



Programmation Avancée

Projet ENSemenC

MAROT Alex
PELICHET Thibert

Table des matières

Table des matières.....	2
I - Introduction.....	3
II - Univers du jeu.....	3
III - Modélisation des plantes.....	3
IV - Sols et météo.....	3
V - Architecture du potager.....	3
VI - Fonctionnalités supplémentaires.....	3
VII - Sauvegarde et chargement de partie.....	3
VIII - Choix techniques et arbitrages.....	3
IX - Tests et validation.....	3
X - Bilan du projet.....	3

I - Introduction

Ce projet a été réalisé dans le cadre du cours de Programmation Avancée à l'ENSC. L'objectif était de créer un simulateur de potager en C#, en utilisant les principes de la programmation orientée objet. Le jeu se présente sous la forme d'une application console, avec une logique de fonctionnement en semaines, des plantes à cultiver, et un ensemble de paramètres (météo, sol, saison, etc.) qui influencent leur croissance.

Nous avons choisi d'orienter notre projet comme un **jeu de gestion type "plateau"**, dans lequel le joueur peut déplacer un curseur sur une grille de terrain, interagir avec chaque case, et gérer ses ressources (notamment des graines, utilisées comme monnaie). L'idée était de rendre le jeu un peu plus interactif tout en restant dans le cadre du terminal.

Le projet nous a permis de mettre en œuvre une architecture orientée objet complète avec classes abstraites, héritage, polymorphisme, encapsulation, tout en respectant les contraintes du sujet. Nous avons aussi intégré un système de météo aléatoire, un mode urgence, des équipements spéciaux et une sauvegarde complète du jeu.

Certaines décisions ont été prises pour garder une interface cohérente et simple, même si cela a parfois entraîné un peu de duplication ou de couplage.

II - Univers du jeu

Le potager se déroule dans un pays que nous avons choisi : **le Brésil**. Ce choix nous permettait de travailler avec **deux saisons distinctes** (saison pluvieuse et saison sèche), ce qui simplifie la simulation tout en restant crédible sur le plan climatique. Chaque saison dure 26 semaines, soit un cycle annuel de 52 semaines.

Les plantes ont des préférences spécifiques : certaines se cultivent mieux pendant la saison pluvieuse, d'autres survivent mieux à la sécheresse. Ce choix du Brésil nous a donc permis de lier plus facilement les cycles de culture à une temporalité réaliste.

L'univers du jeu repose aussi sur une **représentation de type plateau**, c'est-à-dire une grille (array 2D) de terrains affichée dans la console. Chaque case peut contenir un type de sol et éventuellement une plante, représentée par un acronyme de 2 lettres. Le joueur déplace un curseur sur la grille et peut agir sur les cases (planter, arroser, récolter, etc.). Ce choix d'affichage était à la fois immersif et pratique pour visualiser rapidement l'état du potager. Il nous a toutefois imposé certaines contraintes supplémentaires, notamment au niveau de l'organisation des classes, de la gestion du curseur, et des règles d'espacement entre plantes.

III - Modélisation des plantes

L'ensemble du comportement des plantes repose sur une architecture orientée objet, centrée autour d'une classe abstraite **Plante**. Cette classe regroupe tous les attributs et comportements communs à toutes les plantes, tandis que les plantes spécifiques (Soja, Maïs, Café, etc.) héritent de cette base pour personnaliser leurs propriétés.

Classe Plante

Chaque plante possède un ensemble de paramètres biologiques simulés :

- **Nom, acronyme et prix** en graines
- **Espacement requis** autour d'elle pour éviter les conflits de croissance
- **Type de terrain préféré** (sableux, argileux, etc.)
- **Saisons compatibles** (pluvieuse ou sèche)
- **Plage de température idéale**
- **Besoins en hydratation et luminosité**
- **Vitesse de croissance**
- **Espérance de vie (en semaines)**

Tous ces paramètres sont initialisés dans les classes filles, en fonction du type de plante choisi. Par exemple, le cactus tolère bien la sécheresse et aime la chaleur, tandis que le maïs a besoin d'eau et pousse plus rapidement.

Les 7 conditions de croissance

Pour simuler de manière simplifiée la survie des plantes, nous avons défini un système à 7 critères :

1. Température dans la plage tolérée
2. Hydratation suffisante
3. Luminosité correcte
4. Saison compatible

5. Terrain adapté
6. Espacement respecté
7. Absence d'obstacle (maladie, animal...)

Chaque semaine, la plante évalue ces 7 conditions. Si au moins 4 ne sont pas respectées, elle meurt. Si 3 sont mauvaises, elle passe en état de danger (affichage orange). Si tout va bien, elle continue à pousser. Cela donne une gestion assez fine de l'environnement sans être trop lourde à implémenter.

Croissance et récolte

La plante pousse chaque semaine en fonction de la satisfaction de ses conditions. Une variable de "satisfaction cumulée" permet de calculer un rendement plus ou moins élevé au moment de la récolte. Une fois arrivée à maturité, la plante peut être récoltée et rapporte un certain nombre de graines au joueur.

Plantes vivaces

Certaines plantes (comme le cactus, le café ou la canne à sucre) sont **vivaces**, ce qui signifie qu'elles ne meurent pas après une récolte. Au lieu de cela, elles entrent dans une phase d'attente pendant laquelle elles ne produisent plus. Après un délai de dix semaines, et seulement si la saison est compatible, elles peuvent refleurir et produire à nouveau. Ce mécanisme rend la gestion de ces plantes plus stratégique et a nécessité l'ajout de quelques champs supplémentaires (vivacité, délai de récolte, état de production). Indépendamment de la vivacité, les plantes ont une espérance de vie et finissent tôt ou tard par mourir.

Cas particulier : le compost

Le compost est une plante spéciale, qui n'a pas besoin de pousser ni de conditions optimales. Elle permet au joueur de recycler les plantes désherbées. À chaque désherbage (hors compost), la jauge de remplissage du compost augmente. Une fois remplie à 100 %, le joueur gagne 15 graines, et la jauge repart à zéro. Le compost est unique sur le plateau, ce qui oblige à bien choisir son emplacement. C'est un exemple concret de polymorphisme dans notre projet.

Monnaie du jeu : les graines

Le jeu utilise un système de monnaie simple basé sur les **graines**. Le joueur démarre la partie avec un stock initial (100 graines) qu'il peut utiliser pour planter, arroser ou interagir avec certains éléments du potager. Chaque plante a un **prix d'achat** défini dans sa classe (par exemple, 10 graines pour du soja, 20 pour un cactus). Arroser une plante coûte également une petite quantité de graines (5 unités).

En retour, les plantes permettent de **gagner des graines** lorsqu'elles sont récoltées, à condition qu'elles soient arrivées à maturité. Le nombre de graines obtenues dépend à la fois du rendement de la plante et de sa "satisfaction moyenne", calculée sur toute sa durée de croissance.

Le compost permet également de récupérer des graines. À chaque désherbage (sauf si on enlève le compost lui-même), une portion de plante est ajoutée au compost. Lorsqu'il atteint 100 % de remplissage, le joueur reçoit automatiquement 15 graines.

Ce système de gain et de dépense oblige le joueur à gérer ses ressources avec un minimum de stratégie, tout en maintenant une boucle de jeu simple et fluide.

IV - Sols et météo

Le comportement des plantes dans le potager dépend fortement de deux éléments extérieurs : le **type de sol** sur lequel elles sont plantées, et les **conditions météorologiques** qui varient chaque semaine. Ces deux éléments influencent directement plusieurs des sept conditions d'évaluation utilisées par les plantes.

Types de terrain

Chaque case du plateau correspond à un terrain appartenant à une des classes suivantes :

TerrainSableux, **TerrainArgileux**, **TerrainAquatique**, ou **TerrainClassiqueTerreux**. Ces terrains héritent tous d'une classe abstraite **Terrain**, qui définit les propriétés communes :

- **Nom du terrain**
- **Fertilité** (qui pourrait être utilisée dans des extensions futures)
- **Capacité d'absorption d'eau**
- **Couleur affichée dans la console**

Ces terrains sont attribués aléatoirement au début de la partie à l'aide d'un générateur de biomes (**GenerateurBiome**), ce qui permet de créer des zones cohérentes (groupes de cases de même type) et d'éviter une répartition trop chaotique. Le joueur n'a pas de contrôle direct sur la nature des sols, mais doit adapter ses plantations en fonction.

Certaines plantes ont une **préférence de terrain** (par exemple, le cactus pousse mieux dans le sable, le café préfère l'argile). Si cette préférence n'est pas respectée, la plante en souffrira dans son évaluation hebdomadaire.

Simulation météorologique

Chaque semaine, une météo est générée de manière semi-aléatoire à l'aide d'une classe **Meteo**. Cette météo dépend de la **saison en cours** (pluvieuse ou sèche), ce qui influence les plages de température, les chances de pluie, la luminosité moyenne, ainsi que la probabilité d'intempéries.

Les paramètres simulés sont :

- **Température moyenne de la semaine**
- **Quantité de pluie** (entre 0 et 1)
- **Indice de luminosité** (entre 1 et 5)
- **Présence ou non d'intempéries**

Ces valeurs influencent directement les plantes, via des appels aux méthodes **SetTemperature()** et **SetLuminosite()**, qui prennent aussi en compte l'équipement éventuellement présent sur la plante (par exemple, une serre augmente la température reçue).

Les saisons s'alternent automatiquement toutes les 26 semaines. Cela ajoute une contrainte temporelle intéressante, car certaines plantes ne poussent correctement que pendant l'une des deux saisons.

V - Architecture du potager

L'organisation du code repose sur une séparation claire entre la logique du jeu, l'affichage, et les interactions du joueur. Trois classes principales gèrent le cœur de la simulation : **GestionPotager**, **GestionPlateau**, et **VuePotager**.

Classe GestionPotager

Cette classe est le **point central du jeu**. Elle instancie le plateau, gère le temps (semaine après semaine), les saisons, la météo, le stock de graines, ainsi que le déclenchement aléatoire des urgences.

Elle contient la boucle principale de simulation. Chaque tour de jeu représente une semaine. Lorsqu'un joueur choisit d'avancer dans le temps, une nouvelle météo est générée, les plantes sont mises à jour et ont une probabilité de contracter une maladie ou de faire face à un animal (insecte, rongeur..). A chaque tour il y a une chance sur 10 que le mode urgence se déclenche

Elle se charge également d'initialiser tous les composants du jeu : génération des terrains, instanciation des classes de gestion et de vue, et application de la météo en cours à toutes les plantes. En cas de chargement d'une sauvegarde, elle permet de reconstituer l'ensemble de l'état du potager.

Classe GestionPlateau

Cette classe gère **les interactions du joueur avec le plateau**. Elle conserve les coordonnées du curseur, détecte les entrées clavier (flèches, espace, etc.), et déclenche les actions associées à une case :

- Planter une graine
- Arroser une plante
- Récolter une plante
- Désherber une case
- Poser ou retirer un équipement
- Afficher les informations détaillées

Elle est aussi responsable de l'application de la météo aux plantes ainsi que de l'appel à la méthode **Update()** de chaque plante à chaque nouvelle semaine. La logique d'espacement est également gérée ici : certaines plantes ont besoin d'un certain rayon de cases vides autour d'elles, sous peine de ne pas pousser correctement.

Enfin, cette classe garde la référence du compost s'il est placé sur le terrain, afin de gérer sa jauge de remplissage quand une plante est désherbée.

Classe VuePotager

C'est cette classe qui s'occupe de **l'affichage dans la console**. Elle affiche :

- Le plateau de terrain avec les plantes visibles
- Les couleurs correspondant à l'état des plantes (vert, rouge, orange, blanc)
- Les informations détaillées sur une case
- Les options disponibles pour le joueur selon le contexte
- L'interface en mode urgence

Elle est conçue pour être simple à utiliser depuis **GestionPlateau**, et affiche également la météo actuelle, les semaines, et le nombre de graines restantes.

VI - Fonctionnalités supplémentaires

Pour enrichir le gameplay et répondre aux « bonus » suggérés, nous avons ajouté trois extensions majeures : les équipements, et le mode urgence.

VI.1 Équipements (serre et ombrelle)

Le joueur peut placer une serre ou une ombrelle sur une plante déjà en place, moyennant un coût en graines à définir dans le magasin si implémenté.

- **Serre** : augmente de 3 °C la température reçue et accroît l'indice de luminosité de 1.
- **Ombrelle** : réduit de 2 °C la température reçue et diminue l'indice de luminosité de 1.
Ces effets sont appliqués dans `SetTemperature` et `SetLuminosite` avant l'évaluation hebdomadaire. Ils aident à protéger les plantes lors de pics de froid ou de forte chaleur, et à optimiser la croissance en cas de luminosité sous-optimale.
- **Remède** : permet de soigner les maladies
-

VI.2 Obstacles (animaux, insectes, maladies)

Des obstacles peuvent apparaître aléatoirement sur les plantes :

- **Animaux** (les rongeurs sont très rares et mortels, les oiseaux sont plus fréquents et volent seulement les graines de la plante, réduisant les récoltes.)
- **Insectes** (pucerons : ils réduisent l'hydratation de la plante)
- **Maladies** (mildew : déshydrate la plante. rouille : ralentit la croissance de la plante).

Chaque maladie :

- Applique un effet négatif en `Update()`.
- Les maladies peuvent être soignées grâce à un remède qui coûte des graines
- La maladie se propage aux plantes voisines dans un rayon de 2 cases autour de la plante malade, avec le même type de maladie.

VI.3 Mode urgence

Lorsqu'une semaine s'achève, il y a 10 % de chances qu'un mini-jeu mode urgence se déclenche. Nous en avons implémenté trois :

1. **Orage** : protéger les plants d'un orage en déployant une bâche.
2. **Nettoyage** : QTE avec les flèches pour nettoyer le potager : on gagne des graines si on finit à temps
3. **Intrusion nocturne** : chasser un animal (sanglier, rat, chauve-souris, loup) en sélectionnant le répulsif adapté. En cas d'échec, une plante vivante est tuée aléatoirement.

Chaque mini-jeu est indépendant et renvoie un succès ou un échec. Cette mécanique brise la routine, apporte du rythme et exige des décisions rapides.

Ces fonctionnalités renforcent la stratégie, obligent le joueur à anticiper les événements et exploitent pleinement les ressources disponibles.

VII - Sauvegarde et chargement de partie

Pour permettre au joueur de reprendre sa partie ultérieurement, nous avons implémenté un système **entièrement basé sur des fichiers JSON**. Les données sont sérialisées dans un répertoire **Saves/**, chaque slot correspondant à un fichier **<nom>.json**.

Structure des données

La classe centrale est **SaveData**, qui contient :

- **Contexte général** : semaine actuelle, nombre de graines.
- **Météo** : pluie, température, luminosité, intempérie, saison.
- **Terrains** : liste de **TerrainCell** (coordonnées + type de sol).
- **Plantes** : liste de **PlantCell**, qui stocke pour chaque plante son type, ses coordonnées, son état de croissance et de vie (hauteur, semaines depuis plantation, satisfaction, vivacité, obstacle, équipement, état de compost le cas échéant).

Sauvegarde

La méthode **SaveManager.Sauvegarder(string slot, GestionPotager sim)** :

1. Construit un objet **SaveData** à partir de l'état complet de **GestionPotager**.

2. Sériialise cet objet en JSON indenté (pour la lisibilité).
3. Écrit le fichier dans `Saves/<slot>.json`.

La classe garantit l'existence du dossier `Saves` dans son constructeur statique. Un utilitaire `SaveManager.ListerSlots()` fournit la liste des fichiers disponibles, et `SaveManager.Charger(string slot)` désériialise le JSON en `SaveData`.

Chargement

Dans `Program.cs`, le menu de chargement :

1. Recrée le plateau en appelant `GenerateurBiome.CreerTerrainParNom()` pour chaque `TerrainCell`.
2. Reconstruit un `GestionPotager` via son constructeur de restauration, en lui passant la météo et la semaine.
3. Pour chaque `PlantCell` :
 - Appelle `GestionPotager.CreerPlanteParNom()` pour obtenir l'instance correcte.
 - Restaure tous les champs privés ou protégés (hauteur, satisfaction, vivacité, obstacle, équipement, remplissage du compost).
 - Place la plante sur le plateau.

Cette approche garde le code du jeu **completement découplé** de la logique de sérialisation : les classes de simulation ignorent le format JSON, et tout le chargement est géré dans le menu principal. Cela assure à la fois **simplicité d'extension** (ajout d'un nouveau paramètre se traduit par une mise à jour de `SaveData`) et **robustesse** (simple relecture des fichiers).

VIII - Choix techniques et arbitrages

Pour ce projet, nous avons dû arbitrer entre simplicité de mise en œuvre, clarté du code et richesse fonctionnelle. Voici les principaux points de conception et les concessions associées :

1. **Interface 100 % console**

- Avantage : pas de dépendances externes, déploiement immédiat sur n'importe quelle machine Windows.
- Inconvénient : il faut gérer manuellement le curseur, les couleurs et le rafraîchissement de l'écran dans `VuePotager`.

2. Représentation en grille (`Terrain[,]_plateau`)

- Avantage : navigation et affichage naturels, modèles de plateau faciles à visualiser.
- Inconvénient : exposition de méthodes comme `GetTerrain(x,y)` pour accéder aux cases, et logique d'espacement parfois répétitive.

3. Boucle centrale dans `GestionPotager.LancerSimulation()`

- Avantage : point d'entrée unique pour la météo, la mise à jour des plantes et le passage des semaines.
- Inconvénient : la dépendance forte entre la boucle de temps, la météo et la propagation des urgences crée un couplage élevé.

4. Propagation des obstacles et maladies

- Implémentée dans `GestionPlateau.PropagerMaladieAutour(x,y)` pour centraliser la logique.
- Rayon étendu à 2 cases pour plus de réalisme, au prix d'une double boucle de parcours.

5. Gestion de la météo

- Simulation simplifiée via la classe `Meteo`, générée chaque semaine selon la saison.
- Synchronisation nécessaire entre `SetMeteo()` et l'appel à `Update()` dans `GestionPlateau` et dans chaque plante, entraînant un peu de code redondant.

6. Équilibre entre encapsulation et praticité

- Certaines propriétés sont exposées (hauteur, semaines depuis plantation, état des obstacles) pour faciliter la sauvegarde et la restauration.
- Cela peut sembler contrevenir à un strict principe d'encapsulation, mais cela réduit fortement la complexité du chargement.

7. Modularité des mini-jeux

- Chaque urgence implémente l'interface `MiniJeu.Run()` et reste autonome, sans modifier la boucle principale.
- Cela permet de rajouter, supprimer ou tester un seul mini-jeu sans impacter les autres.

Conclusion

Nous avons choisi une architecture simple à comprendre et à étendre, même si cela implique quelques duplications mineures et un couplage un peu plus fort. Ces arbitrages nous ont permis de livrer un simulateur complet, souple, et conforme aux exigences pédagogiques du cours.

IX - Tests et validation

Pour garantir la stabilité et la cohérence du simulateur, nous avons combiné une approche manuelle, orientée scénario, et quelques vérifications automatisées ponctuelles sur les classes clés.

9.1 Méthodologie de test

- **Tests unitaires ponctuels**
Nous avons écrit des tests simples pour les méthodes fondamentales de Plante (croissance, récolte, refloweraison), de Meteo (génération de valeurs dans les plages attendues) et de sauvegarde (SaveManager.Charger/Sauvegarder). Ces tests confirment que les calculs de satisfaction, les bornes de température et d'humidité, et la sérialisation JSON fonctionnent correctement.
- **Tests d'intégration manuels**
L'essentiel des tests s'est fait en console, en suivant des scénarios de jeu complets. Pour chaque fonctionnalité, nous avons lancé plusieurs parties, reproduit des cas limites (ex. plus de 4 conditions ratées), et observé l'affichage et les fichiers de sauvegarde.
- **Itérations rapides**
Après chaque ajout (vivacité, compost, mini-jeu urgence, propagation de maladie, sauvegarde des équipements), nous avons procédé à un cycle rapide : compilation → lancement → reproduction du scénario → correction éventuelle.

9.2 Scénarios validés

1. **Cycle de plantation et récolte**

- Validation du coût en graines à la plantation.
- Croissance selon la météo (différentes saisons).
- Couleur de l'acronyme de la plante (rouge/orange/vert/blanc) selon l'état de chaque plante.
- Récolte correcte (période de maturité, calcul du rendement proportionnel à la satisfaction).



2. Plantes vivaces

- Récolte sans suppression de la plante.
- Blocage de la production pendant 10 semaines, puis reflower uniquement en saison compatible.

3. Compost

- Plantation unique et désherbage correct : +25 % par plante désherbée.
- À 100 %, attribution de 15 graines et remise à zéro de la jauge.

4. Équipements

- Pose et retrait de serre/ombrelle via le menu.
- Effet sur la température et la luminosité dans SetTemperature et SetLuminosite.
- Impact visible sur les 7 conditions (affichage de /.

5. Propagation des maladies

- Lorsqu'une plante porte une **Maladie**, vérification de la propagation dans le rayon de 2 cases.
- Conservation du type de maladie originel.

6. Mode urgence

- Déclenchement aléatoire (10 %).
- Mini-jeux "Orage", "Nettoyage" et "Intrusion nocturne" lancés indépendamment.

- En cas d'échec, une plante vivante est supprimée au hasard.

7. Sauvegarde et chargement

- Création de plusieurs slots, écriture et lecture de JSON indenté.
- Restauration fidèle de l'état complet : météo, graines, plateau, plantes (état de croissance, obstacles, équipements, compost).
- Passage sans erreur de nouveaux paramètres (vivace, délai de refloraison, remplissage compost).

8. Robustesse globale

- Manipulation de coins extrêmes (aucune plante, plateau plein, disparition de toutes les graines).
- Vérification de l'absence de plantages et de warnings à la compilation.

Ces tests ont confirmé le bon fonctionnement de l'ensemble des fonctionnalités et ont mis en évidence quelques ajustements mineurs, aujourd'hui intégrés et validés.

X - Bilan du projet

Le développement de ce simulateur de potager nous a offert une expérience complète, de la conception à la mise en œuvre, en passant par la validation et la documentation. Voici notre retour d'expérience final.

X.1 Points positifs

- **Architecture orientée objet** : classes abstraites (Plante, Terrain), héritage et polymorphisme ont permis de factoriser le code et de faciliter l'ajout de nouvelles plantes ou de nouveaux terrains.
- **Interface console cohérente** : malgré les contraintes d'une application textuelle, le plateau, les couleurs et le curseur offrent une navigation intuitive.
- **Richesse fonctionnelle** : météo, plantes vivaces, compost, équipements, propagation de maladies et urgences (mini-jeux) forment un ensemble de mécaniques variées et complémentaires.

- **Sauvegarde/restauration complète** : format JSON simple et extensible, couvrant l'ensemble de l'état du jeu, ce qui assure une expérience de relais fiable pour l'utilisateur.

X.2 Points à améliorer

- **Encapsulation et découplage** : certaines méthodes (mise à jour météo, propagation de maladies) sont placées dans GestionPlateau pour plus de commodité, mais gagneraient à être déléguées à des classes spécialisées.
- **Répétition de code** : quelques blocs de synchronisation (température, luminosité) se retrouvent à plusieurs endroits. Une refonte légère permettrait de centraliser ces mises à jour.
- **Tests automatisés limités** : le cœur des tests reste manuel. Un jeu de tests unitaires et d'intégration plus étoffé améliorerait la robustesse avant chaque commit.

X.3 Compétences acquises

- **Maîtrise de la POO en C#** : gestion des classes abstraites, interfaces, réflexion pour la propagation des maladies.
- **Gestion d'un projet complet** : du brainstorming (choix du Brésil, deux saisons) à la mise en production (codes, sauvegarde, rapport).
- **Travail en binôme** : coordination des tâches, revues croisées de code et intégration continue manuelle.
- **Interface console avancée** : manipulation du terminal, couleurs, curseur et timers pour les mini-jeux d'urgence.

En conclusion, ce projet nous a permis de développer un simulateur ludique et didactique, tout en renforçant nos bonnes pratiques de programmation orientée objet et notre capacité à gérer un projet de bout en bout.