

Programmation avancée C#

Projet : Création d'un jeu en Programmation Orientée Objet (POO)

ENSeménC

DUFOND-GONZALEZ Lucia - Groupe 4

MORVAN Maureen - Groupe 4

Du 01/04/2025 au 23/05/2025

SOMMAIRE

I. Présentation du jeu	3
A. Description des mondes	3
B. Les plantes	4
C. Les animaux	5
D. Les terrains	5
E. Saison & Météo	6
II. Exécution du jeu	8
A. Configuration des paramètres	8
B. Déroulement d'une partie	8
C. Choix des actions	10
III. Choix de modélisation des données	11
IV. Tests réalisés	13
V. Gestion de projet	14
Bilan critique	15

I. Présentation du jeu

Dans le cadre de ce projet, nous avons programmé un simulateur de potager, dans lequel le joueur cultive un potager. Plutôt que de sélectionner un pays précis rendant le jeu très réaliste, nous avons choisi de créer des mondes fictifs avec des ambiances colorées et immersives.

A. Description des mondes

Cette version du jeu propose deux univers aux ambiances bien distinctes, chacun avec ses terrains, ses plantes et ses animaux caractéristiques.

- La terre brûlée : Un environnement aride, désertique avec des terrains sableux et terreux. Concernant les plantes, on y retrouve des tulipes, des roses, des fraises et des cerises. Enfin, faites attention aux renards qui se feront un plaisir de manger les fruits.
- La forêt enchantée : Un monde plus humide et végétal dans lequel les terrains sont boisés et humides. Des plantes résilientes comme le sapin et le noisetier y sont présentes, ainsi que des trèfles et des rhododendrons. Mais gare aux écureuils qui grignotent les cultures.

Pour favoriser l'immersion des joueurs, des emojis ont été utilisés pour le visuel de la grille. Ainsi, l'expérience de jeu est différente selon le monde choisi par le joueur (cf. Figure 1 et 2).

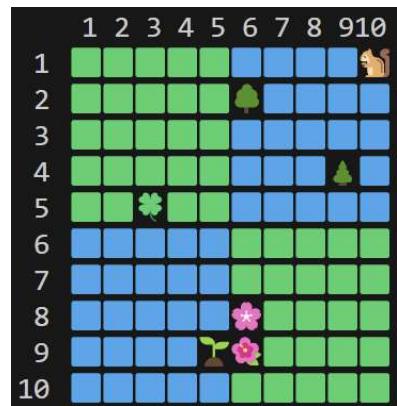
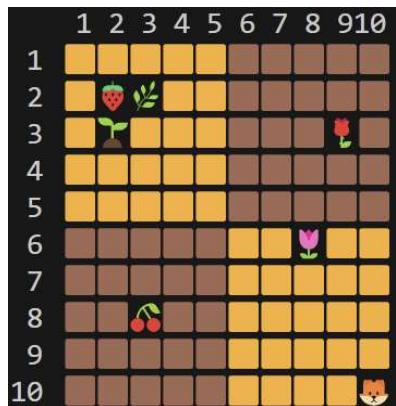


Figure 1 : Illustration de la grille du monde

“La Terre Brûlée”

Figure 2 : Illustration de la grille du monde

“La Forêt Enchantée”

Comme expliqué précédemment, chaque monde possède ses propres terrains, ses plantes et ses animaux. Cependant il possède tous la même météo qui s'adapte en fonction des saisons.

B. Les plantes

Dans notre jeu, chaque plante est représentée par un objet possédant un ensemble riche de caractéristiques, influençant son comportement et sa survie au fil des jours. Elle est positionnée dans la grille via ses coordonnées (x, y), et traverse quatre étapes de croissance, chacune illustrée par un emoji évolutif permettant de visualiser son état (graine, jeune pousse, adulte, plante fanée).

Les plantes peuvent tomber malades, ce qui ralenti leur croissance et nécessite au joueur de les soigner s'il veut, plus tard, récupérer les récoltes. Elles disposent également d'une espérance de vie mesurée en plusieurs cycles de croissance, au terme de laquelle elles meurent.

Chaque espèce végétale a des besoins spécifiques en :

- Quantité d'eau (humidité du terrain),
- Luminosité,
- Température (liée à la météo et à la saison),
- Fertilité du terrain,
- Et une préférence de type de terrain (sableux, humide, boisé...).

Tous les jours, si au moins 50% de leurs besoins sont respectés (prise en compte de la l'aspect maladie donc 6 conditions), alors les plantes poussent. Dans le cas contraire, leur croissance stagne.

Enfin, chaque monde contient une plante envahissante : le rhododendron dans la forêt enchantée, et la fraise dans la terre brûlée. A chaque tour, elles se propagent aléatoirement

sur les cases voisines. Ainsi, le joueur doit les surveiller de près, car elles peuvent rapidement envahir le potager.

C. Les animaux

Les animaux apparaissent aléatoirement dans le potager (avec une certaine probabilité qui change selon la saison) et uniquement par l'un des quatre coins de la grille. Une fois présents, ils se déplacent à chaque tour de manière aléatoire, selon une direction choisie parmi les quatre points cardinaux, à condition que la case cible ne soit pas une tranchée, qu'ils ne peuvent pas traverser.

Lorsqu'un animal rencontre une plante sur son chemin, deux cas de figure se présentent :

- Si la plante rencontrée est sa préférée (définie par l'attribut plantePreferee), l'animal mange ses fruits.
- Dans le cas contraire, il arrache la plante, la faisant immédiatement mourir, ce qui représente une perte pour le joueur.

Pour défendre son potager, le joueur peut faire fuir les animaux (mode classique) en défendant une certaine zone (carré de 9 cases avec pour centre celle sélectionnée par le joueur. Ou il peut installer des épouvantails (mode urgence), une protection efficace et ayant des bénéfices sur le long terme. En effet, l'épouvantail reste jusqu'à la fin de la partie et si un animal détecte un épouvantail à moins de deux cases de distance, il fuit immédiatement le potager, empêchant ainsi tout dégât supplémentaire.

Cette gestion des animaux permet de rendre le jeu plus dynamique et réaliste, obligeant le joueur à anticiper leurs mouvements pour limiter les pertes de récoltes.

D. Les terrains

Les terrains constituent la base de chaque case du potager, et possèdent plusieurs caractéristiques essentielles qui influencent directement la croissance des plantes. Chaque terrain est défini par trois paramètres principaux (tous définis comme en pourcentage) :

- Un taux d'humidité,
- Un niveau de luminosité,
- Un taux de fertilité.

Ces propriétés évoluent quotidiennement en fonction des conditions climatiques, influencées par la météo et la saison. Par exemple, une pluie abondante augmente naturellement l'humidité des terrains et diminue leur luminosité. Tandis qu'en cas de fort ensoleillement c'est l'inverse, l'humidité diminue et la luminosité augmente.

Chaque type de terrain (sableux, terreux, boisé, humide...) offre un environnement plus ou moins adapté aux différentes plantes. Certaines espèces poussent mieux sur des terrains riches, d'autres préfèrent les sols secs ou à faible exposition lumineuse.

Lors de l'ajout du mode d'urgence, nous avons choisi de modéliser les éléments, tels que les épouvantails et les tranchées, comme des terrains spéciaux pour des raisons de praticité et de cohérence. En effet, chaque case de la grille étant déjà représentée par un objet Terrain, il était naturel d'étendre cette structure plutôt que d'ajouter une classe supplémentaire.

Par ailleurs, lorsqu'un terrain contient un élément installé par le joueur (tranchée et épouvantail) il n'a plus vocation à accueillir de plante. Dans ce cas, ses valeurs nutritives ne sont plus prises en compte, simplifiant ainsi la logique du jeu tout en assurant un comportement cohérent de la grille.

E. Saison & Météo

La météo est différente selon les saisons, les deux classes associées sont donc liées. L'objet meteo de la classe Meteo est ainsi créé et modifié directement dans la classe Saison. Lors de la simulation, on crée l'objet saison de classe Saison, qui vient donc ajouter la présence de saisons dans notre monde. La saison change au cours de la partie. Afin de rendre la partie jouable, la saison change tous les quatre jours (donc le temps de quatre

actions), grâce à un static int temps, qui vient comptabiliser le nombre de jours passés, et qui revient à 0 au bout de 12 actions (afin de revenir à la saison du printemps).

Selon la saison, la météo diffère, avec des probabilités de pleuvoir différentes, ainsi que des encadrements de valeurs différents pour les paramètres tels que la luminosité et le niveau de vent, qui peuvent affecter les plantes qui se trouvent dans le monde. La température est également déterminée aléatoirement selon la saison dans les classes dérivées de Meteo.

La classe Meteo est une classe abstraite qui regroupe les éléments de base pour toutes les météos selon les saisons. Elle gère les conditions météorologiques du monde au travers des paramètres temperature et niveauVent. Elle permet aussi de modifier certains paramètres des terrains, qui sont le taux d'humidité et le taux de luminosité. Le booleen estEnTrainDePleuvoir permet d'adapter le visuel à afficher dans Simulation selon la présence ou non de pluie afin d'informer le joueur.

La méthode AfficherHumiditeTerrain() permet d'afficher les informations sur les terrains, dont le taux d'humidité, afin que le joueur puisse voir les changements.

AfficherEvenement() est une méthode virtuelle qui permet d'afficher des messages liés aux événements climatiques, qui sont différents selon la saison.

La classe Meteo contient la méthode Pleuvoir(), qui détermine si la pluie tombe en fonction de probaPleuvoir (déterminé dans les classes dérivées de Meteo). S'il pleut, le booleen estEnTrainDePleuvoir devient true. Pour chaque terrain, l'humidité augmente de 10 (sans pouvoir dépasser 100), et réduit la luminosité de 10 (sans pouvoir aller dans les négatifs). S'il n'y a pas de pluie, le static int nombreJoursSansPluie est incrémenté, et la luminosité des terrains augmente de 20 (sans pouvoir dépasser 100). Si cela fait plus de trois jours qu'il n'y a pas eu de pluie (`nombreJoursSansPluie > 3`), alors le taux d'humidité de tous les terrains diminue de 10.

La fonction DeterminerCatastropheEtVariables() détermine la probabilité d'une catastrophe naturelle à l'aide d'un booleen catastrophe, qui est de 1 chance sur 3 en mode difficile et de 1 chance sur 6 sinon. Si une catastrophe est déclenchée, alors le mode urgence est activé à

l'aide du booléen modeUrgence (qui se trouve dans Simulation). Après cette distinction de cas, la méthode virtuelle DeterminerVariables() est appelée.

Les classes dérivées de Meteo (MeteoPrintemps, MeteoEte, MeteoAutomne et MeteoHiver) reprennent les méthodes DeterminerVariables et AfficherEvenement afin de les adapter à la saison. Cela permet d'avoir des probabilités de pleuvoir différentes, ainsi que des encadrements différents pour les paramètres propres à la météo (température et niveau de vent). Les paramètres dépendent également de la présence ou non d'une catastrophe naturelle (selon la valeur du booléen catastrophe défini dans la classe abstraite meteo).

II. Exécution du jeu

A. Configuration des paramètres

Dès le lancement du programme, les règles du jeu s'affiche et le joueur est invité à configurer les paramètres de sa partie. Il commence par choisir l'univers dans lequel il souhaite évoluer : la terre brûlée ou la forêt enchantée. Il définit ensuite la taille du potager (hauteur et largeur de la grille), puis sélectionne le nombre de jours que durera la simulation et le mode de jeu (facile ou difficile). Ces réglages personnalisables permettent d'adapter la difficulté et la durée de la partie selon les préférences du joueur. Une fois les paramètres définis, la boucle de jeu s'enclenche automatiquement pour simuler le déroulement des jours.

B. Déroulement d'une partie

Chaque jour correspond à un tour de jeu et suit un déroulement précis. En début de journée, une météo est générée aléatoirement en fonction de la saison en cours (printemps, été, automne, hiver). Cette météo influence les paramètres des terrains comme l'humidité, la luminosité ou la fertilité, et peut déclencher des événements extrêmes (sécheresse, tempête, gel) dans les modes les plus avancés, activant alors un mode urgence.

L'interface du potager est mise à jour, affichant l'état actuel de la grille ainsi que les conditions météorologiques du jour (cf. Figure 3). Le joueur peut ensuite choisir une action à effectuer.

Selon son choix, les plantes, les terrains ou les animaux sont modifiés, et un message de retour informe le joueur du succès ou de l'échec de son action. Certains cas sont anticipés pour proposer automatiquement une alternative. Par exemple, le joueur ne peut pas semer sur un terrain qui possède déjà une plante. Or, d'autres cas n'ont pas été contrôlés laissant au joueur la possibilité d'effectuer une action jugée comme inutile pour l'évolution du potager, libre à lui de gérer son potager comme il le souhaite.

À la fin de chaque tour, toutes les plantes tentent de croître si au moins 50 % de leurs besoins sont remplis. Il existe aussi une probabilité de tomber malade, ajustée selon la saison. Ensuite, les plantes envahissantes (comme la fraise ou le rhododendron) se propagent automatiquement, et les animaux se déplacent de manière aléatoire. Avant de passer au jour suivant, une dernière probabilité permet de déterminer si un nouvel animal apparaît dans le potager.

Une fois le nombre de jours écoulé, un inventaire final des récoltes est affiché, suivi d'un écran de fin de jeu remerciant la personne d'avoir joué à ce jeu.



Figure 3 - Illustration de l'interface du jeu

C. Choix des actions

À chaque tour, le joueur peut effectuer une action parmi une liste qui dépend du mode de jeu du tour. Il y a deux modes : Le mode classique et le mode urgence. Ce dernier est activé lorsqu'il y a des catastrophes naturelles et aussi lorsque le nombre d'animaux ou de plantes envahissantes est supérieur à 10 mettant ainsi le potager en danger.

Dans le mode classique les actions proposées sont les suivantes :

- Semer une plante,
- Arroser des terrains,
- Déposer de l'engrais,
- Désherber,
- Traiter une plante malade,
- Faire fuir un animal,
- Récolter les fruits d'une plante (si elle arrive à maturité).

Chacune des actions précédentes entraînent des modifications sur les plantes, les terrains ou les animaux. Cependant, le joueur a également la possibilité de "Passer la journée" s'il ne souhaite rien faire sur son potager ou "Quitter la partie" s'il veut arrêter de jouer.

Dans le mode urgence, des actions supplémentaires sont ajoutées à la liste précédente afin de permettre au joueur de gérer les nuisibles et lutter contre les catastrophes météorologiques :

- Installer un épouvantail, pour effrayer les animaux.
- Creuser une tranchée, pour ralentir la propagation des plantes envahissantes.

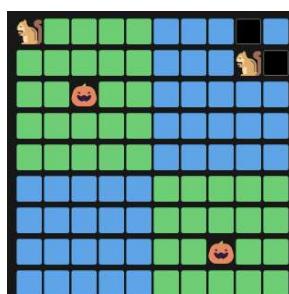


Figure 4 : Illustration d'une grille comportant des épouvantails et des tranchées.

III. Choix de modélisation des données

Pour structurer notre projet de manière claire, évolutive et cohérente avec les principes de la programmation orientée objet, nous avons choisi de modéliser le jeu autour de plusieurs grandes classes reflétant les entités essentielles de notre simulation : Plante, Terrain, Animal, Monde, Meteo, Saison, Simulation, PlanteEnvahissante et Visuel (cf. Figure 5).

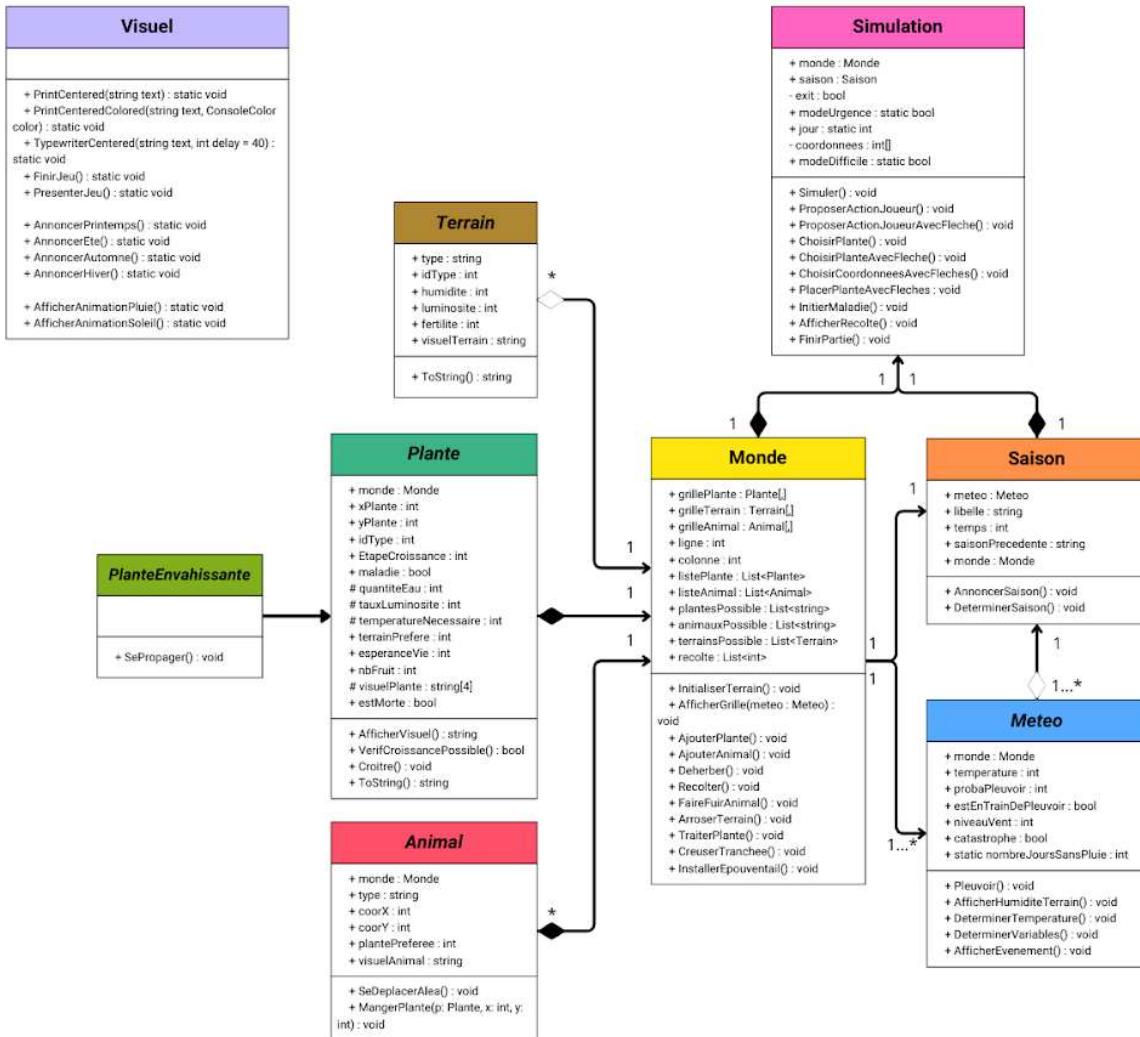


Figure 5 : Illustration du diagramme UML du jeu

La classe **Simulation** pilote la logique principale du jeu. Elle contient la boucle qui gère chaque tour (donc chaque jour), enchaînant les événements dans l'ordre suivant : génération de la météo, affichage de la grille, proposition d'action au joueur, mise à jour des éléments (croissance des plantes, déplacement des animaux, apparition aléatoire

d'événements), puis fin de journée. Elle centralise les interactions entre le joueur et l'univers de jeu.

La classe **Monde** représente la grille du potager, composée de plusieurs matrices (grilleTerrain, grillePlante et grilleAnimal) stockant l'état du potager au cours de son évolution. Chaque case de la grille contient un terrain, et peut éventuellement accueillir une plante ou un animal. Cette séparation nous a permis de gérer finement les interactions entre les différentes entités (ex. un animal mangeant une plante ou encore le fait qu'une plante grandisse plus ou moins vite selon le terrain sur lequel elle est). Le monde contient également les listes dynamiques de toutes les plantes et animaux présents dans le potager, ce qui nous a permis d'ajouter et de supprimer facilement des entités au cours du jeu en parcourant seulement les listes souhaitées.

Ensuite, nous avons choisi de faire des classes distinctes pour chaque élément du jeu. Ainsi, toutes les plantes héritent d'une classe abstraite **Plante**, définissant leurs caractéristiques communes (coordonnées, croissance, maladie, espérance de vie, etc...). Chaque type de plante (Tulipe, Rose, Cerise, Trefle, Sapin, Noisetier) est une sous-classe spécifique. Nous avons également introduit la classe **PlanteEnvahissante** (dont font partie Rhododendron et Fraise), qui hérite de Plante mais surcharge une méthode pour permettre la propagation automatique d'une plante sur les cases voisines.

Créer une classe **Animal**, nous permet de facilement modéliser les créatures pouvant apparaître dans le potager. Nous n'avons pas trouvé nécessaire de faire de classes dérivées étant donné que nous n'avons introduit qu'un type d'animal par monde, or cela pourrait être intéressant d'en créer d'autres avec des déplacements et des comportements spécifiques selon leur nature (insectes, oiseaux, mammifères...).

Puis, la classe **Terrain**, nous a permis de définir les caractéristiques environnementales pour chaque case du terrain indépendamment. De plus, chaque terrain était identifié par un idType, utilisé pour stocker le type de terrain préféré de chaque plante.

Pour gérer la temporalité du jeu nous avons créé les classes **Meteo** et **Saison**. La météo gère l'évolution du climat quotidien (humidité dans l'air, luminosité, vent, température) et les risques de catastrophe (sécheresse, tempête, gel ...). Tandis que la saison, qui encapsule la météo, s'occupe de gérer les météos spécifiques à chaque saison et permet de moduler les probabilités liées aux catastrophes. C'est notamment à ce moment précis qu'entre en jeu le mode *Difficile* (que le joueur a eu la possibilité de sélectionner au début) car il permet de modifier ces probabilités à la hausse. En effet, dans ce mode, le joueur devra faire face à plus de catastrophe météorologique que dans le mode *Facile*.

Enfin, nous avons rassembler l'ensemble des fonctions d'affichage (texte centré, machine à écrire, animations météo et visuels saisonniers) dans une classe **Visuel**, afin d'améliorer l'interface et de rendre le jeu plus agréable. Ce choix d'externaliser une partie de l'affiche nous a permis de séparer le code à proprement lié au jeu de l'interface utilisateur, le rendant ainsi plus lisible et réutilisable.

En résumé, notre modélisation repose sur une architecture orientée objet cohérente et extensible, facilitant l'ajout de nouveaux types de plantes, de mondes ou d'interactions. L'usage des héritages, des grilles 2D, et la séparation des rôles entre classes a permis de construire un jeu riche, fonctionnel et facile à faire évoluer.

IV. Tests réalisés

Tout au long du développement du projet, nous avons réalisé de nombreux tests manuels pour vérifier que chaque nouvelle fonctionnalité fonctionnait comme prévu. Ces tests ont été effectués de manière itérative, à chaque ajout de méthode ou de logique de jeu, afin d'assurer la cohérence et la stabilité du système global.

Ces vérifications nous ont permis de corriger rapidement les erreurs et de mieux comprendre le comportement de certaines entités (notamment les plantes et les animaux) en situation réelle. L'ensemble du code étant accessible, vous pourrez vous référer aux blocs concernés (à la fin de Program.cs) pour observer la logique de test utilisée.

Voici quelques exemples concrets de tests effectués :

- Vérification de la croissance des plantes selon leurs conditions (croissance uniquement si 50 % des besoins sont respectés),
- Contrôle de la propagation correcte des plantes envahissantes (fraise, rhododendron) dans les cases voisines libres,
- Tests sur la suppression automatique des plantes mortes, après l'épuisement de leur espérance de vie,
- Simulation des déplacements aléatoires des animaux et observation des interactions avec les plantes (fruits mangés ou plantes détruites),
- Vérification du fonctionnement de l'épouvantail : les animaux fuient bien s'il se trouve à deux cases ou moins,
- Contrôle des modifications météorologiques selon les saisons, et de leur effet sur les terrains.

Des tests d'interface ont également été réalisés, notamment sur le système de sélection par flèches, afin de garantir une navigation fluide sur la grille. Ainsi, ces tests réguliers nous ont permis d'identifier les erreurs et problèmes inattendus, de les corriger rapidement, et d'aboutir à une version jouable.

Nous avons ajouté une classe abstraite Tests dans notre code, avec différentes fonctions qui reprennent certains codes de nos tests effectués pour vérifier les erreurs.

Lorsque nous effectuions les tests, cette classe et ces fonctions n'existaient pas, il s'agit simplement d'un ajout pour montrer à quoi ont pu ressembler nos tests.

Ces fonctions ont été ajoutées à la suite du code de [Programs.cs](#) en commentaire. Les tests peuvent ainsi être relancés si besoin (en veillant à mettre en commentaire la fonction LancerJeu()).

V. Gestion de projet

Avant de commencer à coder, nous avons pris le temps de planifier et de réfléchir aux classes essentielles à implémenter, ainsi qu'aux fonctionnalités souhaitées. Cette phase de

conception nous a permis de poser des bases solides avant de commencer à coder. Nous nous étions fixées une première version du jeu, centrée uniquement sur les fonctionnalités de base, afin d'assurer une structure stable et fonctionnelle du jeu. Et grâce à notre avancement régulier, nous avons pu ajouter des fonctionnalités supplémentaires, couvrant ainsi l'ensemble des aspects demandés dans le sujet. En effet, en nous consacrant chaque semaine à l'avancement du jeu, que ce soit lors des séances encadrées ou en autonomie à la maison, nous avons réussi à rendre une version du jeu propre et fonctionnelle.

Bien que la répartition exacte des tâches est présentée dans la matrice d'implication, voici tout de même une explication concise de celle-ci. Chacune de nous s'est naturellement spécialisée dans différentes parties du jeu, en fonction de nos affinités :

- Lucia s'est spécialisée dans les parties relatives à la météo, aux saisons et au visuel.
- Maureen s'est concentrée sur les mécaniques liées au monde, aux terrains, aux plantes et aux animaux.

Cependant, certaines parties transversales comme l'interface utilisateur et la classe Simulation ont été réalisées en collaboration. Cela nous a permis de mutualiser nos compétences et de nous approprier entièrement le fonctionnement du jeu, notamment avec l'enchaînement des appels : tirage de la météo du jour, affichage de la grille de jeu, proposition d'action au joueur...

Cette répartition des responsabilités nous a permis d'être plus efficaces tout en ayant une bonne vision globale du projet. En outre, nous avons réussi à établir une communication régulière, ce qui nous a permis de nous tenir au courant de nos avancements respectifs, des difficultés rencontrées et de nous entraider pour les résoudre. Vis-à-vis de l'utilisation de GitHub, nous avons majoritairement travaillé sur la branche principale, ayant confiance dans la qualité du code produit par l'une ou l'autre.

Bilan critique

Ce projet a été très enrichissant et globalement agréable à réaliser, bien qu'il nous ait demandé un investissement important en termes de réflexion et de conception. Nous sommes satisfaites du rendu final, même si nous avons conscience que la structure de certaines classes pourrait être optimisée pour améliorer la lisibilité et la maintenabilité du code.

Ce travail nous a permis de mettre en pratique toutes les notions abordées en cours et en TD, notamment l'héritage, le polymorphisme, l'encapsulation et la gestion de collections. Nous avons aussi pu aller au-delà du programme, en explorant des fonctionnalités avancées comme l'utilisation de Activator, la gestion d'animations textuelles, ou encore la génération dynamique d'objets à partir de chaînes de caractères.

L'introduction d'un mode difficile fut une bonne idée, qui a permis d'ajouter une couche de complexité et de stratégie. Néanmoins, avec plus de temps nous aurions aimé la développer davantage, ainsi que l'ensemble du jeu.

Voici quelques pistes d'améliorations envisagées :

- Créer de nouveaux mondes, que nous avions déjà imaginés, afin d'élargir les environnements du jeu,
- Ajouter des animaux supplémentaires dans chaque monde, en changeant leur plantePreferee. Cela aurait rendu le jeu encore plus intéressant.
- Introduire de nouvelles actions d'urgence, comme la pose de bâches, la construction d'une serre, ou encore des systèmes d'irrigation pour lutter contre les intempéries ;
- Diversifier les météos en fonction des mondes : avec, par exemple, plus de sécheresse dans "La Terre Brûlée" et de fortes inondations dans "La Forêt Enchantée".
- Améliorer l'aspect des maladies, en envisageant leur propagation par zone, ou en introduisant plusieurs types de maladies affectant certaines plantes mais pas d'autres.

Malgré ces pistes d'amélioration, nous sommes fières d'avoir conçu un jeu à la fois fonctionnel et interactif, en relevant les défis techniques et créatifs que le sujet impliquait.