ENSemenCe

Document de justification technique - Projet 2025

Cécile Tollis

Noé Peyrot

1A - Groupe 4

Pour le vendredi 23 mai 2025

Table des matières

I. Introduction

Dans le cadre de l'UE *Programmation avancée*, nous avons dû réaliser un jeu de gestion de jardin en C#. Dans ce document de justification technique, nous allons vous présenter notre réalisation de ce projet, intitulé ENSemenCe. Nous avons programmé en C# avec le framework .NET. Pour travailler en équipe tout au long du projet, nous avons utilisé GitHub pour partager nos avancées mutuelles et avancer efficacement.

Ce projet propose la conception et la programmation d'un jeu sous la forme d'un simulateur de potager. ENSemenCe plonge le joueur dans la gestion d'un jardin virtuel influencé par des conditions climatiques dynamiques. Le joueur doit y cultiver différentes plantes, gérer les aléas météorologiques, faire face à des intrusions d'animaux, et prendre les bonnes décisions au fil des semaines pour assurer la survie et la productivité de son potager.

Ce document a pour objectif de présenter le processus de développement de notre simulateur de potager. Dans un premier temps, nous détaillerons l'univers du jeu avant de décrire le déroulement typique d'une partie. Ensuite, nous expliquerons la modélisation objet que nous avons mise en place et les tests réalisés pour garantir un code opérationnel. Nous aborderons également l'organisation de l'équipe, la répartition des tâches et le planning de réalisation mis en place pour assurer l'avancement du projet. Nous consacrerons également une section à notre partie créative et aux difficultés rencontrées au cours du développement. Enfin, nous terminerons par un bilan critique pour analyser les résultats obtenus et les enseignements tirés de ce projet.

II. Univers du jeu

ENSemenCe plonge le joueur dans un monde fictif où il incarne un jardinier chargé de cultiver un jardin dans un environnement soumis à des aléas climatiques et à des événements imprévus.

Le joueur est accueilli par un message de bienvenue stylisé, qui lui propose de choisir un nom, instaurant immédiatement une forme de personnalisation et d'immersion. Il découvre ensuite qu'il dispose d'un certain nombre de semaines pour faire prospérer son jardin, un délai qui structure l'ensemble du gameplay en une boucle temporelle simple et claire. Mais ce n'est pas la seule condition : en effet, le jeu s'arrête également si le joueur n'a plus de plantes dans son inventaire et que son jardin est vide. Le jardin dans lequel évolue le joueur est divisé en trois types de terrains distincts : terreux, sableux et argileux, chacun ayant un impact sur la croissance des plantes.

L'univers météorologique du jeu contribue à renforcer l'immersion. Chaque semaine, une météo aléatoire s'affiche et influe sur l'évolution du jardin. Soleil, pluie, sécheresse, vent… Ces évènements ne sont pas purement esthétiques mais ont des effets concrets sur les plantes (croissance accélérée, ralentissement, arrosage automatique, etc.).

Enfin, pour ajouter une touche créative, nous avons introduit un mode urgence inspiré d'événements imprévus comme les intrusions d'animaux (rats) ou de créatures plus surprenantes que vous verrez dans la suite du rapport…

Cet univers, bien que simple, repose donc sur des éléments narratifs et mécaniques choisis pour favoriser une expérience de jeu dans une atmosphère détendue mais stimulante.

III. Déroulement d’une partie

Voici le déroulement classique d’une partie de notre jeu :

Lorsque l’utilisateur décide de lancer notre code, un message s’affiche petit à petit pour un effet de style et affiche “Bienvenue dans ENSemenCe”. Il demande à l’utilisateur de choisir un nom pour une expérience plus immersive. On explique ensuite au joueur qu’il dispose de 30 semaines (nous avons fixé cette durée, mais elle reste modifiable si l’on désire jouer plus longtemps) pour constituer le plus beau des jardins. S’affichent ensuite le numéro de la semaine pour donner un repère temporel et la météo de cette semaine (suivant la météo, des effets peuvent s’appliquer sur les plantes du jardin). On rappelle à l’utilisateur qu’il dispose de trois actions par tour et on lui demande s’il souhaite regarder son jardin qui est décomposé en trois terrains : un terrain de terre, un terrain de sable et un terrain d’argile.

Nous lui proposons ensuite une liste d’actions qu’il peut effectuer : semer une plante, arroser un terrain (et par extension toutes les plantes se trouvant dessus), récolter une plante mûre, voir ses récoltes, désherber et passer à la semaine suivante. S’il réalise trois actions, le passage à la semaine d’après se fera automatiquement. S’il décide de semer une plante, son inventaire de plantes s’affiche, et il peut choisir celle qu’il souhaite (il ne peut en planter qu’une à la fois). On lui demande ensuite sur quel terrain il souhaite la planter, et un message s’affiche pour lui rappeler le terrain préféré de la plante en question pour qu’il évite de la planter au mauvais endroit. Le jardin s’affiche enfin pour lui montrer où la plante se trouve, elle est représentée sur un terrain par son initiale (ou deux lettres si deux plantes commencent par la même lettre). S’il souhaite arroser une zone, on lui demande de choisir un terrain (on arrose un terrain à la fois). Un message s’affiche en disant que les plantes sur cette zone ont été arrosées et le joueur a donc accès à leur niveau d’eau (sous forme d’une fraction sur 100 pour représenter un pourcentage). La méthode Arroser n’arrose pas les plantes qui ont un niveau d’eau à 100/100 pour éviter de les noyer, nous avons choisi ce choix de codage car nous pensons qu’un jardinier vérifie les besoins en eau de ses plantes avant de les arroser. S’il décide de récolter une plante (il ne peut en récolter qu’une à la fois), le programme va lui renvoyer la liste des plantes mûres présentes sur le plateau et il pourra choisir celle qu’il souhaite récupérer. Il peut choisir de regarder ses récoltes, dans ce cas, une liste s'affiche avec l’ensemble des plantes qu’il a récoltées depuis le début de la partie. Et enfin, il peut désherber les mauvaises herbes situées sur un terrain en particulier. Bien évidemment, après chaque action, il peut passer à la semaine suivante même s’il lui reste des actions supplémentaires s’il pense avoir fini pour ce tour.

Lors de chaque semaine, la météo change de manière aléatoire, ce qui peut avoir des effets divers sur les plantes du jardin. Une petite pluie viendra gentiment arroser les plantes présentes, le soleil accélérera leur croissance tandis que des nuages la ralentiront, une grosse averse ralentira la croissance des plantes tout en les arrosant, une sécheresse viendra baisser leur niveau d’eau ainsi que leur croissance, une tornade viendra perturber leur croissance tout en présentant le risque de les détruire, et un petit vent pourra intervertir

de manière aléatoire deux plantes présentes dans le jardin, avec le risque qu'elles se retrouvent sur un terrain inadapté à leur bon développement.

Chaque semaine, l'utilisateur a accès aux caractéristiques de ses plantes pour en suivre l'évolution (leur état de santé et leur niveau d'eau). Pour suivre de manière plus intéressante le développement des plantes, il peut également voir sa plante grandir tour par tour. Pour simplifier le code, il peut la voir en dehors du plateau, comme si on zoomait sur la plante pour l'observer. Sur le plateau, la plante reste représentée par sa lettre.

Pour que l'utilisateur ait un choix suffisamment large, nous avons créé dix types de plantes différentes pour qu'il puisse avoir un éventail de choix lors de la gestion de son jardin. Chaque plante a un terrain préféré auquel il devra prêter attention s'il souhaite que ses plantes poussent correctement. Il pourra planter (au choix) du blé, des cerises, des champignons, des endives, des lotus, des pissenlits, des roses, des sapins, des tournesols et des tulipes. Nous avons défini ces plantes pour plusieurs raisons : tout d'abord, nous voulions alterner entre des plantes comestibles, comme les fruits et les légumes, et des plantes "décoratives", comme les lotus et les roses. Pour respecter cela, nous avons cherché quelles plantes étaient les plus simples à réaliser en art ASCII. Nous avons été confrontés à la difficulté de cet art numérique, car il est difficile de représenter de manière simple des entités réelles. Nous avons donc opté pour les plantes qui avaient une représentation ASCII la plus simple possible tout en restant reconnaissables. Et nous avons ajouté un autre type particulier de végétation : les mauvaises herbes. Une mauvaise herbe apparaît sur une case du terrain tous les deux tours et empêche de semer une plante sur cette case.

Et ce fonctionnement se répète lors des 30 tours fixés avant que la partie ne s'arrête. Il est possible de modifier ce nombre comme précisé précédemment si vous souhaitez jouer plus longtemps car nous ne nous sommes pas penchés sur la fonctionnalité de la sauvegarde par manque de temps. Pour pimenter l'expérience de jeu, un mode urgence vient s'ajouter par-dessus tout cela. Le mode urgence est un évènement temporaire durant lequel un élément perturbateur pouvant causer des problèmes apparaît sur le jardin. Le temps est alors ralenti, on ne passe pas d'une semaine à une autre, le temps s'écoule en heures. Le mode urgence se déclenche de manière aléatoire si un intrus arrive dans le jardin. Les deux types d'intrus possibles sont des rats ou encore des Indominus, en référence au projet réalisé lors du semestre précédent. Le rat cherche un terrain avec des plantes dessus, puis une ligne (correspondant à une rangée du terrain) où il y a des plantes, et il se déplace de droite à gauche. Le joueur a un laps de temps assez court pour vite réagir en appuyant sur la barre d'espace pour effrayer le rat, ce qui le fera fuir du terrain. Si le joueur ne réagit pas à temps, le rat arrive sur la plante et la mange, elle disparaît donc du terrain et il s'enfuit de lui-même. Quant à l'Indominus, il cherche un terrain avec au moins une plante et arrive par le bas de ce terrain. S'il atteint la plante, il la mange puis repart comme le rat. Le joueur dispose de grenades qu'il peut utiliser pour faire fuir l'Indominus, mais comme il est en mouvement, la probabilité de le toucher est de ⅓, en gardant en tête qu'il y a une chance sur quatre que la grenade fasse exploser une plante au lieu de toucher l'Indominus. Si on arrive à le toucher, il s'enfuit et la plante continue de vivre.

IV. Modélisation objet réalisée

Pour réaliser ce projet, nous avons utilisé la programmation orientée objet. Pour justifier notre code, nous vous présentons ici nos différentes classes en utilisant des diagrammes UML (Unified Modeling Language). Les attributs et méthodes publics sont représentés par un “+” et les privés par un “-”. Une classe ou une méthode abstraite est écrite en italique. Les flèches pleines représentent un lien d’héritage où la flèche part de la classe héritée vers sa classe mère. Les autres flèches représentent des liens entre les différentes classes avec les cardinalités représentées de chaque extrémité de la flèche, avec une phrase verbale le long de la flèche pour expliciter la nature du lien.

A. Classe Plante et ses classes dérivées

Pour définir l’ensemble de nos plantes, nous avons créé une classe abstraite Plante. Elle a pour attributs :

- Un nom et un symbole (la (ou les) première(s) lettre(s) de la plante) pour l’identifier ;
- Un niveau d’eau et un état de santé pour surveiller son bien-être ;
- Un booléen EstVivante qui passe à false si la plante meurt ;
- Un booléen EstMure qui passe à true lorsque la plante peut être récoltée ;
- Un stade de croissance, une vitesse de croissance et une progression de croissance qui sont des indicateurs permettant de suivre et de faire évoluer la croissance ;
- Un terrain préféré qui est le terrain où la plante poussera le mieux ;
- Un terrain actuel qui est le terrain où se trouve la plante en temps réel.

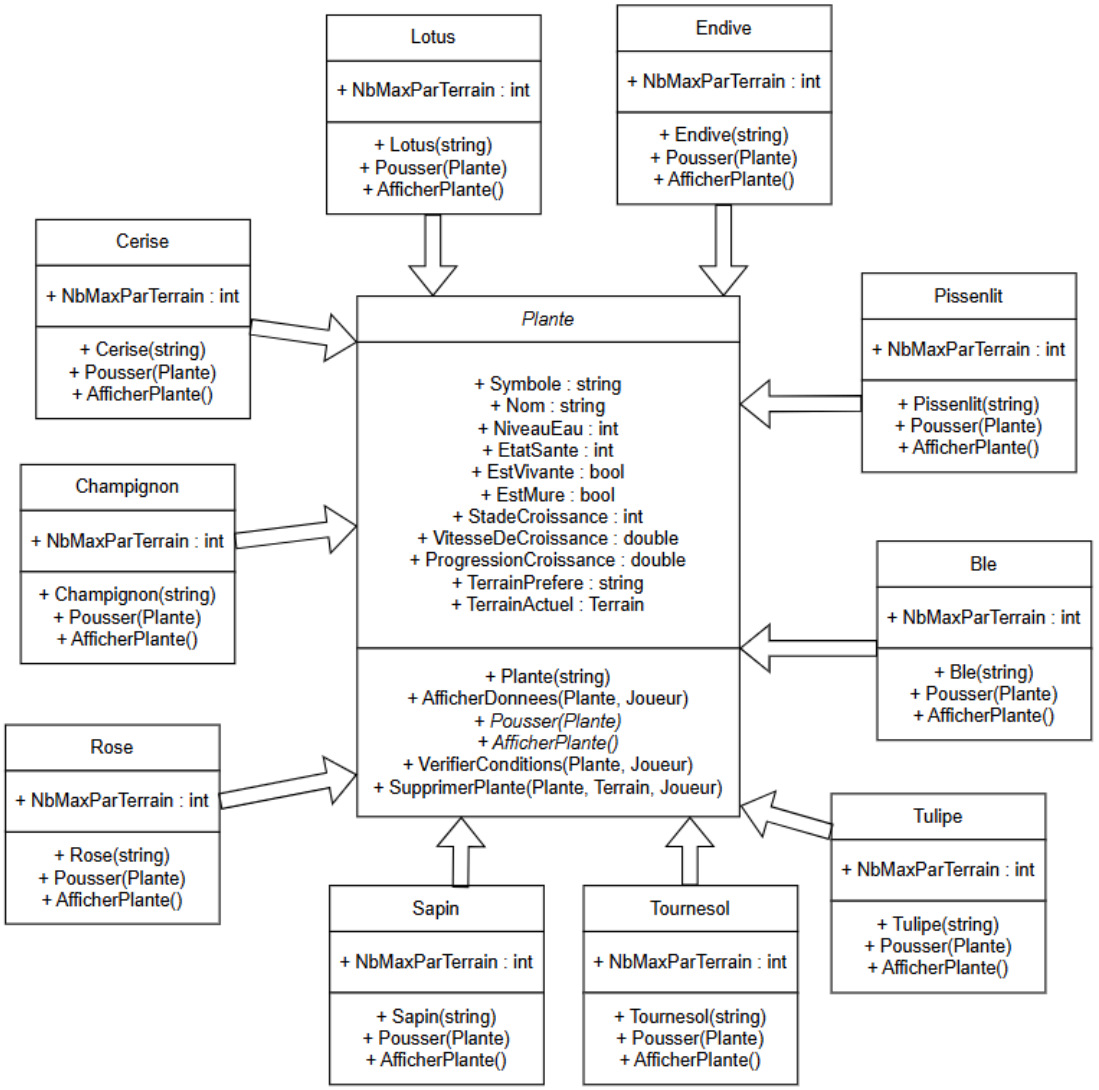
Elle a pour méthodes :

- Son constructeur pour initialiser les caractéristiques de la plante ;
- AfficherDonnees pour avoir accès aux informations importantes de la plante ;
- Pousser pour faire grandir la plante ;
- AfficherPlante pour montrer la plante dans la console ;
- VerifierConditions pour surveiller l’état de santé de la plante ;
- SupprimerPlante pour enlever une plante du plateau si elle meurt ou est détruite.

À partir de cette classe générale, nous avons utilisé la notion d’héritage pour définir l’ensemble de nos plantes. L’avantage d’avoir créé une classe Plante est que nous n’avons pas eu à redéfinir pour chaque plante toutes ces propriétés. Nous avons défini dix plantes différentes qui sont le blé, la cerise, le champignon, l’endive, le lotus, le pissenlit, la rose, le sapin, le tournesol et la tulipe. Pour chacune de ces plantes, nous avons défini un attribut qui fixe un nombre maximal d’une même plante sur un terrain. Dans le constructeur de chacune, nous avons précisé quel était le terrain préféré de cette plante (terre, sable ou argile). Nous avons surchargé la méthode Pousser pour qu’elle puisse faire évoluer les différentes plantes, et également la méthode AfficherPlante qui affiche tour par tour la plante à l’aide d’un switch/case qui contrôle le stade de croissance de la plante pour savoir quel motif afficher. Le dessin de la plante en ASCII Art est donc affiché au fur et à mesure de sa

croissance. Elles ont toutes un stade de croissance propre à elles, et dès qu'il est atteint, la plante en question devient mûre et peut être récoltée par le joueur.

Voici le diagramme UML de la classe Plante ainsi que de ses classes dérivées :



B. Classes Plateau et Terrain (et ses classes dérivées)

Pour avoir une interface agréable et fonctionnelle, nous avons dû définir une classe Plateau, et également une classe abstraite Terrain pour y semer nos différentes plantes. Pour offrir une certaine variété dans le choix du terrain où semer, nous avons défini trois types de terrains : la terre, le sable et l'argile.

La classe Terrain a pour attributs :

- Le type du terrain pour indiquer s'il s'agit d'un terrain terreux, sableux ou argileux ;
- Un tableau de plantes qui permet de créer les cases où les planter ;
- Une matrice de booléens pour gérer la présence des mauvaises herbes

Elle a pour méthodes :

- Son constructeur pour initialiser les caractéristiques du terrain ;
- AfficherTerrain pour afficher visuellement le terrain dans la console ;
- FairePousserMauvaiseHerbe pour faire pousser une mauvaise herbe sur une case d'un terrain.

Les classes héritées n'ont pas d'attributs ou de méthodes supplémentaires, elles ont juste leur constructeur qui les initialise avec un certain nombre de lignes et de colonnes, et également un type de terrain pour savoir s'il s'agit d'un terrain de type sable, terre ou argile.

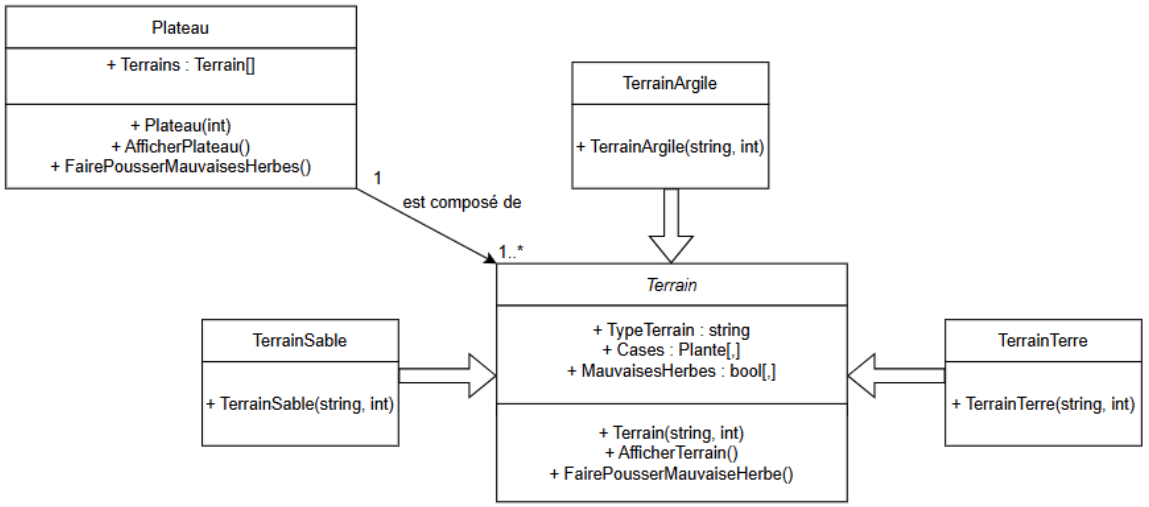
Concernant la classe Plateau, elle a comme attribut :

- Un tableau de terrains pour créer un jardin composé de différentes zones.

Elle a pour méthodes :

- Son constructeur pour initialiser les caractéristiques du plateau : son nombre de lignes, son nombre de colonnes, et quels terrains se trouvent à l'intérieur ;
- AfficherPlateau pour afficher visuellement le plateau en affichant les différents terrains le composant ;
- FairePousserMauvaisesHerbes pour faire pousser des mauvaises herbes sur chaque terrain.

Voici le diagramme UML des classes Plateau et Terrain (et de ses classes dérivées) :



Le lien de cardinalité entre Plateau et Terrain montre qu'un plateau est composé d'un ou plusieurs terrains. Nous aurions pu mettre 0 ou plusieurs, mais un plateau sans terrains serait vide et n'aurait donc aucun intérêt dans le cadre du jeu, d'où l'intérêt d'en avoir au moins un.

C. Classe Meteo

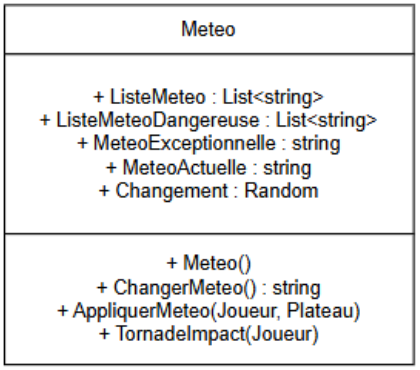
Cette classe nous a permis de gérer la météo qui s'applique sur notre jardin. Elle a pour attributs :

- Une liste des différentes météos pouvant se produire durant la partie ;
- Une liste de météos dites "dangereuses" car elles peuvent perturber l'état de santé les plantes (jusqu'à les détruire comme la tornade) ;
- Une météo exceptionnelle qui est le petit vent échangeant deux plantes aléatoires ;
- Une météo actuelle qui est la météo qui s'applique sur la semaine en cours ;
- Un objet aléatoire pour faire varier la météo à chaque tour.

Elle a pour méthodes :

- Son constructeur pour initialiser les caractéristiques de la météo ;
- ChangerMeteo qui permet de changer la météo choisie aléatoirement ;
- AppliquerMeteo qui permet d'appliquer les effets de la météo au jardin ;
- TornadeImpact qui peut détruire une plante si la météo est la tornade.

Voici le diagramme UML de la classe Meteo :



D. Classe Intrus

Nous avons modélisé une classe pour représenter les intrus pouvant s'introduire dans notre jardin. Pour permettre d'avoir plusieurs évènements différents, nous avons défini deux sous-classes pour avoir deux types d'intrus : le rat et l'Indominus. La classe Intrus a pour attributs :

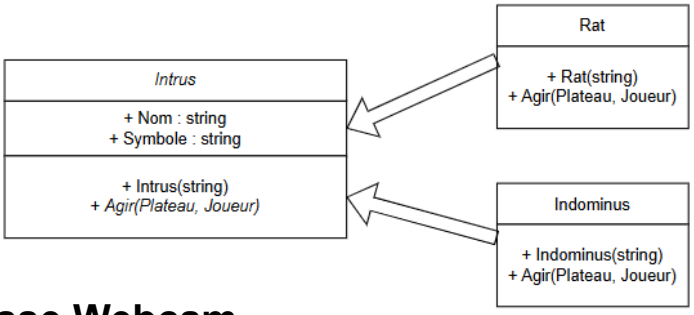
- Son nom et son symbole (première lettre de l'intrus : par exemple, "R" pour "Rat).

Elle a pour méthodes :

- Son constructeur pour initialiser les caractéristiques de l'intrus ;
- Agir qui définit la façon dont l'intrus se déplace dans le jardin.

Les classes héritées n'ont pas d'attributs ou de méthodes supplémentaires, elles ont juste leur constructeur qui les initialise. Cependant, la méthode Agir a été surchargée pour permettre de décrire le comportement de chaque intrus suivant qu'il soit un rat ou un Indominus.

Voici le diagramme UML de la classe Intrus et de ses classes héritées :

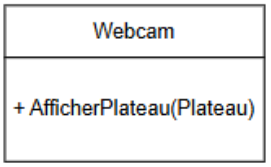


E. Classe Webcam

Nous avons créé cette classe pour permettre au joueur d'avoir un visuel du jardin en activant une webcam.

Elle a n'a pas d'attributs et sa seule méthode est AfficherPlateau, une fonction d'affichage du jardin si le joueur l'a demandé.

Voici le diagramme UML de la classe Webcam :

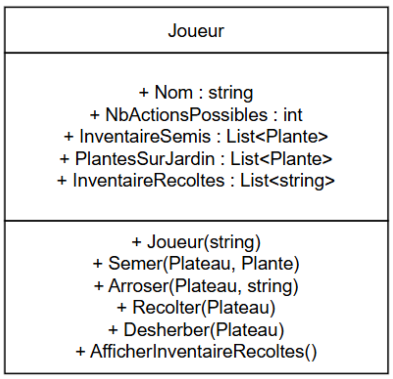


F. Classe Joueur

Nous avons représenté le joueur dans une classe. Elle a pour attributs son nom, le nombre d'actions possibles pour le joueur, son inventaire de semis pour qu'il sache ce qu'il peut planter, l'ensemble des plantes qui sont sur son jardin, et l'inventaire de ses récoltes pour qu'il puisse voir les plantes qu'il a récupérées.

Elle a pour méthodes le constructeur qui initialise les propriétés du joueur, Semer pour planter nos plantes, Arroser pour les arroser, Recolter pour les récupérer lorsqu'elles sont mûres, Desherber pour enlever les mauvaises herbes et AfficherInventaireRecoltes pour avoir accès aux plantes récoltées.

Voici le diagramme UML de la classe Joueur :

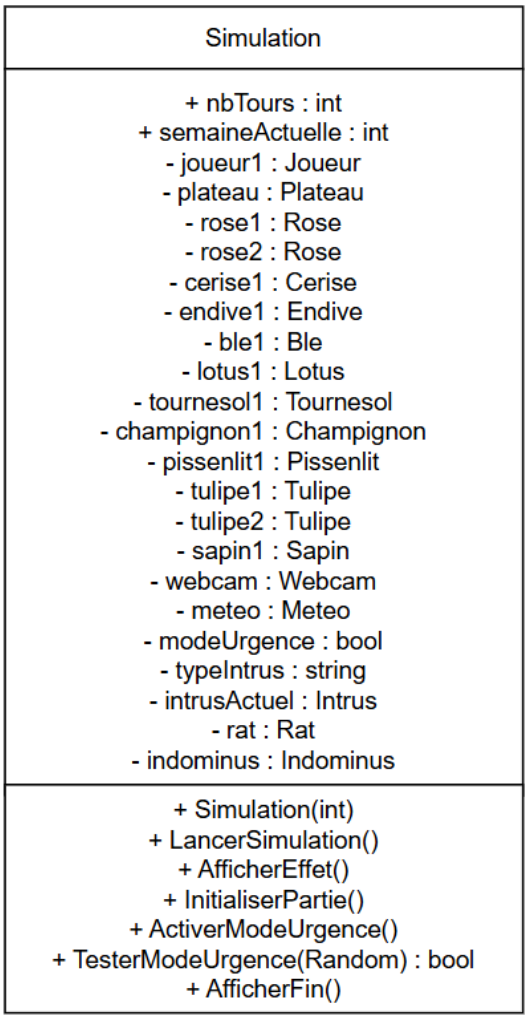


G. Classe Simulation

Enfin, la classe Simulation permet de gérer le déroulement de la partie en appelant toutes les autres classes aux moments opportuns. Elle a pour attributs le nombre de tours de la partie, la semaine actuelle du tour, le joueur associé à cette partie, le plateau qui représente son jardin, deux roses, une cerise, une endive, une pousse de blé, un lotus, un tournesol, un champignon, un pissenlit, deux tulipes et un sapin pour définir l'inventaire des semis du joueur, mais également une webcam pour surveiller le jardin, une météo associée à la partie, un booléen pour activer le mode urgence, une chaîne de caractères pour caractériser le type d'intrus, un intrus, et deux types d'intrus qui sont le rat et l'Indominus.

Elle a pour méthodes son constructeur pour initialiser la simulation, LancerSimulation pour faire tourner le jeu, AfficherEffet pour lancer l'effet de début avec l'affichage du titre, InitialiserPartie pour définir les caractéristiques du joueur, ActiverModeUrgence pour lancer le mode urgence si les conditions sont réunies, TesterModeUrgence qui permet de voir si les conditions sont réunies, et AfficherFin pour avoir une animation à la fin de la partie.

Voici le diagramme UML de la classe Simulation :



V. Tests réalisés

Afin de garantir le bon fonctionnement de la simulation de jardin, nous avons mis en place des tests dans les différentes classes du projet. Ces tests ont été réalisés en simulant des scénarios concrets d'utilisation, comme la croissance d'une plante, l'arrosage, l'évolution des états en fonction du temps ou encore l'ajout de plantes sur le plateau. À chaque étape, nous avons utilisé des instructions d'affichage (Console.WriteLine) pour visualiser l'évolution des objets et détecter rapidement d'éventuels comportements anormaux. Cela nous a permis de valider les interactions entre les différentes classes, de vérifier la cohérence des règles de jeu et de corriger les erreurs au fur et à mesure du développement.

Il a également été nécessaire de faire une série de tests concernant la météo. Étant donné son impact direct sur l'état des plantes, nous devions être sûrs des effets de chaque condition climatique sur le jardin. Pour cela, nous avons lancé de nombreuses simulations de changements météorologiques en boucle, tout en observant les évolutions des attributs des plantes (niveau d'eau, santé, vitesse de croissance). Des tests ont aussi été réalisés pour contrôler les cas exceptionnels, comme le déplacement aléatoire de plantes lors d'un petit vent ou la suppression d'une plante à la suite d'une tornade. L'utilisation du Console.WriteLine nous a permis de suivre pas à pas ces événements dynamiques et d'identifier rapidement les éventuelles incohérences (par exemple, une plante déplacée sans mise à jour de sa position dans le terrain).

Nous souhaitions introduire l'Indominus pour faire un clin d'œil au dernier projet, mais pour que cette fonctionnalité soit opérationnelle, nous avons dû réaliser une grande série de tests pour faire s'il était possible de le toucher avec une grenade, si se rater détruisait vraiment une plante, etc. Pour faciliter cette phase de tests, nous avions provisoirement augmenté sa probabilité d'apparition afin de limiter le nombre de tests.

Pour mieux suivre le code, nous avons commenté le plus possible nos différentes méthodes afin que nous puissions reprendre le travail de l'autre sans difficulté, et cela permettra également à une personne extérieure reprenant notre code de s'y plonger en le comprenant sans trop de problèmes.

Nous avons codé l'ensemble du projet en réalisant des tests à chaque nouvelle fonctionnalité implémentée, cela nous a permis de vérifier que tout se réalisait comme prévu, et quand cela ne marchait pas, nous trouvions de nouvelles solutions. Cette démarche nous a fait prendre conscience de l'intérêt de faire des tests régulièrement dans un projet, et nous avons également pu nous exercer avec le débogueur lorsque les affichages de tests n'étaient pas suffisants pour trouver l'origine du problème.

VI. Gestion de projet

A. Organisation de l’équipe

Notre équipe était composée de deux membres : Cécile Tollis et Noé Peyrot. Dès le début du projet, nous avons mis en place une organisation collaborative simple mais efficace, visant à maximiser notre productivité tout en assurant la qualité du développement.

Nous avons choisi de diviser le travail de manière équilibrée, en affectant à chacun des fonctionnalités distinctes mais complémentaires. Par exemple, l’un se concentrait sur la modélisation des plantes et des terrains, tandis que l’autre développait les mécaniques du mode classique et de la météo. Cette répartition nous a permis d'avancer en parallèle, tout en exploitant au mieux nos compétences respectives.

Afin de faciliter le développement collaboratif, nous avons utilisé GitHub pour le versionnement du code. Chaque fonctionnalité majeure faisait l’objet d’un commit dédié, permettant une intégration progressive. Les communications se faisaient via WhatsApp et en présentiel à l’école, ce qui nous a permis de rester réactifs tout au long du projet.

Nous avons choisi de rédiger le rapport technique ensemble, pour garantir une cohérence dans les explications et refléter fidèlement notre travail collectif. Cette démarche est également documentée dans la matrice d’implication, jointe à la racine de notre projet GitHub.

En cas d’obstacles techniques, nous avons su adapter notre organisation : travail en binôme temporaire sur les parties bloquantes, ajustement des priorités, ou simplification temporaire de certains modules en attendant une solution.

B. Planning de réalisation

Le projet a officiellement débuté le mardi 1er avril 2025, avec un rendu final fixé au vendredi 23 mai 2025. Nous avions pour objectif de travailler régulièrement dès le début afin d’avancer de manière progressive et structurée, tout en prenant de l’avance. Toutefois, cette organisation a rapidement été mise à l’épreuve par l’importante charge de travail imposée par d'autres cours, eux-mêmes exigeants et ponctués de rendus importants.

Malgré nos efforts pour respecter notre planning initial, il s’est avéré difficile de maintenir un rythme constant tout au long des sept semaines du projet. Nous avons parfois dû prioriser d'autres obligations académiques, ce qui a ralenti l'avancement sur certaines phases critiques du développement.

Conscients de l’échéance qui approchait, nous avons su intensifier nos efforts durant les deux dernières semaines. Cette période a été marquée par une forte mobilisation de l'équipe, avec des séances de travail plus longues et une coordination accrue pour rattraper le retard accumulé. Cela nous a permis de finaliser les fonctionnalités essentielles du jeu tout en produisant une documentation complète et soignée. Bien que cette accélération ait été éprouvante, elle témoigne de notre capacité à nous adapter aux imprévus et à travailler efficacement sous pression pour atteindre nos objectifs. Même si le jeu n’est pas entièrement approfondi, nous avons su réaliser une version fonctionnelle dont nous sommes globalement satisfaits.

VII. Partie créative

Pour la partie créative du projet, nous avons tout d'abord mis un accent sur l'esthétique de l'affichage notamment avec l'utilisation de nombreuses couleurs. Les consignes importantes s'affichent en rouge et avec un délai suffisamment long pour que l'utilisateur puisse en prendre connaissance, le numéro de la semaine s'affiche en magenta à chaque tour et lorsque le mode urgence est activé, le fond de la console devient totalement rouge, indiquant un danger.

À chaque tour, le joueur peut voir l'évolution de la pousse de ses plantes grâce à un affichage en console. Nous avons choisi des dessins en caractères ASCII qui représentent le plus fidèlement possible les plantes du joueur. Ces représentations permettent une visualisation rapide de l'état des plantes et rendent l'expérience du jeu plus immersive.

Pour le mode urgence, nous avons choisi de simuler l'attaque soudaine d'un rat ou de l'indominus ! En référence au projet de programmation du semestre dernier, le joueur a la possibilité de faire fuir l'indominus en jetant des grenades, mais attention à ne pas faire exploser le jardin…

Nous avons également rajouté deux mécaniques de jeu intéressantes concernant la météo :

- Tout d'abord, dans les météos dangereuses, nous avons décrit une option "tornade". Lorsque la tornade apparaît, une boucle permet de tester chacune des plantes sur le jardin pour savoir si elle va être arrachée, chaque plante ayant une chance sur trois d'être arrachée durant la tornade. Cette fonctionnalité ajoute du suspense au cours de la partie.

- Enfin, nous avons créé une météo exceptionnelle appelée "petit vent". Lorsque le petit vent apparaît, deux plantes du jardin, choisies aléatoirement, échangent de position, l'une se retrouvant donc sur le terrain de l'autre. Cette fonctionnalité va certainement impacter l'état de santé des plantes si celles-ci se trouvaient sur leur terrain préféré.

VIII. Difficultés rencontrées

Au cours de la réalisation de ce projet, nous avons rencontré plusieurs défis majeurs que nous avons dû surmonter :

- Manque de temps : Le délai imparti pour mener à bien ce projet s'est avéré assez court au vu de la complexité des tâches à accomplir. Cela nous a imposé une gestion stricte des priorités et un rythme de travail soutenu, pour rendre un jeu le plus avancé possible.

- Utilisation difficile de GitHub : Nous avons bien mieux géré le versionnement de notre code sur GitHub par rapport au semestre dernier, même si certains problèmes ont subsisté. Au départ, notre espace de travail ne fonctionnait pas, et nous avons parfois dû effectuer des commits “à la main” pour permettre à l’autre de suivre l’avancée de notre code.

- Difficultés de codage : La programmation orientée objet en C# a présenté des défis techniques. La création des différentes classes nécessaires pour le fonctionnement du jeu ont demandé un travail d'analyse et de débogage important pour assurer un bon fonctionnement.

- Rédaction d’un long rapport : En parallèle du développement technique, la rédaction de ce rapport a demandé de trouver un équilibre entre la réalisation du jeu et la documentation du travail effectué. Nous avons essayé de tenir à jour ce rapport lors de nos avancées afin de le rédiger partie par partie et qu’il ne représente pas une charge de travail trop importante.

Malgré ces obstacles, notre équipe a réussi à avancer en trouvant des solutions adaptées, ce qui nous a permis de mener à bien le projet.

IX. Bilan critique

Le projet ENSemenCe nous a permis de consolider un large éventail de compétences comme la programmation orientée objet et la structuration d'un système complexe.

Parmi les points positifs, nous avons réussi à modéliser de manière cohérente et claire un ensemble de classes qui ont permis de mettre en place un déroulement opérationnel d'une partie. Le travail sur GitHub et l'adoption de bonnes pratiques de versionnage ont largement facilité l'intégration de nos développements respectifs.

Néanmoins, ce projet nous a également confrontés à certaines limites. Le manque de temps, accentué par une charge de travail importante, a freiné le développement de certaines fonctionnalités. Le mode urgence reste simplifié, et certains éléments interactifs pourraient encore être approfondis pour améliorer l'immersion et la fluidité du jeu. Nous avons le sentiment que ce sujet était plus complexe que celui du premier semestre de par son absence de contraintes, car même si cela nous a permis de développer notre créativité, nous avons eu des difficultés au départ car nous n'avions aucune idée des classes attendues, du visuel qu'il fallait réaliser, contraintes que nous n'avions pas eu pour le jeu de plateau car toutes les exigences étaient dans le cahier des charges.

En conclusion, ENSemenCe est un projet ambitieux que nous avons essayé de réaliser au mieux. Il nous a permis de progresser significativement en programmation avancée, en modélisation système et en gestion de projet. Le résultat, bien qu'améliorable, est à la hauteur des objectifs initiaux, et ouvre des perspectives claires d'amélioration si une suite devait être envisagée. Nous en tirons un apprentissage concret, tant sur le plan technique que méthodologique.