

DEVEZE Léa  
LARGE Corentin  
1A - Groupe 3



# Rapport Projet

## Ensemenc



**Module** : Programmation avancée  
**Encadrante** : Mme TETELIN Angélique  
**Année** : 2024-2025

# Sommaire

I. Introduction.....	1
II. Structuration du code.....	1
III. Modélisation des éléments du jeu.....	2
A. Monde.cs.....	2
B. Case.cs.....	3
C. Plantes.cs (et ses héritiers).....	4
IV. Explication du code.....	5
A. Déroulé du jeu.....	5
B. Récapitulatif des méthodes.....	8
V. Diagramme UML.....	10
VI. Organisation de l'équipe.....	11
A. Répartition des tâches dans le groupe.....	11
B. Planning de réalisation.....	11
VII. Conclusion.....	11

# I. Introduction

Dans le cadre du module de Programmation Avancée, le projet 2025 nous propose de concevoir et développer un simulateur de potager sous la forme d'une application console en C#. L'objectif est de créer un jeu de gestion dans lequel le joueur incarne un jardinier chargé de faire prospérer son jardin dans un environnement hostile, évolutif et stratégique.

Notre jeu se déroule sur la planète Mars. Le joueur commence avec un nombre limité de semis, et un plateau divisé en parcelles de terrain possédant des propriétés bien définies (type de sol...). Il doit planter intelligemment ses graines de plantes, en tenant compte des préférences de chaque espèce végétale : type de biome préféré, besoin en eau, saison de semis, vulnérabilités, etc. Le joueur doit aussi faire attention aux intrusions d'araignées, qui empêche le joueur de planter une graine sur la case correspondante.

Le jeu alterne entre deux modes principaux :

- Le mode classique, qui simule le passage du temps à chaque action et permet au joueur de gérer son potager à travers différentes actions : arroser, semer, et chasser les araignées.
- Le mode urgence, qui s'active ponctuellement pour gérer en temps réel des situations critiques telles que des apparitions massives d'araignées. Ce mode introduit une dimension plus dynamique et stratégique, obligeant le joueur à réagir rapidement pour protéger ses cultures.

## II. Structuration du code

Le fonctionnement du jeu repose sur une architecture orientée-objet bien définie, dans laquelle chaque classe remplit un rôle spécifique au sein de la simulation.

Pour ceci, nous avons donc en plus de Program.cs :

- 3 classes principales : Monde.cs, Case.cs et Plantes.cs
- 5 classes héritières représentant chaque type de plante : Vento.cs, Luma.cs, Spira.cs, Vira.cs et Gral.cs.

La classe Monde est responsable de la gestion globale du jeu : déplacement du joueur sur le terrain, passage du temps, gestion du mode urgence, actions du joueur (planter, arroser et chasser), et affichage du plateau dans la console. Il s'agit d'une classe métier, qui regroupe l'ensemble des mécanismes de simulation et de gestion du potager.

La classe Case gère les caractéristiques propres à chaque case ainsi que l'affichage graphique en console sous forme de blocs d'emojis. C'est une classe dédiée à la représentation concrète des éléments du plateau.

La classe Plantes est une classe abstraite qui définit les caractéristiques communes à toutes les plantes du jeu. Elle contient également une méthode de gestion de l'évolution quotidienne d'une plante et son affichage.





Enfin, la classe Plantes possède 5 classes héritières : Vento, Vira, Spira, Gral et Luma. Ces dernières sont des classes dérivées qui spécialisent chaque type de plante avec des paramètres concrets. Chaque plante devient ainsi une entité unique, possédant un comportement et une identité propre, tout en héritant du fonctionnement générique défini dans la classe abstraite Plantes.

### III. Modélisation des éléments du jeu









Afin de mieux comprendre l'organisation interne du jeu et la répartition des responsabilités entre les différentes classes, nous présentons ci-dessous une série de tableaux récapitulatifs qui décrivent, pour chaque classe, les éléments modélisés, leur type ainsi que leur rôle dans la logique du jeu.

#### A. Monde.cs

Élément	Description
<b>Schéma du terrain</b> str[ , ] schema	Correspond à une matrice de str donnant le type de biome de chaque case
<b>Terrain</b> Tableau à deux dimensions de cases Case[ , ] Grille	Correspond à une grille rectangulaire composée d'objets Case, dont le biome est défini par schema. Chaque case est accessible via Grille[x, y] (x = colonne, y = ligne).
<b>Position du joueur</b> int JoueurX, JoueurY	Coordonnées de la case sélectionnée par le joueur. Elles servent pour planter, arroser ou chasser un intru sur une case donnée.
<b>Jour actuel</b> int Jour	Représente le temps dans le jeu. Tous les 30 jours, le <b>mode urgence</b> est activé.
<b>Mode urgence actif</b> bool EstEnUrgence	Booléen indiquant si le mode urgence est actif. Quand il l'est, les araignées apparaissent plus souvent.

<b>Sélection d'une plante à planter</b> ConsoleKey	Lorsqu'une des touches (E, R, T, Y, U) est appuyée, une plante est choisie et sera plantée.
<b>Déplacement joueur</b> ConsoleKey	Appuyer sur une des touches fléchées déplacent le joueur sur la grille selon la direction de la touche :     . Empêche les déplacements hors de la grille.

## B. Case.cs

Élément	Description
<b>Biome de la case</b> string Biome	Indique la nature du terrain de la case : "Terre", "Sable" ou "Argile". La couleur de la case dépend de son biome :  pour la terre,  pour le sable et  pour l'argile.
<b>Plante présente ou non</b> Plantes? Plante	Contient un objet Plante si l'utilisateur a planté quelque chose sur cette case. Sinon, cette propriété est null. Le "?" permet de rendre cet élément non obligatoire pour initialiser une case.
<b>Eau contenue</b> int EauContenue	Indique le niveau d'eau de la case (entre 0 et 100). Initialisé à 50 par défaut.
<b>Présence d'araignée</b> bool AraigneePresente	Indique si une araignée est présente sur cette case. Défaut : false. Si une araignée est sur une case, cela bloque les plantations.
<b>Apparence visuelle (emojis)</b> string[ ] GetEmojiBlock(bool)	Retourne une représentation en <b>3 lignes d'emojis</b> de la case : Le <b>fond</b> dépend du biome (  ,  ,  ) Le <b>centre</b> affiche soit une araignée (  ) , soit une plante, soit le fond du biome si la case est vide, soit du violet (  ) si la case est sélectionnée par le joueur

## C. Plantes.cs (et ses héritiers)

Élément	Description
<b>Nom de la plante</b> string Nom	Nom affiché de la plante. Utilisé dans les infos visibles du joueur.
<b>Emoji d'affichage</b> string Emoji	Représentation emoji de la plante. Affichée si la plante est vivante et à maturité.
<b>Type de plante</b> string Type	Type agricole ou botanique.
<b>Comestible</b> bool Comestible	Indique si la plante est comestible.
<b>Mauvaise herbe</b> bool MauvaiseHerbe	Indique si la plante est une mauvaise herbe.
<b>Saison de semis</b> string SaisonSemis	Saison idéale pour planter la graine.
<b>Vitesse de croissance</b> int VitesseCroissance	Nombre de points de croissance gagnés chaque jour.
<b>Croissance actuelle</b> int NiveauCroissance	Niveau de croissance de la plante (0 à 100). Augmente chaque jour.
<b>Terrain préféré</b> string TerrainPrefere	Type de biome optimal. Si la plante n'est pas plantée dans son biome préféré, elle perd de la vie plus vite.
<b>Besoin en eau</b> int BesoinEau	Niveau idéal d'eau. Une trop grande différence avec EauContenue fait perdre de la vie.

<b>Temps de vie restant</b> int TempsDeVieRestant	Nombre de jours restants avant que la plante ne meure. Diminue chaque jour, et plus vite si conditions mauvaises.
<b>Vulnérabilités</b> Dictionary<string, float> Vulnerabilites	Facteurs environnementaux nuisibles.

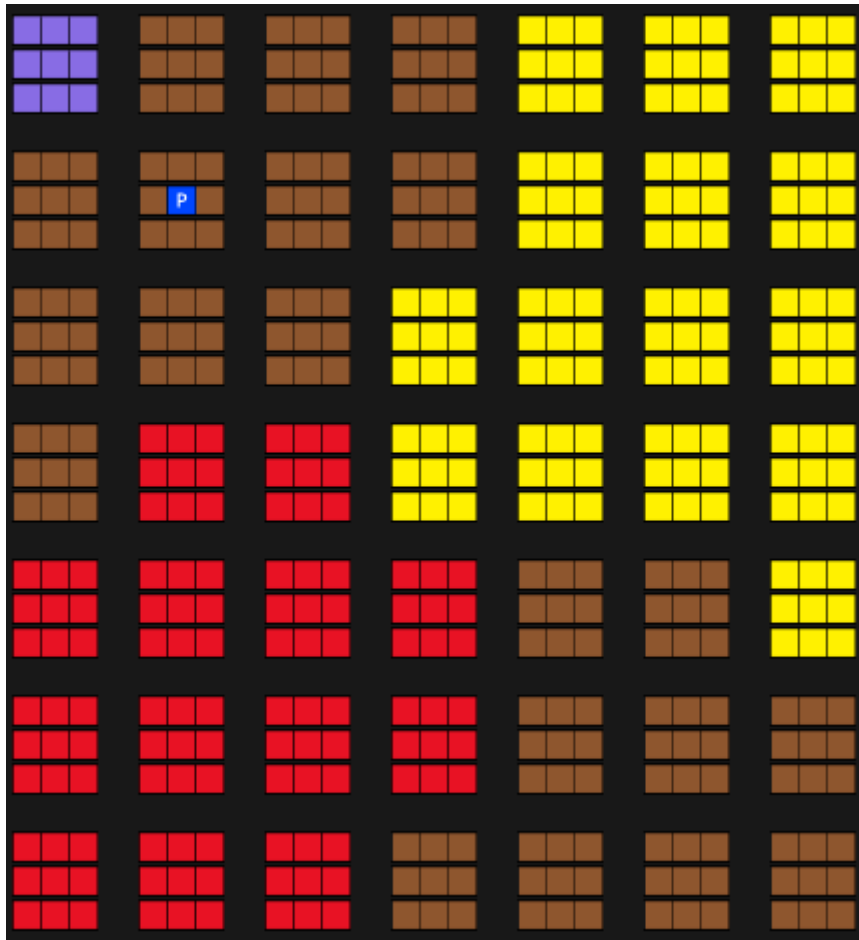
## IV. Explication du code

### A. Déroulé du jeu

Lorsqu'on exécute le fichier Program.cs, le jeu se lance et initialise notre terrain à l'aide de la classe Monde et de sa méthode AfficherMonde(). Cette méthode gère à la fois l'affichage visuel du potager, ligne par ligne, et l'affichage textuel des informations utiles au joueur (biome, plante présente sur la case sélectionnée, et actions disponibles).

Comme nous l'avons expliqué plus tôt, le terrain est constitué d'une grille de parcelles (Case[,] Grille) définies par différents types de sol, dont l'affichage à l'aide d'emojis de couleur permet de rendre ces terrains explicites pour le joueur. Cet affichage est géré dans la classe Case, par la méthode privée FondEmoji(), qui retourne l'emoji de fond correspondant au biome ("■" pour Terre, etc.).

L'affichage visuel global de chaque case est ensuite construit par la méthode GetEmojiBlock(bool estSelectionnee) de la classe Case. Cette méthode retourne un bloc de 9 emojis (3 lignes de 3), représentant graphiquement une case. Elle y intègre la couleur de fond, la présence éventuelle d'une araignée (🕷️) ou d'une plante (via Plantes.GetEmojiAffichage()), et affiche la case en violet (■) si elle est actuellement sélectionnée par le joueur. Le suivi de la position de ce dernier est géré par les attributs JoueurX et JoueurY de la classe Monde, et le curseur se déplace grâce à la méthode BougerJoueur(ConsoleKey key).



L'interface console affiche donc cette grille suivie des informations liées à la case sélectionnée, obtenues via la méthode `GetCaseSelectionnee()` de la classe `Monde`.

```

-----
Jour actuel : 0
-----
Biome de la case : Terre
Pas de plante
-----
Actions possibles :
P : Planter | A : Arroser | C : Chasser
-----

```

Si une plante est présente sur cette case, la méthode `AfficherInfos(int eauContenue)` de la classe `Plantes` est appelée pour afficher ses caractéristiques détaillées (type, saison de semis, croissance, besoin en eau, etc.).



```

-----
Jour actuel : 1
-----
Biome de la case : Terre
Information sur la plante :
Nom: Vento 🍌 | Type: Annuelle | Comestible: Oui | Mauvaise herbe: Non | Saison: Printemps
Croissance: 10/100 | Terrain préféré: Terre | Eau requise: 48/70 | Vie restante: 99 jours
-----
Actions possibles :
P : Planter | A : Arroser | C : Chasser

```

Le jeu tourne à l'aide d'une boucle principale while située dans Program.cs, qui se répète tant que le joueur n'appuie pas sur la touche Échap. À chaque tour, le monde est réaffiché, puis le programme attend que le joueur appuie sur une touche. S'il appuie sur une flèche directionnelle, la méthode BougerJoueur() est appelée pour déplacer le curseur, sans que le temps n'avance. En revanche, certaines touches déclenchent des actions qui consomment une journée entière dans le jeu, ce qui entraîne un appel à la méthode AvancerUnJour(). Parmi ces actions, la touche P permet de planter une graine. Elle appelle la méthode AfficherMenuPlantesEtPlanter() de Monde, qui affiche un menu de sélection de plantes (Vento, Vira, Luma, Spira et Gral) et crée l'objet plante correspondant selon la touche pressée. Ensuite, la méthode Planter(Plantes plante) est appelée pour placer cette plante sur la case sélectionnée, sauf si une araignée y est présente.

```

Sélectionnez une plante à planter :
[E] Vento 🍌
[R] Vira 🌿
[T] Luma 🌿
[Y] Spira 🍌
[U] Gral 🍌

```

La touche A déclenche la méthode ArroserCaseSelectionnee(), qui augmente le niveau d'eau de la case (jusqu'à un maximum de 100). La touche C exécute la méthode ChasserAraignee(), qui supprime une araignée si elle est présente sur la case sélectionnée.

Chaque fois qu'un jour passe, via l'appel à AvancerUnJour(), plusieurs événements se produisent automatiquement. Tout d'abord, le compteur de jours est incrémenté. Ensuite, chaque case du potager subit une évaporation de l'eau (gérée dans une boucle sur toutes les cases), avec un taux de perte plus élevé en mode urgence (défini par l'attribut EstEnUrgence de la classe Monde), mode activé tous les 30 jours.

🔥 Mode URGENCE activé ! Une pluie d'araignée débarque et l'eau vient de fortement s'évaporer !

Cette évaporation est suivie d'une mise à jour de l'état de chaque plante : pour chaque case contenant une plante, la méthode PasserUnJour(string biome, int eauContenue) de la classe Plantes est appelée. Elle gère la diminution du temps de vie restant, l'évolution de la

croissance (NiveauCroissance) et applique des pénalités si le sol ou le niveau d'eau ne correspondent pas aux préférences de la plante.

Enfin, la méthode `AjouterAraigneesAleatoires()` est exécutée pour simuler l'apparition d'araignées sur certaines cases, avec un taux de probabilité de 0,8% par jour en mode normal et 20% en mode urgence. Ce système introduit des menaces ponctuelles, rendant le jeu plus stratégique et obligeant le joueur à bien gérer ses ressources et ses plantations.

L'objectif global du jeu est de faire croître les plantes le plus efficacement possible dans cet environnement martien hostile. Chaque plante possédant ses propres caractéristiques, le joueur doit donc adapter ses choix de culture à chaque case, surveiller l'arrosage, et défendre ses plantations contre les araignées pour assurer la prospérité de son potager.

## B. Récapitulatif des méthodes





Voici des tableaux récapitulatifs des différentes méthodes de Monde, Case et Plantes, et de leurs rôles (à noter que les héritiers de Plantes ne possèdent aucune méthode propre à eux mêmes et n'en modifient aucune tirée de Plantes).

- **Monde.cs**

Nom de la méthode	Rôle
<b>Monde()</b> ( <i>constructeur</i> )	Initialise le monde, place le joueur au coin supérieur gauche et prépare la grille à partir du schéma prédéfini de biomes.
<b>InitialiserMonde()</b>	Crée la grille de jeu ( <code>Case[,]</code> ) à partir d'un tableau 2D de chaînes de caractères représentant les types de sol. Permet d'avoir une structure cohérente et variée du terrain.
<b>Planter(Plantes plante)</b>	Permet d'attribuer une plante à une case spécifique si les coordonnées sont valides et qu'aucune araignée ne bloque la plantation.
<b>AjouterAraigneesAleatoires()</b>	Permet de faire apparaître des araignées sur le plateau. A noter que leur taux d'apparition dépend du mode de jeu en cours (0,8% en classique ou 20% en urgence)
<b>ArroserCaseSelectionner()</b>	Augmente l'eau contenue dans la case sélectionnée de +10, sans dépasser un maximum de 100.

<b>AffihcerMenuPlantesEtPlanter()</b>	Affiche un menu en console permettant au joueur de choisir une plante à planter, puis appelle la méthode Planter() avec la plante sélectionnée.
<b>AvancerUnJour()</b>	Simule le passage d'un jour dans le jeu : incrémente le jour courant, active le mode urgence tous les 30 jours, met à jour l'évaporation de l'eau, la croissance et le vieillissement des plantes, et ajoute aléatoirement des araignées.
<b>ChasserAraignee()</b>	Supprime une araignée si elle est présente sur la case sélectionnée, et affiche un message de confirmation.
<b>GetCaseSelectionnee()</b>	Retourne la case actuellement sélectionnée par le joueur (selon sa position dans la grille).
<b>BougerJoueur(ConsoleKey key)</b>	Met à jour la position du joueur selon les touches directionnelles appuyées (gauche, droite, haut, bas), avec des limites pour ne pas sortir du terrain.
<b>AfficherMonde()</b>	Affiche tout le monde en console avec des emojis, met en évidence la case sélectionnée, affiche les informations de la case (biome, plante, etc.), ainsi que le jour en cours.

- **Case.cs**

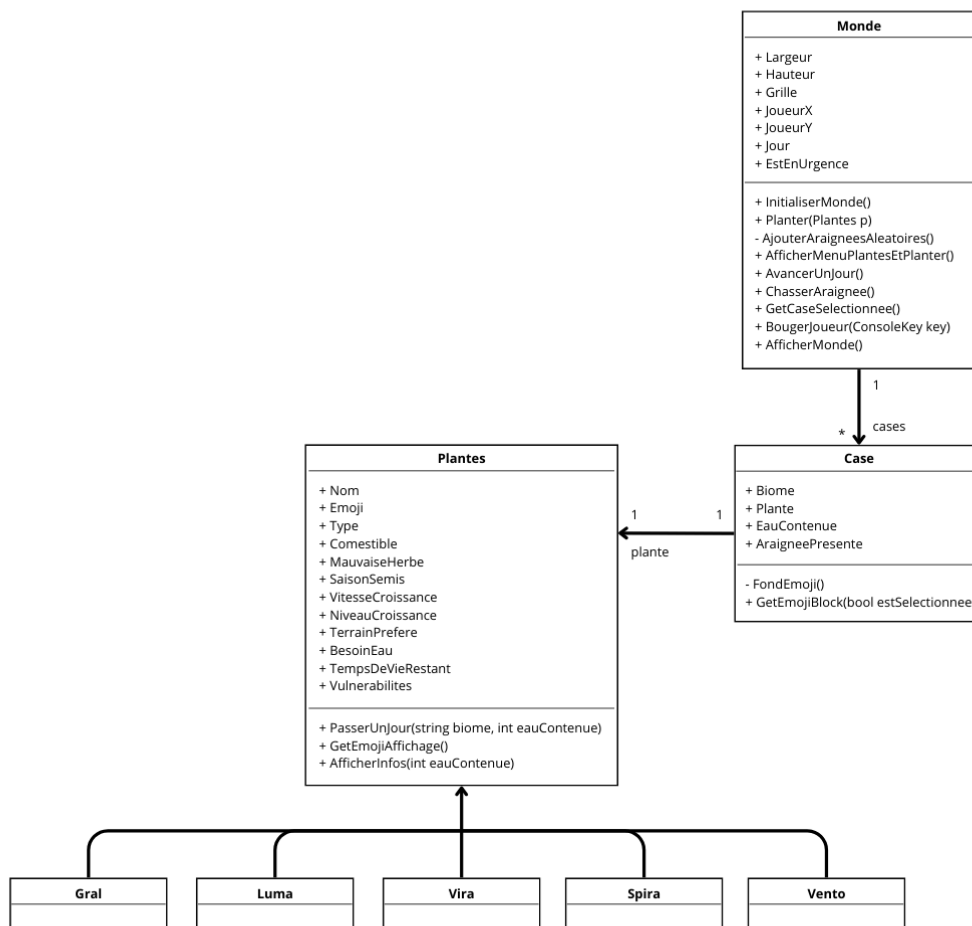
Nom de la méthode	Rôle
<b>Case(string biome) (constructeur)</b>	Initialise une case avec un biome donné, un niveau d'eau par défaut (50). Aucune araignée ni plante présente de base par défaut.
<b>FondEmoji() (privée)</b>	Retourne l'emoji de fond correspondant au type de sol de la case :  pour Terre,  pour Sable,  pour Argile, ou  par défaut.
<b>GetEmojiBlock(bool estSelectionnee)</b>	Retourne un tableau de trois chaînes de caractères représentant visuellement la case (3×3 emojis), incluant le fond, la plante ou l'araignée éventuelle au centre, et une surbrillance violette si la case est sélectionnée.

- Plantes.cs

Nom de la méthode	Rôle
<b>PasserUnJour(string biome, int eauContenue)</b>	Simule l'évolution d'une plante en une journée : réduit le temps de vie restant, augmente la croissance si la plante est encore en vie, et applique des pénalités si le biome ou l'eau ne correspondent pas aux préférences de la plante.
<b>GetEmojiAffichage()</b>	Retourne l'emoji à afficher pour la plante selon son état : un trou (💩) si elle est morte, son emoji si elle est mature, ou (P) si elle est en croissance.
<b>AfficherInfos(int eauContenue)</b>	Retourne une chaîne de caractères détaillant les caractéristiques de la plante (nom, type, comestibilité, saison, croissance, besoin en eau, durée de vie restante).

## V. Diagramme UML

Nous avons précédemment réalisé des tableaux expliquant nos éléments modélisés puis nos méthodes. A présent, voici le diagramme UML de notre programme.



## VI. Organisation de l'équipe

### A. Répartition des tâches dans le groupe

Tout au long de ce projet, nous avons alterné entre des sessions de codage collectives, réalisées pendant les travaux dirigés, et des phases de travail individuel.

La répartition précise du code, documentée dans notre matrice d'implication, a été ajustée progressivement en fonction de l'avancement du projet.

Enfin, le rapport a été rédigé et mis à jour de manière régulière entre chaque séance de travail.

### B. Planning de réalisation

Nous avons commencé le projet le 14 avril 2025 et l'avons achevé le 19 mai 2025. Le planning des réalisations ci-dessous montre notre avancée chronologique. La couleur **bleue** correspond aux tâches qui étaient en cours de réalisation et la couleur **verte** correspond au jour où la tâche a été terminée.

Tâches/Date	14-avr	15-avr		22-avr	23-avr	24-avr		05-mai	06-mai	07-mai		11-mai	12-mai	13-mai	14-mai	15-mai	16-mai	17-mai	18-mai	19-mai
Rédaction du rapport																				
Création du plateau																				
Création de Plantes.cs																				
Developp. de Case.cs																				
Developp. de Monde.cs																				
Affichage du plateau																				
Déplacer le curseur																				
Héritiers Plantes																				
Planter une plante																				
Gestion survie des plantes																				
Gestion des araignées																				
Developp. mode urgence																				

## VII. Conclusion

En conclusion, nous avons un jeu qui fonctionne correctement comprenant les fonctionnalités attendues, dans lequel le joueur incarne un jardinier sur Mars, chargé de faire prospérer son potager dans un environnement difficile. Le joueur peut se déplacer sur une grille de terrains, planter différentes espèces de plantes, les arroser, et lutter contre des araignées envahissantes. Chaque action doit être réalisée stratégiquement et impacte la survie et la croissance des plantes.

Parmi les améliorations qui pourraient enrichir l'expérience de jeu, l'une des plus intéressantes serait de permettre au joueur de récolter ses plantes lorsqu'elles atteignent

leur maturité. Ces plantes pourraient alors être stockées dans un inventaire personnel, accessible à tout moment et pourraient ensuite être mises en vente dans un mode magasin, où le joueur pourrait obtenir de l'argent en fonction de la rareté ou de la qualité de ses cultures. Cet argent virtuel pourrait ensuite être utilisé pour acheter de nouvelles graines, y compris de nouvelles variétés de plantes aux comportements, besoins et vulnérabilités spécifiques.

Enfin, d'autres formes de menaces pourraient également être ajoutées pour complexifier le déroulement du jeu, comme des éléments plus destructeurs que les araignées, qui pourraient manger ou endommager les plantes. On pourrait imaginer des vers souterrains, des champignons toxiques ou des tempêtes martiennes affectant temporairement la croissance de certaines espèces.

Ces ajouts rendraient le jeu plus complet, plus évolutif et offriraient au joueur des objectifs à plus long terme, tout en renforçant l'aspect stratégique de la gestion du potager.