                Animaux.Remove(animal);
                animauxEffrayes++;
            }
        }

        Console.WriteLine($"Vous avez effrayé {animauxEffrayes}
animal(s)!");
    }

    public void AfficherEtat()
    {
        Console.WriteLine("\n=== ÉTAT DU JARDIN ===");
        Console.WriteLine($"Date: {DateCourante:dd/MM/yyyy}");
        Console.WriteLine($"Localisation: {Localisation.Nom}");
        Console.WriteLine($"Argent: {Argent}€");

        foreach (var terrain in Terrains)
        {
            Console.WriteLine($"\nTerrain {terrain.Id}
({terrain.TypeTerrain}, {terrain.Surface}m²):");
            foreach (var plante in terrain.Plantes)
            {
                Console.WriteLine($"- {plante.Nom}: {plante.Etat},
Santé: {plante.Sante:P0}, Taille: {plante.Taille:F1}cm");
            }
        }
    }
}
```

Magasin.cs:

```
public class Magasin
{
    public static void AfficherMenu(Jardin jardin)
    {
        bool continuer = true;
        while (continuer)
        {
            Console.Clear();
            Console.WriteLine("=== MAGASIN ===");
            Console.WriteLine($"Argent: {jardin.Argent}€");
            Console.WriteLine("\n1. Acheter des semis");
            Console.WriteLine("2. Vendre des récoltes");
```

```
            Console.WriteLine("3. Acheter un terrain");
            Console.WriteLine("4. Retour");

            Console.Write("\nChoix: ");
            var choix = Console.ReadLine();

            switch (choix)
            {
                case "1":
                    AcheterSemis(jardin);
                    break;
                case "2":
                    VendreRecoltes(jardin);
                    break;
                case "3":
                    AcheterTerrain(jardin);
                    break;
                case "4":
                    continuer = false;
                    break;
                default:
                    Console.WriteLine("Choix invalide.");
                    break;
            }

            if (continuer)
            {
                Console.WriteLine("\nAppuyez sur une touche pour
continuer...");
                Console.ReadKey();
            }
        }
    }

    private static void AcheterSemis(Jardin jardin)
    {
        var prixSemis = new Dictionary<string, int>
        {
            {"Carotte", 1},
            {"Tomate", 2},
            {"Salade", 2},
            {"Rose", 3},
            {"PommeDeTerre", 2},
```

```
            {"Tournesol", 2},
            {"Cactus", 6},
            {"Bambou", 4},
            {"PlanteMagique", 10},
            {"Mandragore", 8}
        };

        Console.WriteLine("\n=== ACHAT DE SEMIS ===");
        foreach (var semis in prixSemis)
        {
            Console.WriteLine($"{semis.Key}: {semis.Value}€/unité");
        }

        Console.Write("\nQuel semis? (nom ou 'annuler'): ");
        string choix = Console.ReadLine();

        if (choix.ToLower() == "annuler") return;

        if (!prixSemis.ContainsKey(choix))
        {
            Console.WriteLine("Semis non disponible.");
            return;
        }

        Console.Write("Quantité? ");
        if (!int.TryParse(Console.ReadLine(), out int quantite) ||
quantite <= 0)
        {
            Console.WriteLine("Quantité invalide.");
            return;
        }

        int cout = prixSemis[choix] * quantite;
        if (cout > jardin.Argent)
        {
            Console.WriteLine("Fonds insuffisants.");
            return;
        }

        if (jardin.InventaireSemis.ContainsKey(choix))
            jardin.InventaireSemis[choix] += quantite;
        else
            jardin.InventaireSemis[choix] = quantite;
```

```
        jardin.Argent -= cout;
        Console.WriteLine($"Achat réussi! Vous avez maintenant
{jardin.InventaireSemis[choix]} graines de {choix}.");
    }

    private static void VendreRecoltes(Jardin jardin)
    {
        var prixVente = new Dictionary<string, int>
        {
            {"Carotte", 2},
            {"Tomate", 3},
            {"Salade", 2},
            {"Rose", 5},
            {"PommeDeTerre", 2},
            {"Tournesol", 2},
            {"Cactus", 6},
            {"Bambou", 4},
            {"PlanteMagique", 10},
            {"Mandragore", 8}
        };

        if (jardin.InventaireRecoltes.All(r => r.Value == 0))
        {
            Console.WriteLine("\nVous n'avez aucune récolte à
vendre!");
            return;
        }

        Console.WriteLine("\n=== VENTE DE RÉCOLTES ===");
        foreach (var recolte in jardin.InventaireRecoltes)
        {
            if (recolte.Value > 0)
            {
                int prix = prixVente.GetValueOrDefault(recolte.Key, 1);
                Console.WriteLine($"{recolte.Key}: {recolte.Value}
unités ({prix}€/unité)");
            }
        }

        Console.Write("\nQuelle récolte? (nom ou 'annuler'): ");
        string choix = Console.ReadLine();
```

```
        if (choix.ToLower() == "annuler") return;

        if (!jardin.InventaireRecoltes.ContainsKey(choix) ||
jardin.InventaireRecoltes[choix] <= 0)
        {
            Console.WriteLine("Récolte non disponible.");
            return;
        }

        Console.Write("Quantité? ");
        if (!int.TryParse(Console.ReadLine(), out int quantite) ||
quantite <= 0)
        {
            Console.WriteLine("Quantité invalide.");
            return;
        }

        if (quantite > jardin.InventaireRecoltes[choix])
        {
            Console.WriteLine("Vous n'avez pas assez de cette
récolte.");
            return;
        }

        int prixUnitaire = prixVente.GetValueOrDefault(choix, 1);
        int gain = prixUnitaire * quantite;

        jardin.InventaireRecoltes[choix] -= quantite;
        jardin.Argent += gain;

        Console.WriteLine($"Vente réussie! Gain: {gain}€");
    }

    private static void AcheterTerrain(Jardin jardin)
    {
        var prixTerrains = new Dictionary<string, int>
        {
            {"Sable", 50},   // par m²
            {"Terre", 70},
            {"Argile", 60},
            {"Mixte", 65}
        };
```

```
        Console.WriteLine("\n=== ACHAT DE TERRAIN ===");
        foreach (var terrain in prixTerrains)
        {
            Console.WriteLine($"{terrain.Key}: {terrain.Value}€/m²");
        }

        Console.Write("\nType de terrain? (nom ou 'annuler'): ");
        string type = Console.ReadLine();

        if (type.ToLower() == "annuler") return;

        if (!prixTerrains.ContainsKey(type))
        {
            Console.WriteLine("Type non disponible.");
            return;
        }

        Console.Write("Surface (m²)? ");
        if (!double.TryParse(Console.ReadLine(), out double surface) ||
surface <= 0)
        {
            Console.WriteLine("Surface invalide.");
            return;
        }

        int cout = (int)(prixTerrains[type] * surface);
        if (cout > jardin.Argent)
        {
            Console.WriteLine("Fonds insuffisants.");
            return;
        }

        int nouveauId = jardin.Terrains.Max(t => t.Id) + 1;
        Terrain nouveauTerrain = type switch
        {
            "Sable" => new TerrainSable(nouveauId, surface),
            "Terre" => new TerrainTerre(nouveauId, surface),
            "Argile" => new TerrainArgile(nouveauId, surface),
            "Mixte" => new TerrainMixte(nouveauId, surface),

        };

        if (nouveauTerrain != null)
```

```
        {
            jardin.Terrains.Add(nouveauTerrain);
            jardin.Argent -= cout;
            Console.WriteLine($"Terrain {type} de {surface}m² acheté
pour {cout}€!");
        }
    }
}
```

Meteo.cs:

```
public enum MeteoEvent
{
    Pluie,
    Gel,
    Averse,
    VagueChaleur,
    Secheresse,
    Orage,
    Grele,
    VentViolent,
    Brouillard,
    None
}

public class Meteo
{
    public double Temperature { get; set; }
    public double Precipitation { get; set; } // de 0 à 1
    public double LumiereDispo { get; set; } // de 0 à 1
    public MeteoEvent EvenementActuel { get; set; }
    public string Description { get; set; }

    public Meteo(double temp, double precip, double lumiere, MeteoEvent
evenement, string desc)
    {
        Temperature = temp;
        Precipitation = precip;
        LumiereDispo = lumiere;
        EvenementActuel = evenement;
        Description = desc;
    }
}
```

Pays.cs:

```
public enum TypeEvenement
{
    Intempérie,
    Intrus,
    Bonus
}

public class Evenement
{
    public string Description { get; }
    public TypeEvenement Type { get; }
    public int Impact { get; } // positif ou négatif

    public Evenement(string description, TypeEvenement type, int
impact)
    {
        Description = description;
        Type = type;
        Impact = impact;
    }
}



public abstract class Pays
{
    public string Nom { get; }
    public List<string> PlantesDisponibles { get; }
    public List<Evenement> EvenementsPossibles { get; }

    protected Pays(string nom)
    {
        Nom = nom;
        PlantesDisponibles = new List<string>();
        EvenementsPossibles = new List<Evenement>();
    }

    public abstract Meteo GenererMeteo(DateTime date);
    public abstract Evenement GenererEvenement();

    internal double GetImpactMeteo(MeteoEvent evenementActuel)
```

```
    {
        throw new NotImplementedException();
    }
}

public class France : Pays
{
    public France() : base("France")
    {
        PlantesDisponibles.AddRange(new[] { "Carotte", "Tomate",
"Rose", "Salade", "Pomme de terre" });

        EvenementsPossibles.AddRange(new[]
        {
            new Evenement("Orage violent avec grêle!",
TypeEvenement.Intempérie, -20),
            new Evenement("Canicule exceptionnelle",
TypeEvenement.Intempérie, -15),
            new Evenement("Gel tardif", TypeEvenement.Intempérie, -30),
            new Evenement("Lièvre dans le potager",
TypeEvenement.Intrus, -10),
            new Evenement("Campagnol affamé", TypeEvenement.Intrus,
-15),
            new Evenement("Voisin offre des plants",
TypeEvenement.Bonus, +25),
            new Evenement("Pluie bienfaisante", TypeEvenement.Bonus,
+10)
        });
    }


    public override Meteo GenererMeteo(DateTime date)
    {
        Random rand = new Random();
        MeteoEvent evenement = MeteoEvent.None;

        // Variation saisonnière
        double baseTemp;
        double basePrecip;

        if (date.Month >= 3 && date.Month <= 5) // Printemps
        {
            baseTemp = 10 + rand.NextDouble() * 15; // 10-25°C
```

```
            basePrecip = 0.4 + rand.NextDouble() * 0.4; // 40-80%
        }
        else if (date.Month >= 6 && date.Month <= 8) // Été
        {
            baseTemp = 18 + rand.NextDouble() * 20; // 18-38°C
            basePrecip = 0.1 + rand.NextDouble() * 0.3; // 10-40%
        }
        else if (date.Month >= 9 && date.Month <= 11) // Automne
        {
            baseTemp = 5 + rand.NextDouble() * 15; // 5-20°C
            basePrecip = 0.3 + rand.NextDouble() * 0.5; // 30-80%
        }
        else // Hiver
        {
            baseTemp = -5 + rand.NextDouble() * 10; // -5 à +5°C
            basePrecip = 0.2 + rand.NextDouble() * 0.3; // 20-50%
        }

        // Ajouter une variation aléatoire
        double temp = baseTemp + (rand.NextDouble() * 6 - 3); // ±3°C
        double precip = Math.Max(0, Math.Min(1, basePrecip +
(rand.NextDouble() * 0.4 - 0.2))); // ±20%
        double lumi = 0.7 + rand.NextDouble() * 0.3; // 70-100%

        string desc = temp switch
        {
            > 30 => "Canicule",
            > 25 => "Chaud et ensoleillé",
            > 20 => "Agréable",
            > 15 => "Doux",
            > 10 => "Fraîche",
            > 5 => "Froid",
            > 0 => "Très froid",
            _ => "Gelée"
        };

        desc += precip switch
        {
            > 0.8 => " avec pluies torrentielles",
            > 0.6 => " avec pluie",
            > 0.4 => " avec averses",
            > 0.2 => " légèrement pluvieux",
            _ => " sec"
```

```
        };

        return new Meteo(temp, precip, lumi, evenement, desc);
    }

    public virtual double GetImpactMeteo(MeteoEvent evenement)
    {
        return evenement switch
        {
            MeteoEvent.Gel => -0.3,
            MeteoEvent.VagueChaleur => -0.2,
            MeteoEvent.Secheresse => -0.4,
            MeteoEvent.Orage => -0.1,
            MeteoEvent.Grele => -0.5,
            MeteoEvent.Pluie => 0.1,
            MeteoEvent.Averse => 0.2,
            _ => 0
        };
    }

    public override Evenement GenererEvenement()
    {
        Random rand = new Random();
        return
EvenementsPossibles[rand.Next(EvenementsPossibles.Count)];
    }
}
```

Recolte.cs:

```
public class Recolte
{
    public string NomPlante { get; }
    public string TypeProduit { get; }
    public int Quantite { get; }
    public double Qualite { get; }
    public Saison SaisonRecolte { get; }
```

```
    public Recolte(string nom, string type, int quantite, double
qualite, Saison saison)
    {
        NomPlante = nom;
        TypeProduit = type;
        Quantite = quantite;
        Qualite = qualite;
        SaisonRecolte = saison;
    }

    public double GetVenteBase()
    {
        return TypeProduit switch {
            "Fruit" => 3.5,
            "Légume" => 2.5,
            "Fleur" => 4.0,
            "Aromatique" => 5.0,
            "Médicinale" => 6.0,
            _ => 1.5
        } * (1 + Qualite);
    }
}
```

Simulation.cs:

```
public class Simulateur
{
    private Jardin Jardin { get; set; }
    private List<Plante> Catalogue { get; set; }
    private Magasin Magasin { get; set; }

    public Simulateur(Jardin jardin, List<Plante> catalogue)
    {
        Jardin = jardin;
        Catalogue = catalogue;
        Magasin = new Magasin();
    }

    public void AfficherMenuPrincipal()
    {
        while (true)
        {
```

```
            Console.Clear();
            Console.WriteLine("=== ENSemenC - Simulateur de Potager
===");
            Console.WriteLine($"Date:
{Jardin.DateCourante:dd/MM/yyyy}");
            Console.WriteLine($"Localisation:
{Jardin.Localisation.Nom}");
            Console.WriteLine($"Argent: {Jardin.Argent}€");
            Console.WriteLine("\n1. Gérer le jardin");
            Console.WriteLine("2. Passer une semaine");
            Console.WriteLine("3. Accéder au magasin");
            Console.WriteLine("4. Quitter");

            Console.Write("\nChoix: ");
            var choix = Console.ReadLine();

            switch (choix)
            {
                case "1":
                    GererJardin();
                    break;
                case "2":
                    PasserSemaine();
                    break;
                case "3":
                    Magasin.AfficherMenu(Jardin);
                    break;
                case "4":
                    return;
                default:
                    Console.WriteLine("Choix invalide.");
                    break;
            }
        }
    }

    private void GererJardin()
    {
        while (true)
        {
            Console.Clear();
            Console.WriteLine("=== GESTION DU JARDIN ===");
            Jardin.AfficherEtat();
```

```
            if (Jardin.Animaux.Any())
            {
                Console.WriteLine("\nAnimaux présents:");
                foreach (var animal in Jardin.Animaux)
                {
                    Console.WriteLine($"- {animal.Nom}
{animal.GetSymbole()}");
                }
            }

            Console.WriteLine("\nActions disponibles:");
            Console.WriteLine("1. Semer des plantes");
            Console.WriteLine("2. Récolter les plantes mûres");
            Console.WriteLine("3. Traiter les plantes malades");
            Console.WriteLine("4. EffrayerAnimaux");
            Console.WriteLine("5. Retour");

            Console.Write("\nChoix: ");
            var choix = Console.ReadLine();

            switch (choix)
            {
                case "1":
                    SemerPlantes();
                    break;
                case "2":
                    RecolterPlantes();
                    break;
                case "3":
                    TraiterPlantesMalades();
                    break;
                case "4":
                    EffrayerAnimaux();
                    break;
                case "5":
                    return;
                default:
                    Console.WriteLine("Choix invalide.");
                    break;
            }
```

```
            Console.WriteLine("\nAppuyez sur une touche pour
continuer...");
            Console.ReadKey();
        }
    }

    private void EffrayerAnimaux()
    {
        Jardin.EffrayerAnimaux();
        Console.WriteLine("\nAppuyez sur une touche pour
continuer...");
        Console.ReadKey();
    }

    private void SemerPlantes()
    {
        Console.WriteLine("\n=== SEMER DES PLANTES ===");

        // Afficher les semis disponibles
        Console.WriteLine("\nSemis disponibles:");
        foreach (var semis in Jardin.InventaireSemis)
        {
            if (semis.Value > 0)
                Console.WriteLine($"- {semis.Key}: {semis.Value}
graines");
        }

        if (Jardin.InventaireSemis.All(s => s.Value <= 0))
        {
            Console.WriteLine("\nVous n'avez plus de semis
disponibles!");
            Console.WriteLine("Visitez le magasin pour en acheter.");
            return;
        }

        Console.Write("\nQuelle plante voulez-vous semer? (nom ou
'annuler'): ");
        string choix = Console.ReadLine();

        if (choix.ToLower() == "annuler") return;

        if (!Jardin.InventaireSemis.ContainsKey(choix) ||
Jardin.InventaireSemis[choix] <= 0)
```

```
        {
            Console.WriteLine("Semis non disponible.");
            return;
        }

        // Choisir le terrain
        Console.WriteLine("\nTerrains disponibles:");
        foreach (var terrain1 in Jardin.Terrains)
        {
            double espaceOccupe = terrain1.Plantes.Sum(p =>
p.PlaceNecessaire);
            double espaceDispo = terrain1.Surface * 10000 -
espaceOccupe; // Conversion m² en cm²
            Console.WriteLine($"- Terrain {terrain1.Id}:
{terrain1.Surface}m² ({terrain1.TypeTerrain}), " +
                              $"Espace libre:
{espaceDispo/10000:F2}m²");
        }

        Console.Write("\nSur quel terrain? (numéro): ");
        if (!int.TryParse(Console.ReadLine(), out int terrainId))
        {
            Console.WriteLine("Numéro invalide.");
            return;
        }

        var terrain = Jardin.Terrains.FirstOrDefault(t => t.Id ==
terrainId);
        if (terrain == null)
        {
            Console.WriteLine("Terrain introuvable.");
            return;
        }

        // Créer la plante
        Plante nouvellePlante = CreerNouvellePlante(choix);
        if (nouvellePlante == null)
        {
            Console.WriteLine("Type de plante inconnu.");
            return;
        }

        if (terrain.CanPlant(nouvellePlante))
```

```
        {
            terrain.AddPlant(nouvellePlante);
            Jardin.InventaireSemis[choix]--;
            Console.WriteLine($"{choix} semée avec succès sur le
terrain {terrainId}!");
        }
        else
        {
            Console.WriteLine("Pas assez d'espace pour cette plante.");
        }
    }

    private Plante CreerNouvellePlante(string typePlante)
    {
        return typePlante switch
        {
            "Carotte" => new Carotte(),
            "Tomate" => new Tomate(),
            "Rose" => new Rose(),
            "Salade" => new Salade(),
            "PommeDeTerre" => new PommeDeTerre(),
            "Tournesol" => new Tournesol(),
            "Cactus" => new Cactus(),
            "Bambou" => new Bambou(),
            "PlanteMagique" => new PlanteMagique(),
            "Mandragore" => new Mandragore(),

        };
    }

    private void PasserSemaine()
    {
        Jardin.PasserSemaine();

        // Vérifier si des plantes sont prêtes à être récoltées
        bool plantesRecoltables = Jardin.Terrains
            .Any(t => t.Plantes.Any(p => p.Etat ==
EtatPlante.Recoltable));

        if (plantesRecoltables)
        {
            Console.WriteLine("\nATTENTION: Certaines plantes sont
prêtes à être récoltées!");
```

```
        }
    }

    private void RecolterPlantes()
    {
        int totalRecolte = 0;

        foreach (var terrain in Jardin.Terrains)
        {
            foreach (var plante in terrain.Plantes.ToList())
            {
                if (plante.Etat == EtatPlante.Recoltable)
                {
                    int quantite = plante.Recolter();
                    totalRecolte += quantite;

                    if
(Jardin.InventaireRecoltes.ContainsKey(plante.Nom))
                        Jardin.InventaireRecoltes[plante.Nom] +=
quantite;
                    else
                        Jardin.InventaireRecoltes[plante.Nom] =
quantite;

                    if (plante.Etat == EtatPlante.Mort)
                        terrain.RemovePlant(plante);

                    Console.WriteLine($"- {plante.Nom}: {quantite}
unités récoltées");
                }
            }
        }

        if (totalRecolte > 0)
        {
            Console.WriteLine($"\nTotal récolté: {totalRecolte}
unités");
        }
        else
        {
            Console.WriteLine("\nAucune plante n'est prête à être
récoltée.");
        }
```

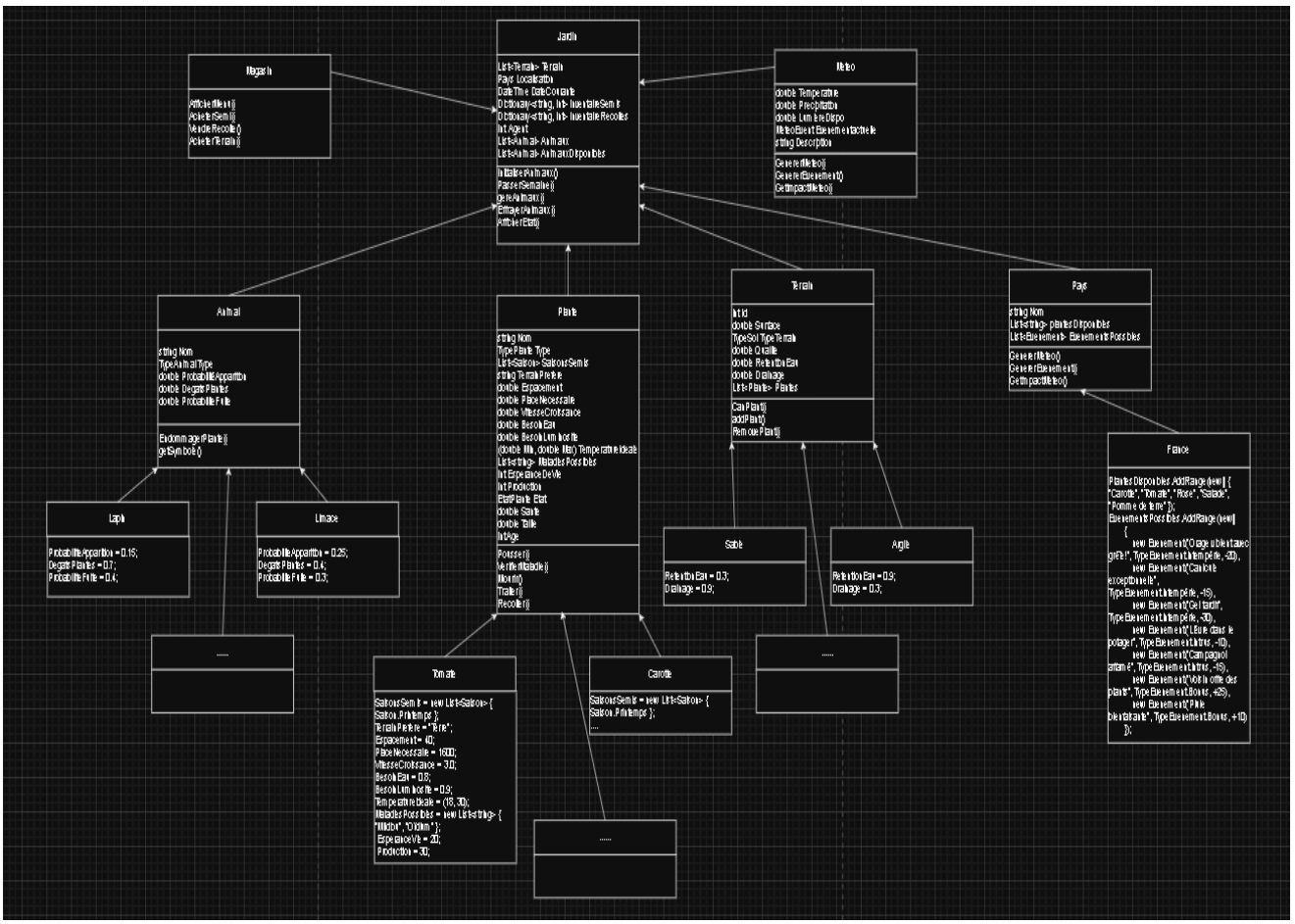
```
    }

    private void TraiterPlantesMalades()
    {
        int plantesTraitees = 0;

        foreach (var terrain in Jardin.Terrains)
        {
            foreach (var plante in terrain.Plantes)
            {
                if (plante.Etat == EtatPlante.Malade)
                {
                    plante.Traiter();
                    plantesTraitees++;
                    Console.WriteLine($"- {plante.Nom} a été traitée");
                }
            }
        }

        if (plantesTraitees == 0)
        {
            Console.WriteLine("\nAucune plante malade à traiter.");
        }
        else
        {
            Console.WriteLine($"\nTotal plantes traitées:
{plantesTraitees}");
        }
    }
}
```

test dans Program.cs:

```
var france = new France();
var jardin = new Jardin(france);




// Initialisation des données
var catalogue = new List<Plante>
{
```

```
    new Carotte(),
    new Tomate(),
    new Rose(),
    new Salade(),
    new PommeDeTerre(),
    new Tournesol(),
    new Cactus(),
    new Bambou(),
    new PlanteMagique(),
    new Mandragore()
};

var simulateur = new Simulateur(jardin, catalogue);
simulateur.AfficherMenuPrincipal();

// Création du simulateur
```

- Diagrammes UML complets


Jardin
Magasin
Meteo
Terrain
Page
Animal
Plante
Sable
Argile
Tomate
Carotte
Laplu
Limace
France