DELBREIL Laura

L'HERMITE Gaëlle

PEPIN POUSSARD Lou

BORDEAUX INP Ensc

- Programmation avancée - Projet 2025 -

"RAPPORT ENSEMENC"

A. TETELIN

23 mai 2025

INTRODUCTION

Dans le cadre des enseignements dispensés de Programmation Avancée, nous avons été amenés à concevoir et à développer un jeu de simulation de jardinage dit ENSemenCe. Ce jeu à pour but de proposer une expérience « agricole » à l'utilisateur en lui permettant de cultiver des plantes dans des terrains variés tout en s'adaptant aux caprices météorologiques et en protégeant ses parcelles des divers intrus qui pourraient s'y glisser.

Pour ce projet, nous avons utilisé le langage C# afin de concevoir une interface sur le terminal VSCode. Notre code est structuré en diverses classes principales que nous vous présenterons dans ce rapport. Ce rapport vise à décrire de manière structurée et technique l'ensemble des aspects du projet : l'univers du jeu, ses mécaniques, la conception logicielle, notre organisation d'équipe, ainsi qu'un bilan critique sur le déroulement du projet.

I. UNIVERS DU JEU & SPÉCIFICITÉS

- Contexte

Notre version du jeu repose sur un concept original associant culture et stratégie. Ici, le joueur incarne un cultivateur de plantes à la réputation controversée tel que le haschich, la coca, le khat, l'opium, la salvia ou encore les champignons hallucinogènes. Chaque espèce est spécifique, elles ont des besoins d'eau et de lumière, des préférences de terrains et de température et des comportements biologiques qui leurs sont propres. La partie se déroule dans un monde fictif ou les contraintes et les intrus sont soigneusement choisis pour amuser et divertir le joueur. L'environnement est structuré sous forme de terrains agricoles de types différents (argile, terre et sable) divisés en 6 parcelles distinctes. De plus, le jardin est régulièrement soumis à des alertes météo extrêmes et à l'intrusion de menaces externes diverses.

- Environnement dynamique

Dans ce monde, les saisons influencent la température, la lumière et la pluviométrie, affectant directement la croissance des plantes. Des évènements climatiques imprévus comme des gelées, des sécheresses ou des pluies torrentielles par exemple, viennent aléatoirement bouleverser le jardin.

- Indésirables

Les indésirables, ou intrus, menacent régulièrement et aléatoirement le jardin. Ces intrus peuvent être des policiers, des chiens renifleurs, des rats ou encore des maladies, ils se déplacent sur les parcelles en détruisant les plantes sur leurs passages. Pour chaque intrus, le joueur dispose d'une solution spécifique pour s'en débarrasser, sinon, il détruira toutes les plantations.

- Objectif du jeu

L'objectif du joueur est de réussir à faire vivre ses plantations jusqu'à la récolte. En assurant leur croissance, l'arrosage et en les protégeant contre les intrus et les intempéries.

II. FONCTIONNEMENT ET DÉROULEMENT D'UNE PARTIE

Le programme du jeu est articulé autour de cycles de 14 jours. À chaque tour, le joueur peut organiser son jardin et ses actions comme il le souhaite. L'interface du jeu propose des commandes de navigation interactives permettant de se déplacer et d'agir sur les parcelles et sur le jardin.

- Actions joueur

Parmi les actions joueur existant dans le jeu, on retrouve :

- Planter (p) : Le joueur se positionne sur la parcelle de son choix et peut choisir parmi la liste des plantes possibles. Les plantes sont affichées dans un menu intéractif avec le stock de graines disponible. Les besoins spécifiques de la plante sont également affichés pour permettre au joueur de bien sélectionner son terrain.

- Arroser (a) : Le joueur ajoute 1 litre d'eau à la parcelle sélectionnée. Une plante doit être présente pour qu'elle puisse être arrosée. De plus, ce niveau d'eau fluctue en fonction de la météo et de l'évaporation naturelle.

- Récolter (r) : Si la plante est arrivée à maturité, c'est -à -dire que sa croissance dépasse les 75%, alors le joueur peut la récolter afin d'obtenir des graines. Une plante mature qui n'est pas récoltée finira par mourir.

- Bloquer (b) : Si un intrus est présent dans le jardin, le joueur peut tenter de le bloquer en se positionnant sur la case qu'il occupe et en choisissant la solution adaptée dans un menu déroulant interactif. Si la bonne solution est sélectionnée alors l'intrus est chassé. Si l'intrus n'est pas arrêté, il tuera les plantes qui se trouvent sur son passage et s' il atteint le bout du jardin, toutes les plantes mourront.

- Flèches directionnelles : Permettent au joueur de naviguer dans le jardin.

- Touche Entrée : permet de valider les actions en général et de passer au tour suivant en faisant avancer la temporalité de 14 jours et d'appliquer les effets météo et de croissance des plantes.

- **Déroulement d'une partie**

- Démarrage : Le joueur choisir la durée, en année, de la partie qu'il souhaite jouer, puis le nombre de terrains qu'il souhaite posséder dans son jardin (1, 4 ou 9 possibles)

- Initialisation : Les types de terrains sont définis aléatoirement et la météo est générée. Le nombre de graines de départ est prédéfinis et la grille de jardin est mise en place. Le premier affichage propose au joueur une interface avec toutes les informations nécessaires au bon déroulement du jeu.

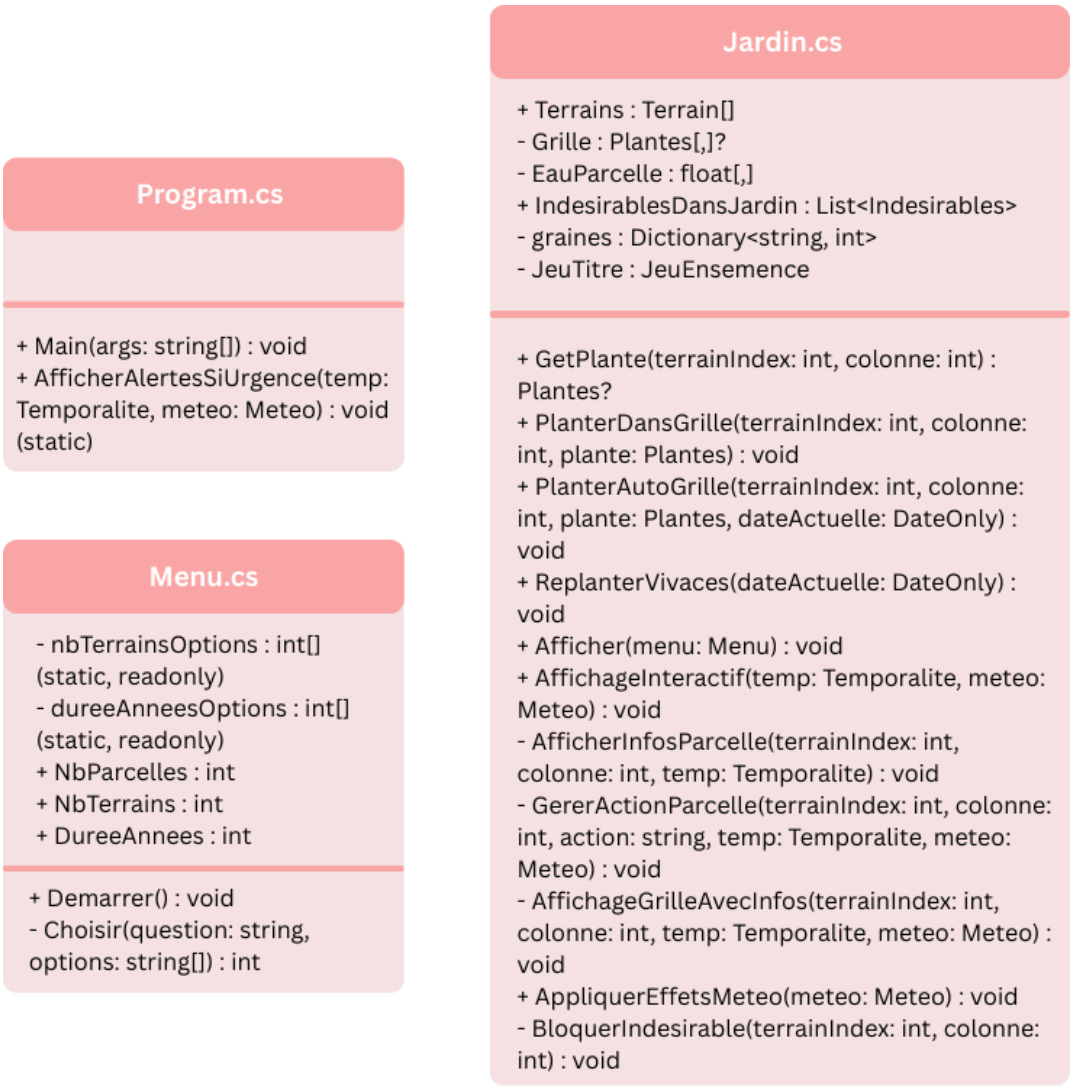
- Cycle de jeu : Avant chaque tour, les effets météo sont appliqués au jardin, avec possiblement l'apparition aléatoire d'une urgence climatique ou d'un intrus. Si une urgence de ce type est déclarée, alors l'avancée temporelle ralentit de 14 à 1 jour, afin de permettre au joueur de régler le problème pour sauver ses plantations. Pendant le tour de jeu, le joueur peut se déplacer et effectuer les actions précédemment présentées sur le jardin. Puis, lorsque toutes les actions sont effectuées, il appuie sur entrer pour passer au tour suivant. Après le tour de jeu, la croissance des plantes est calculée selon leur santé, la météo et le type de sol dans lequel elles grandissent afin de se mettre à jour pour le prochain tour.

- Fin de la partie : La partie se termine lorsque la durée prévue est écoulée.

III. MODÉLISATION OBJET DE NOS CLASSES

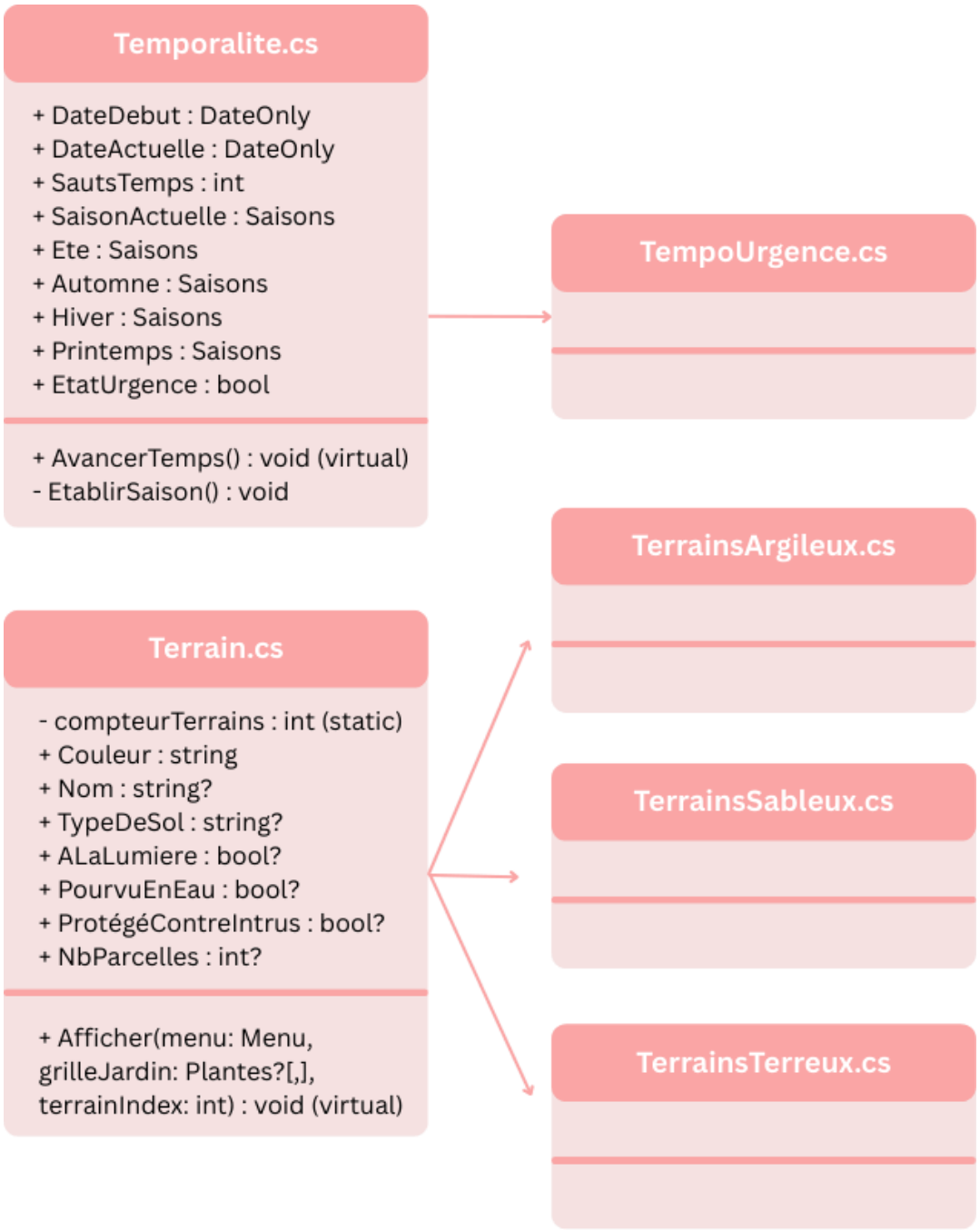
Pour réaliser ce projet, nous avons créé un total de 22 classes (Program.cs y compris) pour offrir au joueur une expérience fluide de notre simulateur de jardin. Les classes, leurs attributs et leurs méthodes sont présentées ci-dessous :

- **Program.cs** : classe contenant le point d'entrée du programme (Main), gère le lancement du jeu, le menu, l'initialisation du jardin et la boucle principale des tours de jeu.
- **Menu.cs** : classe qui permet d'afficher pour le joueur le menu interactif.
- **Jardin.cs** : classe qui permet de gérer le jardin, en stockant les terrains et la grille de jeu.



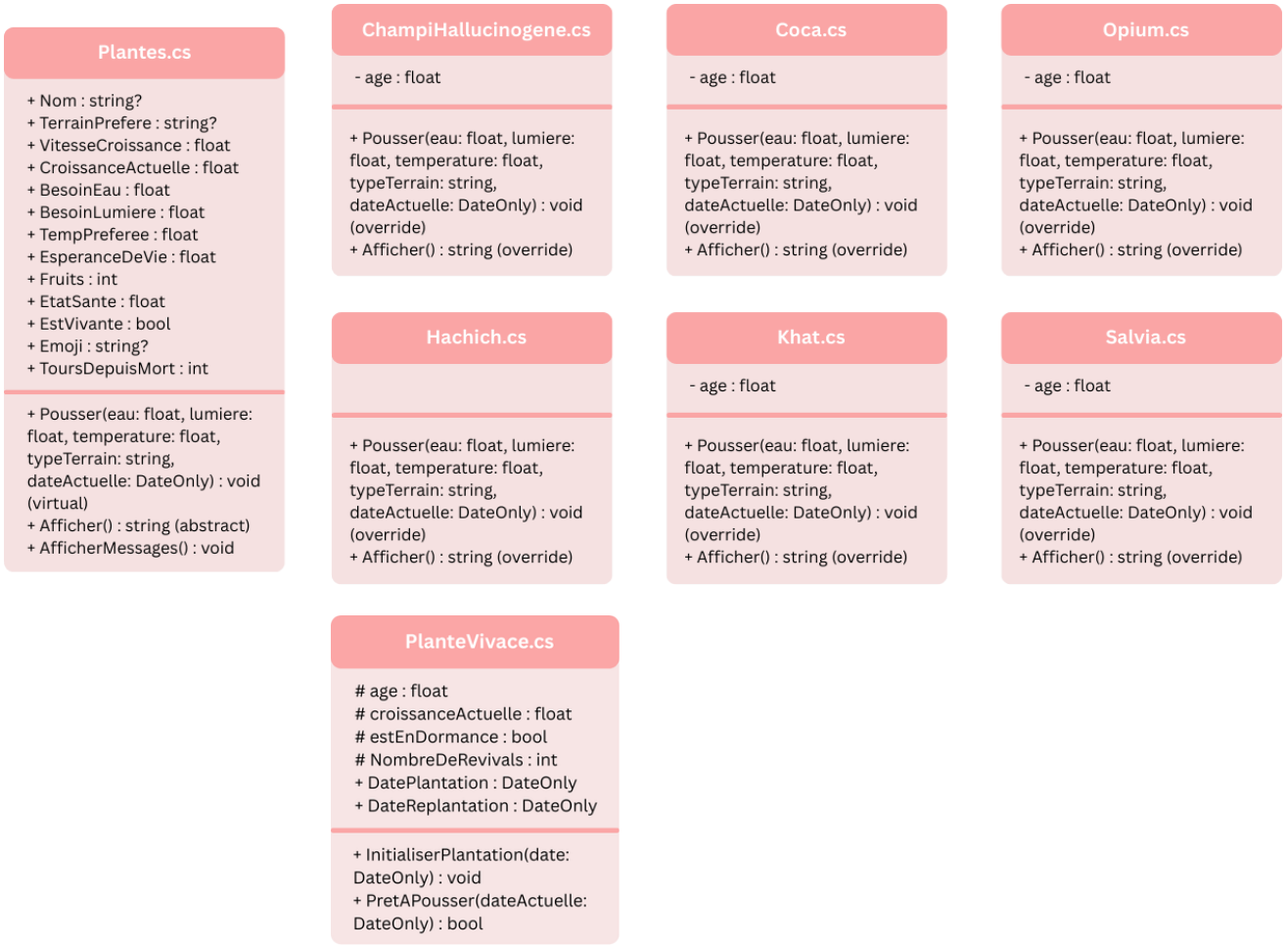
- **Terrain.cs** : classe mère qui permet de définir les caractéristiques générales d'un terrain. Trois classe sont héritières de cette classe :
  - **TerrainsArgileux.cs** : qui modélise un terrain dont le type de sol est l'argile.
  - **TerrainsSableux.cs** : qui modélise un terrain dont le type de sol est le sable.
  - **TerrainsTerreux.cs** : qui modélise un terrain dont le type de sol est la terre.

- **Temporalite.cs** : classe mère qui permet de gérer le passage du temps en mode classique dans le jeu, gère la date et la saison actuelle. Une classe hérite de cette classe :
    - **TempoUrgence.cs** : classe qui permet de gérer la temporalité lors du passage en mode urgence (temps qui passe de jour en jour plutôt que de 2 semaines en 2 semaines).

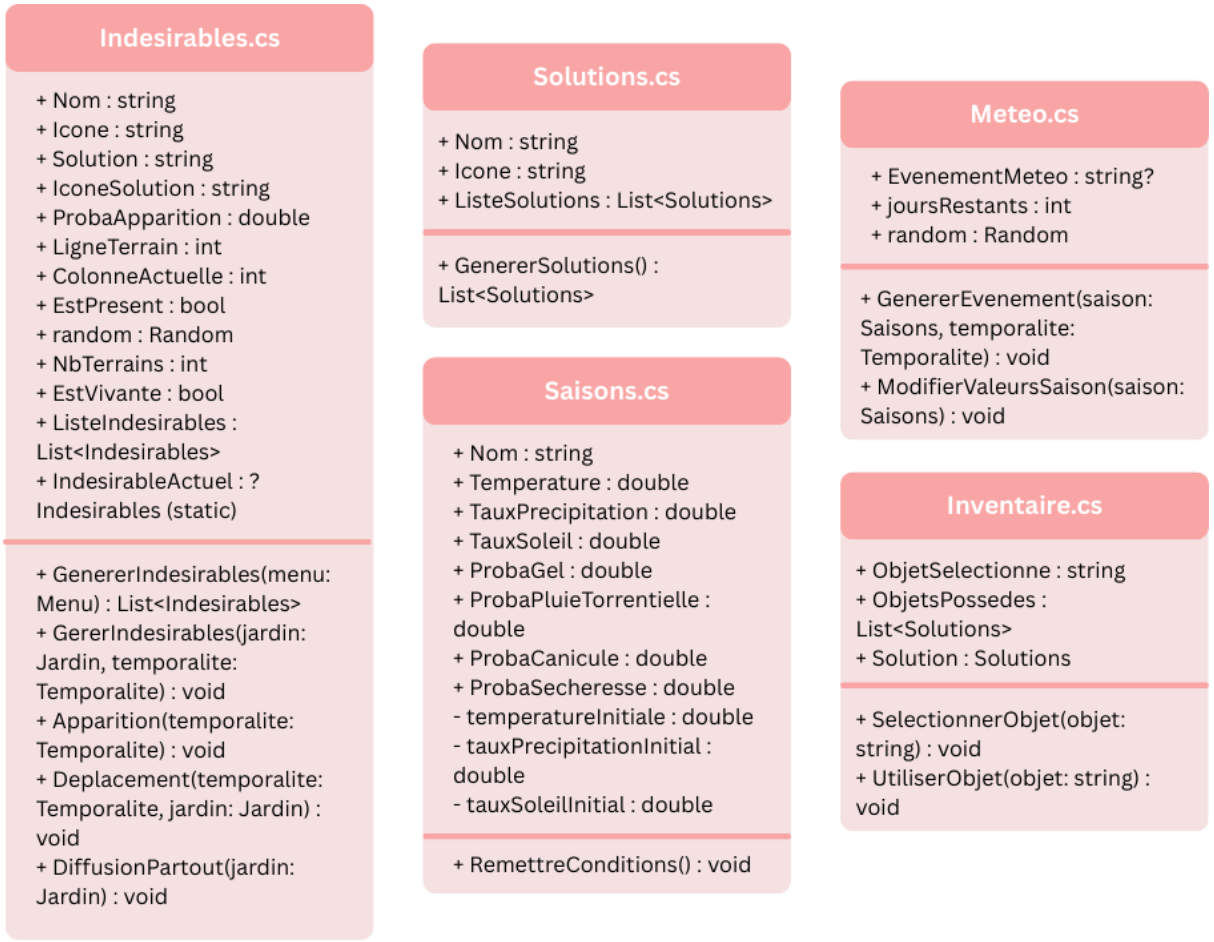


- **Plante.cs** : classe abstraite qui permet de définir les caractéristiques des plantes. Cette classe a 7 classes filles :
    - **ChampiHallucinogene.cs** : qui modélise un champignon hallucinogène.
    - **Coca.cs** : qui modélise un plant de coca.

- **Opium.cs** : qui modélise un plant d'opium.
- **Hachich.cs** : qui modélise un plant de haschich.
- **Khat.cs** : qui modélise un plant de khat.
- **Salvia.cs** : qui modélise un plant de salvia.
- **PlanteVivace.cs** : qui modélise des plantes vivaces, donc qui réapparaissent d'une année sur l'autre.



- **Saisons.cs** : classe qui permet de décrire les caractéristiques climatiques d'une saison.
- **Meteo.cs** : classe qui permet de changer la météo actuelle du jeu.
- **Indesirables.cs** : classe qui permet de gérer les indésirables (intrus et maladies), leur apparition et leur mouvement.
- **Solutions.cs** : classe qui permet de gérer les solutions face aux indésirables, pour les bloquer.
- **Inventaire.cs** : classe qui a au final été abandonnée puisque pas optimale pour le stockage et l'affichage de solutions.



IV. TESTS DE FONCTIONNEMENT RÉALISÉS

1. *Les tests liés aux plantes*

- **Plantation**

Afin de tester la bonne plantation des plantes, nous avons testé de planter chaque type de plantes dans chaque type de terrains. Cette partie était fonctionnelle. Cependant, nous avons rencontré un problème avec la plantation sur une parcelle sur laquelle une plante était auparavant morte. En effet, il était considéré qu'une plante était toujours présente sur cette parcelle, et ses informations étaient toujours fournies à l'utilisateur. Pour résoudre ce problème, nous avons ajouté une condition. En effet, initialement, il était possible de planter de nouveau si la plante sur cette parcelle avait pour valeur null. Nous avons donc ajouté le fait que Plante.EstVivante soit faux dans cette boucle. Cela a ainsi permis de replanter sur une parcelle auparavant utilisée.

- **Croissance**

Afin de tester la croissance des plantes, nous avons, après avoir codé une plante uniquement (Hachich.cs), appelé cette plante dans le Main(). Ceci nous a permis de visualiser la plante dès le début de la partie, sans implémentation de la méthode de sélection de Parcelle permettant de planter sur un terrain voulu.

Les tests ont inclus toutes les étapes de croissance des plantes, en vérifiant que chaque Afficher() retournait l'emoji approprié selon la CroissanceActuelle associé à la plante.

Nous avons donc eu un affichage de la plante, et à partir de cela nous avons modifié l'affichage de la plante (en ajoutant des espaces pour avoir un espacement constant des Parcelles). Ensuite des simulations de plusieurs tours (sur plusieurs saisons) ont été effectuées afin d'observer l'évolution des plantes.

Ensuite nous avons ajouté des ConsoleWriteLine lorsque des erreurs s'affichaient. Puis nous avons, par essai-erreur, corrigé les éventuels problèmes de vitesse de croissance (nos plantes grandissent trop vite).

Une fois les bugs résolus sur une plante, nous avons implémenté les différentes autres plantes (Salvia.cs, Coca.cs, Opium.cs, ChampiHallucinogene.cs).

- **Repousse automatique**

Bien que cette fonctionnalité n'ait pas pu être implémentée complètement, nous avons tenté de la réaliser et avons donc codé des tests de fonctionnement pour celle-ci. Pour cela nous avons, après implémentation de la Classe PlanteVivace, repris le fonctionnement des tests mis en place pour vérifier la croissance.

En effet nous avons planté dès le début du jeu des plantes qui devaient pousser petit-à-petit selon le temps, puis après leur mort, revenir à la vie un an après leur plantation.

Nous avons donc testé l'affichage des dites plantes, puis mis des Console.WriteLine pour vérifier le statut du booléen "EstEnVie", qui a permis de vérifier également que les plantes entrent en dormance à la fin de leur cycle.
Malheureusement ces tests n'ont pas pu être menés jusqu'à la correction totale des erreurs et incohérence de code, et ainsi la classe Plante Vivace n'est pas entièrement fonctionnelle.

2. Les tests de météo et temporalité

Pour la temporalité, le test principal à réaliser était de vérifier que le temps passait bien plus lentement lorsqu'un événement urgent (météorologique ou indésirable) était présent. Il était également essentiel de vérifier que le passage en mode classique avait bien lieu lors de la disparition de cet événement. Une variable booléenne EnUrgence a dû être ajoutée pour permettre le déclenchement d'une temporalité précise en fonction de sa valeur.

Concernant la météo, il a été nécessaire de vérifier que les valeurs de précipitations, d'ensoleillement et de température pour une saison donnée étaient bien modifiées lors de l'événement météorologique. Pour cela, un affichage de ces valeurs a été effectué. Cela nous a permis de confirmer la modification temporaire (avec retour aux valeurs de base des saisons) lors d'un événement météorologique.

3. Test affichage et interaction

Après avoir relié les différentes branches (Plantes, Météo/Temporalité et Terrain), ce qui nous a pris plusieurs jours afin que tout soit relié et que les variables soient bien récupérées dans chaque méthode de chaque classe. Nous avons dû modifier l'accessibilité de certains attributs afin de les appeler dans les méthodes d'autres classes. Ensuite nous avons effectué des tests pour que tout fonctionne.

La grille de plantation a été testée pour chaque colonne (de 0 à 5) afin d'assurer que la totalité de la grille était cultivable. Est apparue une erreur d'index dans le code de Jardin.cs qui a été mise en évidence par l'affiche, puisque les plantes de la sixième colonne ne s'affichent pas et n'avais pas de message "{plante} planté dans le terrain {terrain} à la colonne {colonne}. Après correction, nos plantes étaient liées correctement au Terrain.

V. UTILISATION DE L'IA SUR LE PROJET

L'intelligence artificielle (via ChatGPT et Copilot) a été utilisée à plusieurs moment du projet afin de nous épauler, notamment pour accompagner la résolution de bugs et certaines décisions esthétiques.

- **Aide au design visuel de l'application**

L'IA a été sollicitée pour choisir les couleurs de fond des cases selon les types de terrain (terre, sable, argile), dans une logique à la fois fonctionnelle et esthétique (contraste et lisibilité).

Également, afin de disposer des éléments dans la grille console, et ajuster l'affichage des terrains, qui étaient initialement prévu en carré de terrain et non en ligne de terrain. L'AI nous a permis de nous rendre compte de la complexité de la chose selon notre implémentation des Terrains, et ainsi nous sommes passés à un affichage en ligne des terrains, plus simple pour nous.

- **Suggestion et normalisation d'émojis**

Afin d'optimiser notre temps, l'AI nous a également été utile lors de la sélection d'émojis pour représenter la croissance de nos plantes, et créer une pousse logique et esthétique de notre Jardin.

- **Génération de l'ASCII**

Le titre du jeu, affiché en grand dans la console à l'ouverture, a été généré à l'aide d'IA à partir d'un prompt créatif écrivant le titre du projet d'une manière esthétique.

- **Vérification et débogage**

L'IA a été plus largement mobilisée sur ce projet pour analyser les messages d'erreurs complexes et peu lisibles de C# (ex : erreurs de typage, dépassement de tableau, méthode non accessible…). Notamment lors de la liaisons des classes codées séparément, afin de clarifier les erreurs de syntaxes et les erreurs apparaissant dans la console.

VI. ORGANISATION GÉNÉRALE DE L'ÉQUIPE

1. Organisation du travail d'équipe

Pour réaliser ce projet, nous avons travaillé en trinôme sur VSCode en collaborant sur un repository GitHub commun. La possibilité d'être en trinôme nous a permis de coder davantage de classes. Cependant, un temps plus important nous a été nécessaire pour lier nos différentes classes entre elles, puisque trois façons de coder différentes.

Nous avons mis en place des règles au niveau du GitHub en début de projet pour avoir un suivi plus clair des ajouts de chacune. En effet, le message des commits était normalisé sous ce format : "NOM. nom_branch (v1) : Description_des_modifications". La communication a été essentielle entre nous trois pour permettre de réaliser des merges entre nos branches de manière optimale et stratégique.

Pour fluidifier le processus de merge, en réduisant les possibles conflits, nous avons créé un total de 5 branches : PlantesLou, PlantesLoubis, TerrainsLaura, TempoMeteoGaelle et interface-affichage. Cela nous a permis de travailler en simultané sur des branches différentes et des fonctionnalités diverses de ce jeu.

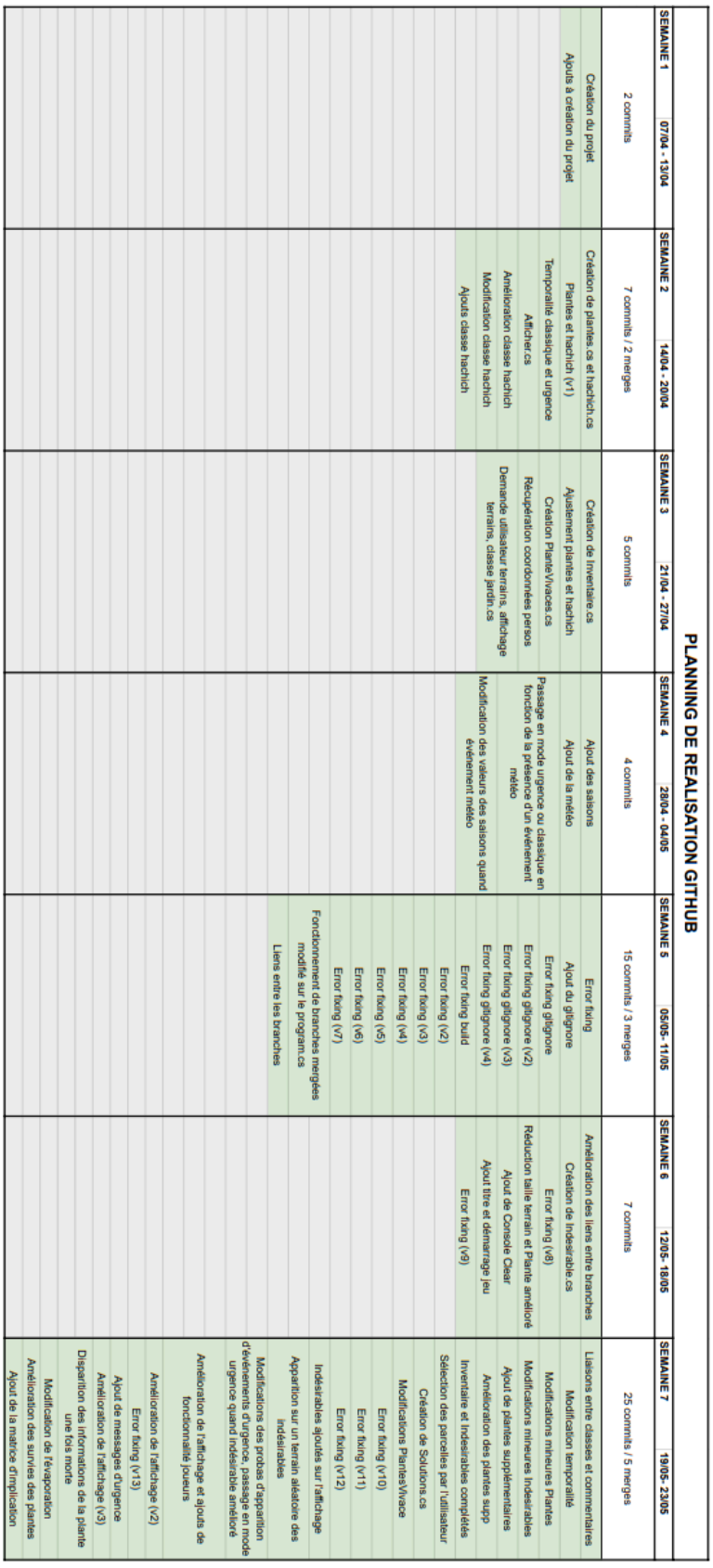
2. Matrice d'implication

Pour la réalisation du projet ENSEMENCE, il est possible de voir la répartition des tâches entre nous trois sur la matrice d'implication ci-dessous. Selon nous, elle représente avec justesse le travail réalisé par chacune et divisé équitablement entre les différents membres.

MATRICE IMPLICATION				
PROJET	UE: CO6SFPA0 1A Programmation Avancée **ENSEMENCE: POTAGER MAGIQUE**	VERSION **FINALE**	**23 MAI 2025**	
TÂCHES		ÉTUDIANTES		
Numéro	**LIBELLÉ**	**Lou PÉPIN POUSSARD**	**Laura DELBREIL**	**Gaëlle L'HERMITE**
1	Choix du Sujet du potager	34%	33%	33%
2	Établissement de la logique du projet	33%	34%	33%
3	Codage Plantes	90%	5%	5%
4	Codage Terrain	5%	90%	5%
5	Codage Météo et Temporalité	5%	5%	90%
6	Codage Intrus	0%	35%	65%
7	Merge des branches Principale et liaisons des classes	80%	10%	10%
8	Commentaires du Code	33%	33%	34%
9	Page Acceuil et menu du jeu	7%	83%	10%
10	Esthétique du projet	25%	50%	25%
11	Débogage (gestions des erreurs)	33%	33%	34%
12	Rapport	30%	20%	50%
13	Matrice Implication	50%	25%	25%
	TOTAL	33%	35,%	32%

3. Planning de réalisation GitHub

Le planning suivant représente les commits et merges réalisés chaque semaine.



VII. BILAN

Pour conclure ce rapport sur notre projet de Programmation Avancée **ENSEMENCE**, ce projet nous a apporté beaucoup de connaissances et compétences en programmation orientée objet. En effet, il nous a permis de nous familiariser encore davantage avec cette façon de programmer en l'appliquant à un projet sur le long terme. Nous notons également une amélioration du travail d'équipe comparé au semestre précédent, et ce même en étant un trinôme. Nous avons en effet eu très peu de conflits à résoudre lors de merge, avec une séparation plus nette entre nous.

Nous sommes fières du jeu de simulation de jardin que nous rendons ce jour, puisque nous avons réussi à implémenter la grande majorité des fonctionnalités voulues initialement. Au-delà des fonctionnalités requises, nous avons ajouté de nombreuses fonctionnalités bonus, telles que la disparition de l'ensemble des plantes lorsque l'indésirable n'est pas arrêté sur six parcelles. L'affichage est aussi intuitif pour le joueur et lui permet de réaliser de nombreuses actions.

Pour valider ce programme, nous avons réalisé de nombreux tests, pour vérifier la fluidité d'utilisation de notre jeu. Nous pensons donc proposer un jeu à la fois agréable et divertissant, qui plaira au plus grand nombre.