CO6SFPA0 : 1A Programmation Avancée

Projet 2025 : ENSemenC

Justificatif technique

UNG Maxime & RECH Mathieu

Groupe 1 & 2 - 1A

Promotion 2027

Table des matières :

Introduction :

Dans le cadre du module de Programmation Avancée dispensé à l'ENSC au semestre 6, nous avons développé le projet ENSemenC : un simulateur de potager interactif visant à mettre en œuvre les principes fondamentaux de la programmation orientée objet (POO) en langage C#. Ce projet s'inscrit dans la continuité pédagogique du premier semestre où nous avions conçu JurENSiC World, un jeu basé sur une architecture procédurale. L'objectif était alors de maîtriser la logique algorithmique et les structures de contrôle de base. ENSemenC, en revanche, nous a permis d'approfondir la conception logicielle en orientant notre réflexion autour de la modularité, de l'héritage, de l'encapsulation et du polymorphisme. Le jeu, centré sur la gestion d'un potager virtuel, illustre concrètement ces concepts en simulant le comportement de plantes influencées par leur environnement et les décisions du joueur.

Ce rapport présente les objectifs et le fonctionnement général du jeu ENSemenC, puis décrit l'architecture orientée objet que nous avons mise en place ainsi que les relations entre les différentes classes du programme. Il détaille ensuite la structure du code et son organisation fonctionnelle, avant de revenir sur notre organisation de travail en binôme et le déroulement du projet. Enfin, un bilan critique et des perspectives d'amélioration viennent conclure notre analyse.

I - Présentation du jeu :

1) Objectif

Le jeu ENSemenC propose au joueur d'incarner un jardinier en charge de l'entretien d'un potager au fil des semaines. Il s'agit de gérer efficacement ses plantations pour assurer leur croissance, leur santé et leur récolte, tout en s'adaptant à des conditions environnementales changeantes. L'objectif pédagogique est de faire comprendre au joueur la complexité d'un écosystème cultivé, en lui donnant les moyens d'agir sur des paramètres tels que l'arrosage ou les périodes de semis, et d'observer les conséquences de ses choix. La finalité du jeu n'est pas uniquement de récolter le plus de plantes possible, mais de maintenir un équilibre durable et cohérent entre les besoins des plantes, les caractéristiques des terrains et les aléas climatiques.

2) Déroulement d'une partie

Une partie se déroule sous forme de tours hebdomadaires, chacun correspondant à une semaine dans le jeu. Au début de chaque tour, la saison en cours est mise à jour, influant sur la température, les précipitations et la luminosité. Ces valeurs, générées aléatoirement mais dans une cohérence saisonnière, impactent directement les conditions de croissance des plantes.

Le joueur dispose de cinq actions par semaine. Il peut notamment :

- Choisir de nouvelles graines à semer parmi celles possibles, en fonction de la saison et du terrain sélectionné
- Choisir une plante à arroser pour en ajuster l'hydratation
- Choisir une plante à récolter parmi les plantes arrivées à maturité
- Passer à la semaine suivante.

Chaque plante évolue en fonction de son environnement, peut tomber malade ou mourir si les conditions sont trop défavorables. Un affichage console permet au joueur de suivre en temps réel l'état de chaque plante : taille, âge, terrain, état de santé.

À tout moment, une situation d'urgence peut survenir : une invasion de rats. Ce mode urgence suspend temporairement le déroulement normal de la semaine et introduit un mini-événement interactif. Le joueur a alors la possibilité de choisir parmi plusieurs réponses tactiques :

- Faire du bruit pour tenter de faire fuir le rat
- Choisir 2 plantes à mettre en serre pour les protéger
- Déployer une bâche pour réduire les dégâts
- Installer un épouvantail afin de diminuer la probabilité de futures invasions
- Sacrifier ses plantations en utilisant l'option extrême de "terre brûlée" pour exterminer le rat

Ce système introduit un aspect aléatoire et stratégique supplémentaire, stimulant l'adaptation et la gestion de crise.

La simulation se poursuit tant que le joueur le souhaite. Il n'existe pas de fin unique, ni de score absolu, mais une synthèse est proposée à la fin de la partie pour récapituler les récoltes effectuées. Le but est d'inciter à une gestion durable et réfléchie du potager dans un environnement semi-réaliste.

II - Modélisation des données :

Cette partie présente la façon dont les éléments principaux du jeu ENSemenC sont représentés et gérés à travers des structures orientées objet. Elle expose les différentes classes mises en place, leurs attributs, propriétés et méthodes, en insistant sur leur rôle dans le fonctionnement du simulateur. L'ensemble repose sur l'utilisation des notions d'encapsulation, d'héritage et d'abstraction, telles qu'abordées dans les cours de Programmation Avancée.

1) Plantes

Les plantes constituent les entités centrales du jeu. Elles sont modélisées par une classe abstraite `Plante`, définie dans le fichier `Plante.cs` qui regroupe les caractéristiques communes à

toutes les variétés. Cette classe n'est pas instanciable directement, ce qui impose le passage par des classes filles concrètes pour redéfinir les comportements spécifiques à chaque espèce. Chaque classe fille utilise un constructeur pour initialiser l'ensemble des attributs en fonction des spécificités biologiques de l'espèce simulée. Pour garantir l'encapsulation, la majorité des attributs sont déclarés en `private`, ce qui protège l'intégrité des données en empêchant leur modification directe depuis l'extérieur.

Les attributs principaux incluent :

- `private string nom` : nom de la plante
- `private bool estVivace` : indique si la plante est vivace ou annuelle
- `private bool estComestible` : indique si la plante est comestible
- `private List<string> saisonsSemis` : saisons compatibles avec le semis
- `private string typeTerrainPrefere` : type de terrain le plus adapté
- `private double espacement` : distance minimale nécessaire entre deux plants (en cm)
- `private double surfaceNecessaire` : surface requise à la plantation (en m²)
- `private double vitesseCroissance` : croissance hebdomadaire (en cm)
- `private double besoinEau` : besoin en eau hebdomadaire (en L)
- `private double besoinLuminosite` : besoin en lumière (en % de luminosité)
- `private Tuple<double, double> plageTemperature` : température idéale min/max (°C)
- `private List<string> maladiesPotentielles` : maladies que peut contracter la plante
- `private double esperanceDeVie` : durée de vie maximale (en semaines)
- `private int productionMax` : production maximale atteignable
- `private Terrain terrainAssocie` : terrain affecté à la plante
- `protected bool estMalade` : état de santé

Les attributs `espacement` et `surfaceNecessaire` ne sont pas encore exploités dans les fonctionnalités actuelles du jeu mais ont été prévus pour permettre des extensions ultérieures. `estMalade` est déclaré comme `protected` car modifiable uniquement par les méthodes des sous-classes ou la classe mère pour respecter l'encapsulation tout en permettant une gestion flexible des états pathologiques.
L'accès à l'ensemble de ces attributs s'effectue via des propriétés publiques en lecture seule correspondant à chacun des attributs et assurant un contrôle précis et sécurisé. La propriété `public double Taille { get; protected set; }` est également ajoutée pour permettre de suivre l'évolution hebdomadaire de la hauteur de la plante.

La classe `Plante` déclare deux méthodes abstraites, redéfinies dans chaque sous-classe, permettant de personnaliser les comportements des plantes selon leur espèce :

- `Pousser(tauxConditionsFavorables)` : simule la vitesse de croissance en fonction de l'environnement
- `PeutEtreRecoltee(age)` : détermine si la plante est prête à être récoltée en fonction de sa maturité

Elle propose aussi des méthodes communes permettant de modéliser des comportements partagés par toutes les espèces :

- `AttraperMaladie()` : gère les infections potentielles
- `Grignoter()` : modélise les dégâts causés par les rongeurs
- `CalculerTauxConditions()` : évalue l'adéquation entre les conditions environnementales et les besoins de la plante

Les espèces du jeu sont déclinées en 12 classes filles qui redéfinissent les méthodes abstraites et les attributs propres à chaque espèce. Chaque classe est définie dans un fichier distinct :

- `Aubergine.cs`
- `Carotte.cs`
- `Courgette.cs`
- `Fraise.cs`
- `Laitue.cs`
- `Lavande.cs`
- `Menthe.cs`
- `Radis.cs`
- `Rose.cs`
- `Tomate.cs`
- `Tournesol.cs`
- `Tulipe.cs`

Dans le fonctionnement de la boucle principale du jeu, gérée au sein du fichier `GameLoop.cs`, chaque plante instanciée est ajoutée à une collection `List<Plante>` gérée par la classe `Game`. Cette liste permet de conserver et manipuler dynamiquement les plantes au fil des semaines selon leur état et leur évolution.

2) Terrains

Les terrains constituent un paramètre environnemental déterminant pour la croissance des plantes. Ils sont modélisés par une classe abstraite `Terrain`, définie dans `Terrain.cs` qui définit les caractéristiques générales communes à tous les types de sols. Cette classe n'est pas instanciable directement, et impose aux sous-classes de définir leurs propriétés propres via leur constructeur. À ce stade du développement, aucune méthode abstraite n'a été définie, mais la structure de la classe permet d'en ajouter ultérieurement pour spécialiser davantage le comportement de chaque type de terrain.

Les attributs sont :

- `private string nom` : nom du terrain

- `private double capaciteEau` : capacité maximale de rétention d'eau (en L/m²)
- `private double fertilite` : niveau de fertilité (sur 100)
- `private double quantiteEauActuelle` : quantité d'eau présente dans le sol à un instant donné (en L/m²) qui influe sur l'hydratation des plantes

L'encapsulation est ici assurée par l'utilisation de propriétés publiques en lecture seule correspondant à chacun des attributs.

Un terrain peut être modifié uniquement via les méthodes communes suivantes :

- `public void AjouterEau(double quantite)` : ajoute une certaine quantité d'eau au terrain sans dépasser la capacité de rétention maximale définit dans `capaciteEau`
- `public string AfficherProprietes()` : retourne une fiche descriptive des caractéristiques du terrain à partir de ses propriétés

Chaque type de sol est défini par une classe fille héritant de `Terrain` et déclarée dans son propre fichier :

- `Terre.cs` : terrain équilibré, capacité moyenne et bonne fertilité
- `Sable.cs` : terrain filtrant, faible capacité en eau et fertilité réduite
- `Argile.cs` : terrain dense, forte rétention d'eau mais fertilité moyenne

L'association entre une plante et un terrain se fait via la méthode `Plante.AssocierTerrain()` présente dans la classe `Plante`. Cette relation influe directement sur le calcul du taux de respect des conditions optimales de croissance, évalué chaque semaine dans la boucle de jeu `Game`.

3) Conditions climatiques

Les conditions climatiques sont un facteur clé qui influence chaque semaine l'évolution des plantes. Leur représentation est assurée par la classe `Saison`, définie dans le fichier `Saison.cs`, qui regroupe à la fois la gestion du temps calendaire (nom de la saison et numéro de semaine) et la génération des paramètres météorologiques hebdomadaires. Elle débute par la définition de l'énumération `enum SaisonNom`, intégrée directement dans le fichier, qui énumère les quatre saisons standards du cycle annuel : Hiver, Printemps, Été et Automne. Cette énumération typée permet d'assurer une manipulation explicite, cohérente et sécurisée des saisons tout au long du jeu.

Les propriétés auto-implémentées sont utilisées pour représenter les données de la saison. Elles disposent d'un accesseur `get` public pour permettre la lecture extérieure, et d'un accesseur `set` privé pour restreindre la modification à l'intérieur de la classe uniquement, ce qui garantit l'encapsulation :

- `public SaisonNom Nom { get; private set; }` : nom de la saison en cours (dans l'ordre d'apparition : Hiver, Printemps, Été, Automne)
- `public int NumeroSemaine { get; private set; }` : numéro de la semaine dans le jeu (1 à 52)

À partir de ces informations, la classe utilise plusieurs méthodes pour générer de manière pseudo-aléatoire les paramètres climatiques adaptés à la saison en cours. Cette génération repose sur l'objet `Random rng`, instancié en tant qu'attribut privé de la classe, afin d'assurer une variabilité contrôlée à chaque appel :

- `public double GenererTemperature()` : retourne une température aléatoire (en °C)
- `public double GenererPrecipitations()` : retourne une quantité de précipitations simulée (en L/m²)
- `public string DeterminerSaison()` : détermine la saison en fonction du numéro de semaine en cours, chaque saison durant 13 semaines.

Chaque ensemble de conditions hebdomadaires est généré de manière pseudo-aléatoire, mais dans des plages définies adaptées à chaque saison. Ces plages sont précisées dans un `switch-case` appliqué à l'énumération `SaisonNom`, ce qui permet d'associer à chaque saison des bornes spécifiques pour la température, les précipitations et la luminosité, assurant ainsi un réalisme saisonnier dans la simulation.

Les valeurs retournées sont utilisées chaque semaine dans la méthode `private void SimulerSemaine()` de la classe `Game` de `GameLoop.cs` pour évaluer les conditions environnementales de chaque plante. Elles entrent directement dans le calcul du taux de respect des conditions de croissance effectué par `Plante.CalculerTauxConditions()` et déterminent également la quantité d'eau ajoutée aux terrains en début de semaine, via la méthode `Terrain.AjouterEau()` appliquée à chaque instance de terrain. Cette classe permet ainsi de simuler un environnement agricole semi-réaliste et évolutif, introduisant de la variabilité dans les cycles de croissance sans nécessiter d'intervention manuelle du joueur.

4) Choix des actions

Chaque semaine de jeu donne lieu à un tour pendant lequel le joueur peut effectuer jusqu'à 5 actions. Ces choix sont modélisés dans la classe `Game` du fichier `GameLoop.cs` au sein de la méthode `public void Start()` par la variable `int choix` dont la valeur varie de 1 à 5 selon l'option sélectionnée par le joueur via l'interface console et récupérée par `Console.ReadKey()`. Le nombre d'actions restantes est suivi à l'aide de l'attribut `private int nbActionsRestantes` initialisé à 5 en début de semaine. Chaque action effectuée décrémente cette valeur.

Les actions reliées à chaque valeur de `int choix` sont définies par des méthodes qui permettent d'interagir avec les plantes ou de faire avancer le jeu :

1. `private void SemerPlante()` : permet de sélectionner une plante à semer avec `private Plante ChoisirPlante()` et un terrain avec `private Terrain ChoisirTerrain()`. La plante est alors instanciée et ajoutée à la liste des plantes suivies.
2. `private void ArroserPlantes()` : permet d'arroser une plante choisie parmi celles plantées, ce qui augmente la quantité d'eau du terrain associé via `Terrain.AjouterEau()`.
3. `private void RecolterPlantes()` : permet de récolter les plantes arrivées à maturité. La méthode vérifie cette condition avec `Plante.PeutEtreRecoltee()`.
4. `SimulerSemaine()` : fait avancer le jeu d'une semaine.
5. Quitte le jeu en mettant fin à la méthode `Start()` et affiche une synthèse des plantes récoltées avec `private void AfficherSynthese()`.

Lorsque `nbActionsRestantes` atteint zéro, la semaine prend fin automatiquement. Cela impose une stratégie de priorisation : le joueur doit gérer ses ressources de manière optimale en fonction de la météo, de l'état des plantes et des urgences éventuelles.

5) Passage du temps

Le passage du temps dans le jeu ENSemenC repose sur un découpage hebdomadaire, chaque itération de jeu correspondant à une nouvelle semaine. Ce mécanisme est modélisé par l'attribut `private int semaine`, au sein de la classe `Game` contenue dans `GameLoop.cs`. Cette variable est initialisée à 0 au début de la partie et est incrémentée à chaque tour avec l'appel à la méthode `SimulerSemaine()`.

Ce fonctionnement est central dans la dynamique du jeu, car de nombreux éléments en dépendent :

- La saison courante est recalculée en fonction de `semaine` via le constructeur `Saison()`, influençant les conditions météorologiques et donc les conditions de survie. Chaque saison comprenant 13 semaines en commençant par l'hiver avec la semaine 1.
- Les conditions climatiques hebdomadaires (température, précipitations, luminosité) sont générées à chaque début de semaine par les méthodes de la classe `Saison` mentionnées en II.3
- L'âge de chaque plante augmente, ce qui peut influencer leur récoltabilité selon leurs maturité avec `Plante.PeutEtreRecoltee()`, ainsi que leur survie en cas de dépassement de l'espérance de vie défini dans `Plante.EsperanceDeVie`.

Le déclenchement d'une nouvelle semaine est provoqué manuellement par le joueur, lorsqu'il choisit l'action correspondante ou automatiquement lorsque les 5 actions disponibles ont été utilisées. Ce système de progression temporelle structurée permet d'introduire une régularité dans le jeu tout en rythmant les interactions possibles. Il constitue également un levier important pour la montée progressive en complexité de la simulation (apparition d'événements, vieillissement des plantes, etc.).

6) Mode urgence

Le mode urgence est implémenté dans `Game` à travers l'événement aléatoire d'invasion de rats. Ce système introduit une rupture temporaire dans le déroulement classique du jeu hebdomadaire, imposant une gestion de crise ponctuelle et immédiate.

Ce mode est géré par la méthode `private void InvasionRat()` de la classe `Game`. Il est déclenché avec une probabilité fixée par l'attribut `private double probaInvasionRat`, qui est initialisé à 0.05 soit 5 % au début de la partie. À chaque action effectuée dans la boucle principale, un tirage aléatoire avec `rng.NextDouble()` est effectué pour déterminer si l'événement se produit.

Lorsqu'une invasion a lieu, une boucle d'urgence se met en place pendant trois tours consécutifs, pendant lesquels le joueur doit choisir parmi plusieurs actions : faire du bruit, fermer la serre, déployer une bâche, installer un épouvantail ou brûler le terrain. Chacune de ces options correspond à un `case` spécifique dans un `switch`, et modifie les variables suivantes :

- `int bacheDeployee` : utilisé comme facteur de protection dans la méthode `Plante.Grignoter(Random rng, int bacheDeployee)`. Lorsque `bacheDeployee` vaut 1, il réduit les chances de subir des dégâts complets lors de l'attaque. Ce paramètre est transmis à chaque appel de `Plante.Grignoter()` durant une invasion et réinitialisé à 0 après chaque tour.
- `List<int> plantesProtegeesDansSerre` : contient les indices des plantes temporairement protégées lors de la fermeture de la serre. Celles-ci ne peuvent pas être ciblées par le rat pendant le tour.
- `double probaInvasionRat` : diminue si le joueur installe un épouvantail, réduisant ainsi la probabilité d'une prochaine invasion. Elle est recalculée aléatoirement tout en ne pouvant jamais devenir négative.

Pendant ces tours, les plantes sont vulnérables à des attaques répétées du rat, via la méthode `Plante.Grignoter()` appliquée à une plante aléatoire. Si la plante a une taille nulle après l'attaque, elle est retirée du potager. L'état de la serre et la bâche influencent la sélection et la résistance des plantes ciblées.

7) Affichage et Interface Utilisateur

L'interface utilisateur du jeu ENSemenC repose sur une interface console textuelle. Elle a été pensée pour guider progressivement le joueur dans ses décisions, tout en restant sobre et accessible. À chaque étape, les informations essentielles sont affichées via des blocs structurés utilisant la méthode `Console.WriteLine()`, illustrant les différentes phases de la simulation. Les choix d'actions sont effectuées à chaque étape en entrant un chiffre correspondant à l'action concernée dans la console lu avec `Console.ReadKey()` ou `Console.ReadLine()`

La Figure 1 présente l'ouverture d'une semaine de jeu. On y retrouve un message de bienvenue, les paramètres climatiques de la semaine 1 et le menu des choix d'actions proposé, qui est affiché automatiquement après chaque action.

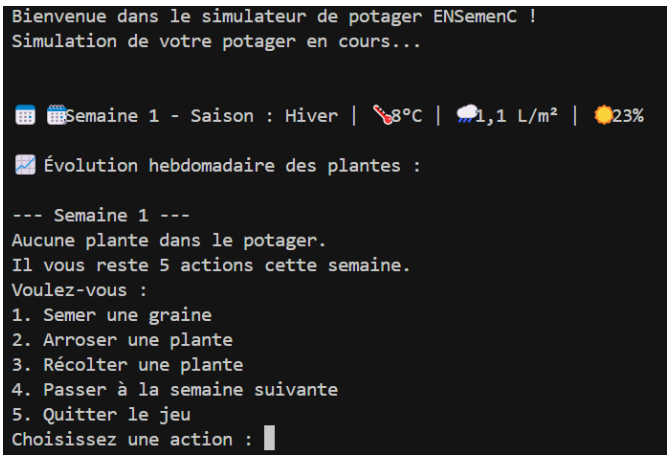

Figure 1 : Interface utilisateur à l'ouverture du jeu

La Figure 2 illustre l'interface de l'action 1 de semis, comprenant un menu de sélection de la plante, suivie de la fiche descriptive de la plante sélectionnée, indiquant la possibilité ou non de semer en fonction de la saison en cours et enfin le menu de choix du terrain de semis qui apparaît lorsque ce dernier est possible.

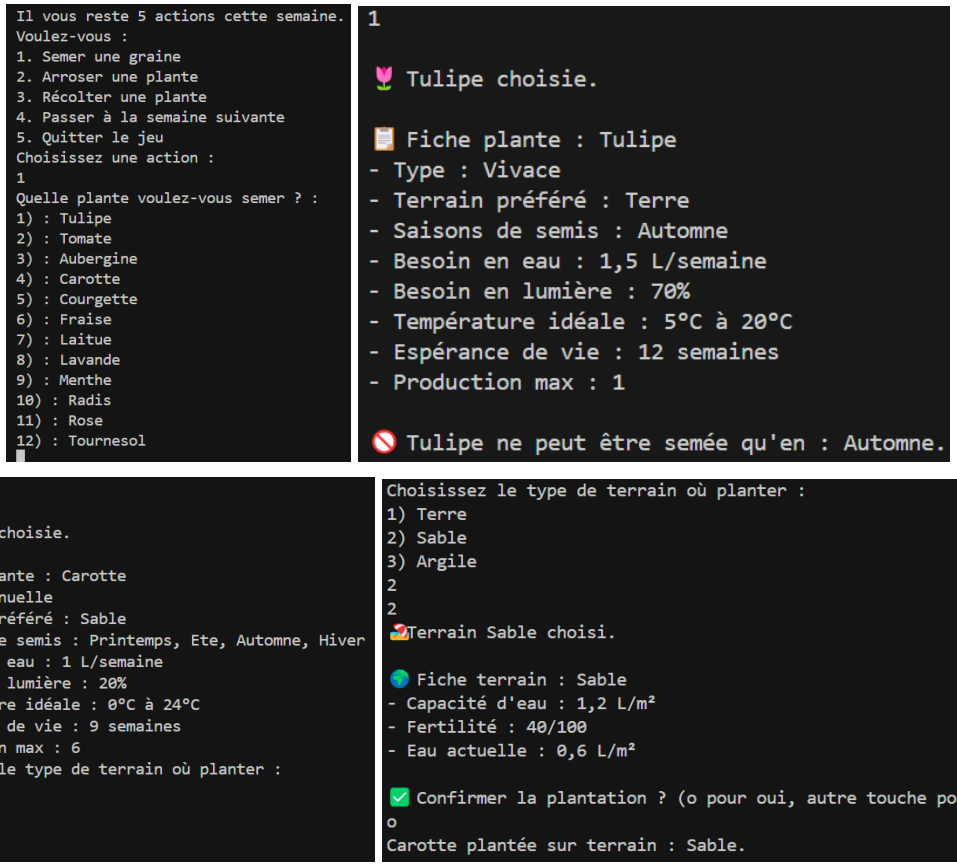

Figure 2 : Interface de l'action 1. semer une graine
(de haut en bas et de gauche à droite : sélection de la plante ; semis impossible ; semis possible ; choix du terrain de semis)

La Figure 3 montre le récapitulatif hebdomadaire qui intervient au passage à la semaine suivante lorsque le nombre d'actions hebdomadaires maximum est atteint ou que l'action 4 est sélectionnée. Il affiche, à l'aide des méthodes `Plante.Pousser()` et `Plante.ToString()`, l'état des plantes comprenant leur croissance, taille, maladies contractées et âge. Les plantes décédées sont également notifiées avec `Plante.VerifierSurvie()`.

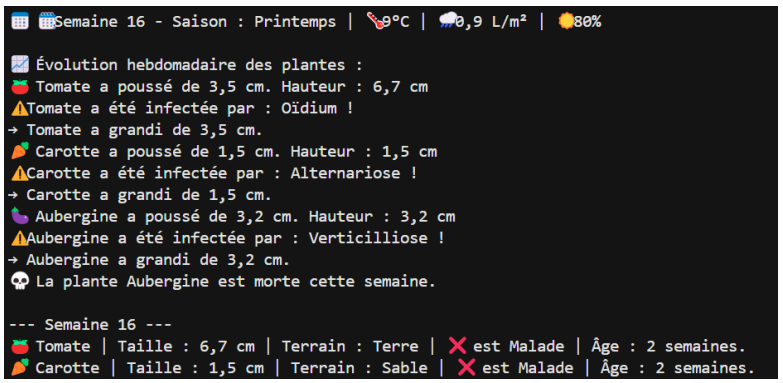

Figure 3 : Affichage du récapitulatif hebdomadaire

La Figure 4 illustre l'interface de l'action 2 d'arrosage d'une plante, comprenant un menu de sélection contenant la liste des plantes disponibles dans le potager.

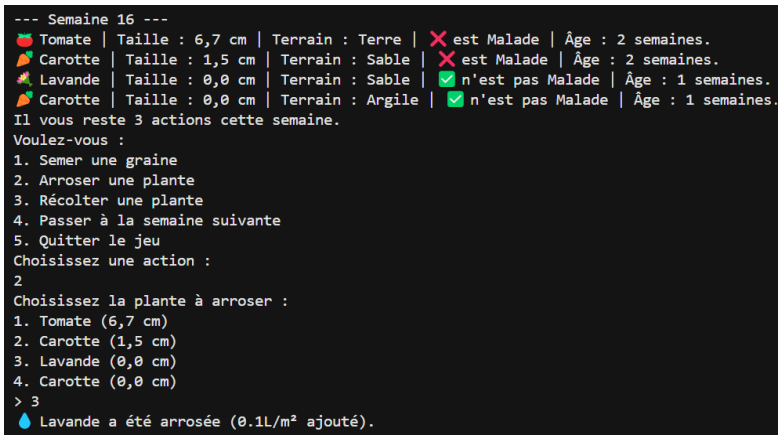

Figure 4 : Interface de l'action 2. Arroser une plante

La Figure 5 illustre l'interface de l'action 3 de récolte, comprenant un menu de sélection contenant la liste des plantes disponibles dans le potager et indiquant la possibilité ou non de récolter la plante choisie en fonction de ses conditions de maturité.

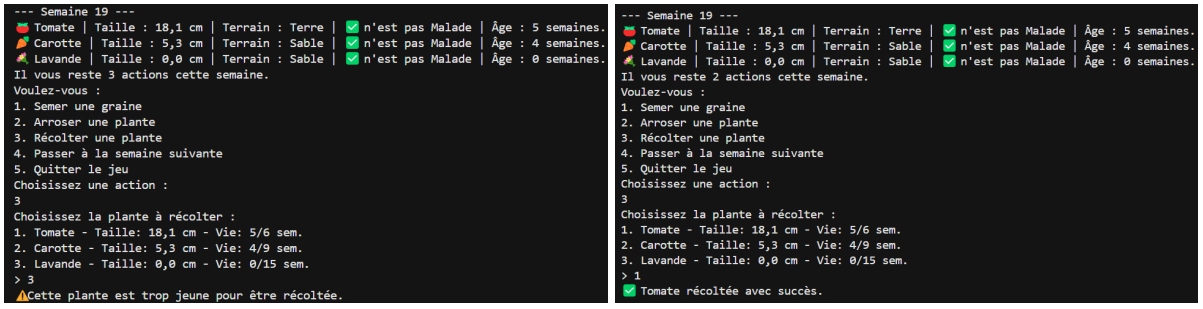

Figure 5 : Interface de l'action 3. Récolter une plante (à gauche : récolte impossible ; à droite : récolte effectuée)

La Figure 6 montre l'interface du mode urgence, déclenché lors d'une invasion aléatoire de rat, comprenant un message d'alerte, l'état de la plante grignotée par le rat à l'aide de la méthode `Plantes.Grignoter()`, suivie d'un menu de sélection des 5 actions d'urgence disponibles. L'action 2 permet d'ouvrir un menu composé des plantes disponibles dans le potager et d'en choisir 2 à protéger dans une serre. Les actions 1, 3, 4 et 5 correspondent respectivement à faire du bruit, à la pose d'une bâche de protection, d'un épouvantail et du cas extrême de « terre brûlée ». Une fois l'action choisie, l'effet sur le rat et la plante qu'il a ciblé est notifié dans la console. Cette ensemble est répétée au maximum 3 fois avant qu'un message de fin de mode urgence indiquant le départ du rat s'affiche.

```
⩓ALERTE !! ⩓
Un RAT s'est infiltré dans votre potager ! Il commence à ronger vos plantes !

🐀 Le rat rôde dans le jardin... (Tour 1/3)
Carotte a été grignotée ! Elle a perdu 1,5 cm...
🥕 Carotte | Taille : 8,4 cm | Terrain : Sable | ✅ n'est pas Malade | Âge : 7 semaines.
🌿 Lavande | Taille : 3,2 cm | Terrain : Sable | ✅ n'est pas Malade | Âge : 3 semaines.

Actions d'urgence possibles :
1. Faire du bruit (peut effrayer le rat)
2. Fermer la serre (protection temporaire de certaines plantes)
3. Déployer une bâche (rend toutes les plantes plus difficiles à grignoter)
4. Installer un épouvantail (retarde la prochaine invasion)
5. ☠ Terre brûlée ☠ : exterminez l'envahisseur au prix de vos plantations (il le mérite)
👉 Choisissez une action d'urgence :
```

```
1
📢 Vous avez fait du bruit ! Le rat s'est enfui !
✅ Fin de l'invasion de rats (pour cette fois). Restez vigilant !
```

```
2
🏠 Vous pouvez mettre 2 plantes à l'abri dans la serre.

Sélectionnez la plante à protéger :
1. Carotte (8,4 cm)
2. Lavande (3,2 cm)
> 2
✅ Lavande est protégée dans la serre.

Sélectionnez la plante à protéger :
1. Carotte (8,4 cm)
> 1
✅ Carotte est protégée dans la serre.

🐀 Le rat rôde dans le jardin... (Tour 2/3)
🛡️Lavande est à l'abri dans la serre !
Le rat n'a pas pu la grignoter.
🥕 Carotte | Taille : 8,4 cm | Terrain : Sable | ✅ n'est pas Malade | Âge : 7 semaines.
🌿 Lavande | Taille : 3,2 cm | Terrain : Sable | ✅ n'est pas Malade | Âge : 3 semaines.
```

```
3
🛡️La bâche limite les dégâts pour ce tour.

🐀 Le rat rôde dans le jardin... (Tour 2/3)
Fraise a été grignotée ! Elle a perdu 0,0 cm...
🍓 Fraise | Taille : 7,9 cm | Terrain : Terre | ✅ n'est pas Malade | Âge : 5 semaines.
🥒 Courgette | Taille : 1,0 cm | Terrain : Terre | ✅ n'est pas Malade | Âge : 1 semaines.
🥒 Courgette | Taille : 2,8 cm | Terrain : Terre | ✅ n'est pas Malade | Âge : 1 semaines.
```

```
4
🎃 Vous installez un épouvantail ! Moins de risques à l'avenir.

🐀 Le rat rôde dans le jardin... (Tour 3/3)
Fraise a été grignotée ! Elle a perdu 2,0 cm...
🍓 Fraise | Taille : 7,9 cm | Terrain : Terre | ✅ n'est pas Malade | Âge : 5 semaines.
🥒 Courgette | Taille : 2,8 cm | Terrain : Terre | ✅ n'est pas Malade | Âge : 1 semaines.
🥒 Courgette | Taille : 2,8 cm | Terrain : Terre | ✅ n'est pas Malade | Âge : 1 semaines.
```

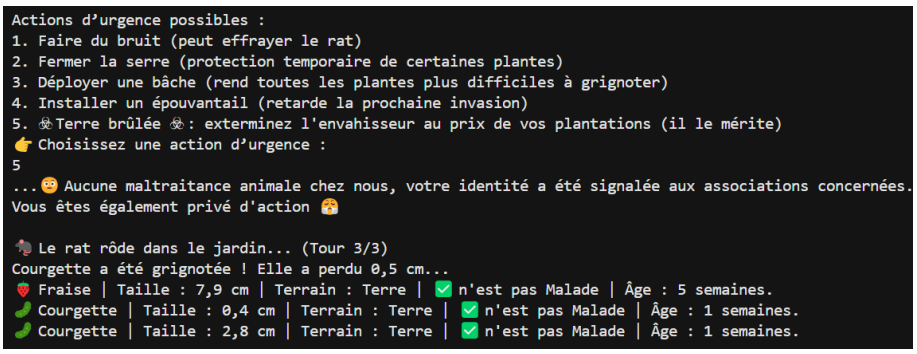

Figure 6 : Interface du mode urgence (de haut en bas : entrée dans le mode urgence puis action 1 à 5)

Enfin, la Figure 7 illustre la clôture du jeu avec l'affichage d'une synthèse de fin de partie, qui présente un récapitulatif des plantes récoltées par espèce.

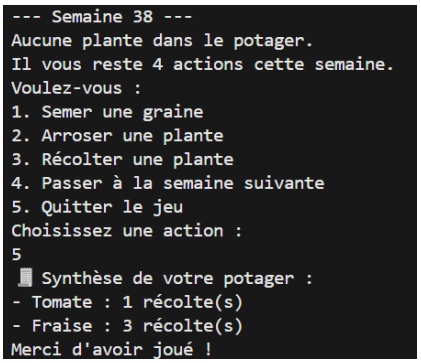

Figure 7 : Affichage de la synthèse des plantes récoltées en fin de partie

L'ensemble de ces interactions est encadré par des retours visuels clairs, des messages informatifs et des émojis intégrés, assurant une bonne lisibilité à chaque étape du jeu.

III - Structuration du code :

La structuration du code dans le projet ENSemenC suit une approche orientée objet, en cohérence avec les concepts introduits en cours de Programmation Avancée. Les entités manipulées dans le jeu, les comportements dynamiques et les événements du cycle de jeu sont représentés à travers des objets. L'organisation en classes permet de centraliser les comportements, d'éviter les duplications et de faciliter l'évolution du programme.

L'ensemble du code est réparti en fichiers spécialisés, chacun dédié à une responsabilité précise. L'interaction avec l'utilisateur, la gestion des états internes et des événements ponctuels sont ainsi gérés de manière modulaire et hiérarchisée. Le tout est commenté pour faciliter sa lecture ou modification par d'autres utilisateurs.

1) Structure globale

ENSemenC est structuré autour de 20 fichiers `.cs`, chacun correspondant à une classe du simulateur. Cette organisation reflète les entités et mécanismes du jeu, conformément aux principes de l'approche orientée objet. La logique de simulation est centralisée dans la classe `Game` du fichier `GameLoop.cs`, qui orchestre l'exécution du jeu tour après tour.

Les entités du jeu : plantes et terrains, sont chacune modélisées et gérées dans des fichiers dédiés sous forme de classes abstraites ou dérivées. Les saisons sont quant à elles modélisées et gérées dans la classe `Saison`, qui encapsule la gestion du temps et la génération des conditions climatiques hebdomadaires. Les interactions avec l'utilisateur sont intégrées directement dans les méthodes de la classe `Game`, en cohérence avec l'environnement console retenu.

La gestion des actions du joueur, des états environnementaux et de l'évolution des plantes est encapsulée dans des méthodes courtes et spécialisées, favorisant la clarté du code et la réutilisabilité. La structure repose également sur l'usage de méthodes communes pour garantir une cohérence des traitements généralisée, ainsi que de méthodes abstraites définies dans les classes abstraites de base `Plante` et `Terrain` pour permettre une redéfinition de comportement spécifique adaptée dans chaque sous-classe.

La structure générale du programme peut être résumée de la manière suivante :

- `Program.cs` : point d'entrée de l'application, instancie la classe `Game` et lance la simulation avec la méthode `Game.Start()`.
- `GameLoop.cs` : définit la classe principale `Game` qui contient l'ensemble des méthodes de déroulement d'une partie de jeu.
- `Plante.cs` + 12 classes filles (cf. II.1) : modélisation et encapsulation des comportements généraux et spécifiques des différentes espèces de plantes.
- `Terrain.cs` + 3 classes filles (cf. II.2) : modélisation et encapsulation des comportements généraux et spécifiques des différents types de sol.
- `Saison.cs` : modélisation, simulation et gestion du passage du temps ainsi que des conditions climatiques.

2) Initialisation et Affichage

Cette sous-partie décrit le déroulement technique de l'initialisation d'une partie et les mécanismes de gestion de l'affichage en console, à distinguer de la modélisation des données d'interface. Le code suit une logique chronologique : initialisation des variables, affichage d'introduction, déroulement des tours hebdomadaires, et mise à jour visuelle régulière de l'état du potager.

La méthode `Start()` de la classe `Game` constitue le point d'entrée fonctionnel du jeu. Elle commence par l'affichage d'un message de bienvenue, suivi d'un premier appel à `SimulerSemaine()`.

Les paramètres de simulation sont initialisés dès le constructeur de la classe `Game`, avant même le premier affichage :

- `private int semaine = 0` : compteur du temps global, incrémenté à chaque nouvelle semaine
- `private int nbActionsRestantes = 5` : nombre d'actions disponibles en début de semaine
- `private double probaInvasionRat = 0.05` : probabilité d'apparition du mode urgence
- Des listes vides :
  - `List<Plante> plantes` pour stocker les plantes en jeu
  - `List<int> semainesPlantation` pour enregistrer la semaine de semis de chaque plante
  - `List<string> nomsPlantesRecoltees` et `List<int> quantitesRecoltees` pour le suivi du nombre de plantes récoltés par espèce

L'affichage en console est réalisé via `Console.WriteLine()`, organisé en blocs visuellement distincts :

- Le numéro de la semaine : `$"\n--- Semaine {semaine} ---"`
- L'état du potager avec `AfficherEtatPlantes()`
- Les informations météorologiques avec `SimulerSemaine()`
- Les menus d'actions hebdomadaires

Les entrées utilisateur sont capturées par :

- `Console.ReadKey()` pour les menus d'action numérotés
- `Console.ReadLine()` pour les sélections plus détaillées, nécessitant d'entrée plus d'un caractère comme celui des plantes

Des conditions (`if`, `else if`, `else`) sont largement utilisées pour contrôler l'affichage des messages selon l'état du jeu (nombre d'actions restantes, validation ou refus d'une action, erreurs de saisie).

Le déroulement d'une semaine repose sur la variable `nbActionsRestantes`, décrémentée après chaque action. Lorsqu'elle atteint zéro, un message s'affiche automatiquement pour prévenir le joueur, et `SimulerSemaine()` est appelée pour basculer à la semaine suivante. Dans cette méthode, l'affichage des évolutions hebdomadaires repose sur un `foreach` parcourant la liste `plantes`. Pour chaque plante, trois types de conditions génèrent des messages spécifiques :

- Si la taille de la plante augmente après l'appel à `Pousser()`, une ligne signale sa croissance
- Si `estMalade` passe à `true` via `AttraperMaladie()`, une alerte est affichée
- Si `VerifierSurvie()` retourne `false`, ou si l'âge calculé via `private int AgePlante(index)` atteint `EsperanceDeVie`, la plante est supprimée et un message de décès est affiché

En parallèle, une autre condition vérifie si une invasion de rat doit être déclenchée : un tirage aléatoire via `rng.NextDouble()` est comparé à `probaInvasionRat`. Si la condition est remplie, `InvasionRat()` est appelée, ce qui suspend temporairement le cycle normal du jeu au profit de l'affichage du mode Urgence.

Ainsi, toute la gestion de l'affichage repose sur une alternance fluide entre lecture des données, conditions d'état et sorties visuelles structurées. Cela garantit à l'utilisateur une lecture claire et réactive des événements influençant son potager à chaque tour.

3) Gestion des actions

La gestion des actions s'organise autour d'un menu réaffiché à chaque tour de jeu, permettant au joueur de sélectionner l'une des cinq interactions possibles. Ce menu est implémenté dans la méthode `Start()` de la classe `Game`, à l'aide d'un bloc `switch` associé à la variable `choix`. Cette variable est alimentée à partir d'un caractère saisi via `Console.ReadKey()`. Selon la valeur obtenue, le programme redirige l'exécution vers la méthode correspondant à l'action choisie. Chaque action repose sur une méthode dédiée qui enchaîne les opérations prévues. L'ensemble du système s'appuie sur des appels à des méthodes spécifiques, des conditions de contrôle et de vérifications de validité de la saisie utilisateur via des `if`/`else`, assurant robustesse, clarté et évolutivité du code.

a) Action 1. Semer une plante

La méthode `SemerPlante()` commence par l'appel à `ChoisirPlante()`. Celle-ci affiche les douze plantes disponibles, attend l'entrée de l'utilisateur, et retourne une instance de la plante choisie. La méthode vérifie ensuite si la saison en cours permet le semis via `SaisonsSemis.Contains(saisonActuelle)`. En cas de validation, la méthode appelle `ChoisirTerrain()` pour sélectionner un type de sol.

Enfin, une confirmation utilisateur permet d'ajouter la plante à la liste `plantes` et d'enregistrer la semaine de plantation dans `semainesPlantation`. Chaque saisie utilisateur est captée avec `Console.ReadLine()` ou `ConsoleKeyInfo`, validée via des conditions `if`/`else`, et associée à un retour visuel adapté.

b) Action 2. Arroser une plante

La méthode `ArroserPlantes()` affiche les plantes en jeu à l'aide d'une boucle `for`, en itérant sur la liste `plantes` avec leur indice. Pour chaque plante, son nom et sa taille sont affichés. L'utilisateur saisit un entier correspondant à l'indice de la plante à arroser. Si la sélection est valide, la méthode appelle `Plante.AjouterEau(0.1)` sur le terrain associé à la plante. Cette méthode incrémente l'attribut `quantiteEauActuelle` du terrain. Un message de confirmation est ensuite affiché. En cas d'entrée invalide, une alerte est renvoyée.

c) Action 3. Récolter une plante

La méthode `RecolterPlantes()` utilise elle aussi une boucle `for` sur la liste `plantes`, et affiche chaque plante avec son nom, sa taille et son âge. L'âge de la plante est calculé à l'aide de la méthode `AgePlante(index)`, qui effectue la différence entre la semaine actuelle (`semaine`) et la valeur correspondante à l'index dans la liste `semainesPlantation`.

`RecolterPlantes()` appelle ensuite la méthode abstraite `Plante.PeutEtreRecoltee()` avec l'âge en paramètre pour vérifier si la plante à atteint la maturité suffisante pour être récoltées selon les conditions nécessaires spécifiques à l'espèce.

Si la plante est prête à être récoltée, elle est supprimée de la liste `plantes` avec `plantes.remove` et son nom est ajouté à la liste `nomsPlantesRecoltees`. Sa quantité est alors soit initialisée à 1 soit incrémentée dans la liste `quantitesRecoltees`. Si les conditions de maturités ne sont pas remplies, un message d'erreur est affiché.

d) Action 4. Passer à la semaine suivante

Lorsque l'utilisateur choisit cette action, la variable `nbActionsRestantes` est réinitialisée à 5 et la méthode `SimulerSemaine()` est immédiatement appelée. Cette méthode incrémente `int semaine`, génère les conditions climatiques de la nouvelle semaine et met à jour les plantes.

e) Action 5. Quitter le jeu

Cette dernière action passe la variable `bool continuer` à `false`, ce qui met fin à la boucle principale de jeu. Elle appelle également la méthode `AfficherSynthese()` qui affiche un récapitulatif des plantes récoltées. Cette méthode parcourt les listes `nomsPlantesRecoltees` et `quantitesRecoltees` à l'aide d'une boucle `for`, puis affiche chaque espèce avec le nombre de récoltes associées. Un message final de remerciement clôt l'expérience de jeu.

4) Simulation hebdomadaire

La méthode `SimulerSemaine()` pilote la progression temporelle du potager et coordonne les évolutions hebdomadaires de l'ensemble des éléments du jeu. Elle joue un rôle central dans la simulation en appelant, à chaque nouvelle semaine, les fonctions qui régissent le climat, la croissance, la maladie, la survie et les événements exceptionnels. Cette méthode est déclenchée soit automatiquement lorsqu'aucune action n'est restante, soit explicitement par le joueur via l'action "4. Passer à la semaine suivante".

La séquence commence par l'incrémentation de la variable `semaine`, suivie de l'instanciation d'un objet `Saison`, auquel sont passés le numéro de la semaine et un objet `Random`. Trois méthodes de la classe `Saison` sont ensuite appelées pour déterminer les conditions climatiques :

- `Saison.GenererTemperature()` : renvoie une température aléatoire adaptée à la saison courante.
- `Saison.GenererPrecipitations()` : calcule une valeur de précipitation hebdomadaire.
- `Saison.GenererLuminosite()` : retourne un pourcentage de luminosité moyen pour la semaine.

Ces valeurs sont affichées pour informer le joueur des conditions de la semaine, puis elles sont exploitées pour simuler les effets sur chaque plante.

La mise à jour des plantes repose sur une boucle `foreach` parcourant la liste `plantes`. Pour chaque plante :

- L'eau issue des précipitations est absorbée par le terrain associé via `Terrain.AjouterEau(precipitations)` appelé à travers la propriété `Plante.TerrainAssocie`. La méthode `Terrain.AjouterEau()` incrémente l'attribut `quantiteEauActuelle`, dans la limite autorisée par `Terrain.CapaciteEau`.
- Le taux de respect des conditions favorables est ensuite calculé via la méthode `Plante.CalculerTauxConditions(temperature, luminosite)`, qui combine cinq critères : adéquation du terrain avec le type préféré, taux d'humidité, température, luminosité, et état de santé de la plante. Chaque critère génère un score sur 1. L'ensemble étant pondéré pour déterminer le score globale des conditions de croissance.
- La méthode `Plante.Pousser(tauxConditions)` incrémente ou non la taille de la plante en fonction du taux calculé. La méthode `Plante.AttraperMaladie(Random rng)` est appelée pour déterminer si la plante contracte une maladie. Elle modifie l'état interne du booléen `estMalade` en fonction d'un tirage aléatoire.
- La méthode `Plante.VerifierSurvie(tauxConditions)` évalue si la plante survit à la semaine. Ce test dépend de son espérance de vie déterminée par l'attribut `Plante.EsperanceDeVie` dans la sous classe correspondant à l'espèce et du taux de conditions favorable calculé précédemment. Si la plante meurt, elle est retirée de la liste principale avec `plantes.Remove`.

5) Mode urgence : invasion de rat

Le mode urgence est une séquence spéciale déclenchée par la méthode `InvasionRat()` lorsque le résultat du tirage aléatoire effectué automatiquement à la fin de chaque action est inférieur à `probaInvasionRat`, initialisée à 0.05 en début de partie.

La méthode fonctionne en trois tours compressés dans la même semaine. À chaque tour :

- Un message d'alerte est affiché.
- Une plante est choisie aléatoirement via `rng.Next(plantes.Count)`.

- Si elle n'est pas protégée, elle subit l'effet de `Plante.Grignoter(Random rng, int bacheDeployee)` qui réduit sa taille de manière pseudo-aléatoire selon la présence ou non d'une bâche de protection.

Le joueur choisit ensuite une action via `Console.ReadKey()` :

- 1. Faire du bruit : provoque un tirage aléatoire. Si `rng.NextDouble()` est supérieur à 0.7, le rat prend la fuite et la méthode se termine prématurément.
- 2. Fermer la serre : permet de sélectionner jusqu'à deux plantes. Leurs indices sont stockés dans la liste `plantesProtegeesDansSerre`, consultée à chaque tour pour les immuniser des attaques du rat.
- Déployer une bâche : assigne la valeur `1` à la variable locale `bacheDeployee`, réduisant l'efficacité de `Grignoter()` ce tour-là.
- Installer un épouvantail : décrémente `probaInvasionRat` d'une valeur aléatoire entre 0 et 0.02 via `probaInvasionRat -= rng.NextDouble() * 0.02`.
- Terre brûlée : ne modifie aucun attribut du jeu. Elle bloque simplement le tour d'action du joueur avec un message symbolique.

Chaque décision affecte directement les variables du jeu. À la fin de chaque tour, la liste `plantesProtegeesDansSerre` est vidée et `bacheDeployee` est réinitialisée automatiquement à chaque début de tour. La méthode se poursuit jusqu'à un maximum de trois tours ou jusqu'à ce que le rat soit repoussé prématurément par une action efficace.

IV - Gestion de projet :

1) Planification et outils

Le projet ENSemenC a officiellement débuté le 1er avril 2025, date de publication des consignes. Cependant, la constitution du binôme a été retardée, Maxime UNG et Mathieu RECH ne parvenant pas à trouver de partenaire respectif avant le 14 avril. Ce démarrage effectif à mi-avril a réduit la durée de travail à six semaines, avec une date de rendu fixée au 23 mai.

Une première réunion a eu lieu le 16 avril afin de fixer les grandes lignes du projet, définir les choix de modélisation retenus, déterminer les priorités fonctionnelles et procéder à une répartition initiale des tâches. Malgré les quatre séances de TP encadrées prévues au cours de cette période, celles-ci n'ont pas pu être utilisées pour un travail collaboratif, les deux étudiants n'étant pas dans les mêmes groupes de TD.

Le travail s'est ainsi déroulé à distance, chaque développeur avançant en autonomie entre deux mises en commun. L'ensemble du développement a été réalisé localement sous Visual Studio Code avec l'environnement .NET 8.0. L'utilisation d'un dépôt GitHub a permis de faciliter la coordination, grâce à une gestion efficace du partage du code et de son versionnage. Maxime UNG

travaillait sur une branche personnelle à son nom, tandis que Mathieu RECH intervenait directement sur la branche principale. Une fois une fonctionnalité finalisée, elle était poussée sur GitHub pour être relue par l'autre membre. Dans le cas de Maxime UNG, ses apports étaient fusionnés dans la branche principale après validation par Mathieu RECH. En cas de conflits, ceux-ci étaient résolus en concertation, avant de procéder à des tests de la version retenue. Cette organisation permettait un développement parallèle contrôlé, avec un suivi régulier des ajouts et ajustements, discutés via WhatsApp ou lors de rencontres informelles à l'école.

Dans un contexte de fin de semestre particulièrement chargé, avec neuf projets à rendre dans des unités d'enseignement différentes, ENSemenC s'est retrouvé repoussé à plusieurs reprise dans le planning de travail respectif du binôme. Bien que sa charge de développement soit importante, il était aussi le dernier à devoir être rendu au niveau du calendrier. En conséquence, la grande majorité du code a été réalisée entre le 12 et le 23 mai, avec un retard de 48 heures sur la remise du justificatif technique, finalement déposé le 25 mai.

2) Répartition des tâches

La répartition des tâches a été convenue lors de la réunion du 16 avril. Maxime UNG a pris en charge la création des classes `Plantes` et Terrains, comprenant l'implémentation de leur comportement, ainsi que la gestion des conditions climatiques et des interactions entre plantes et environnement. Mathieu RECH s'est quant à lui concentré sur la construction de la boucle de jeu, en implémentant les différentes possibilités d'actions, tant pour le mode classique que pour le mode urgence.

Chaque membre a également effectué des relectures croisées, permettant d'identifier des incohérences ou erreurs potentielles, qui étaient corrigées soit directement, soit après concertation. Cette double validation a permis de mieux optimiser la qualité du code en essayant d'éviter au maximum les répétitions de code. Enfin, la rédaction du justificatif technique a été principalement assurée par Maxime UNG, avec relecture et ajustements ponctuels de la part de Mathieu RECH.

La matrice d'implication en tableau 1 synthétise la contribution de chacun à chaque tâche. Elle montre une forte spécialisation des compétences, mais aussi un travail de collaboration sur l'ensemble du contenu.

MATRICE D'IMPLICATION : PROJET PROGAV			
Nom du Projet :	**ENSemenC**		**date: 23/05/2025**
MEMBRES DE L'EQUIPE: UNG Maxime - GR1 et RECH Mathieu - GR2			
n°	**Tâche/Fonctionnalité/Fonction**	**Nom du/des Codeurs**	**Pourcentage de participation si tâche partagée**
1	Classes et sous-classes de Plante	Maxime UNG / Mathieu RECH	90 / 10
2	Classes et sous-classes de Terrain	Maxime UNG	
3	Gestion des conditions climatiques	Maxime UNG	
4	Interactions Plantes-environnement	Maxime UNG / Mathieu RECH	70 / 30
5	Affichage dynamique	Maxime UNG / Mathieu RECH	30 / 70
6	Boucle du jeu mode Classique	Maxime UNG / Mathieu RECH	20 / 80
7	Boucle du jeu mode Urgence	Mathieu RECH	
8	Rédaction justificatif	Maxime UNG	

Tableau 1 : Matrice d'implication du projet ENSemenC

Bilan et perspectives :

Le projet ENSemenC a permis de développer un simulateur de potager jouable, conforme aux principales attentes du cahier des charges. Les fonctionnalités essentielles comme la gestion des semis, le choix des terrains, l'évolution des plantes au fil des semaines, l'impact de la météo, l'apparition d'aléas et les actions hebdomadaires ont été implémentées. Le jeu propose une boucle cohérente avec une alternance entre temps d'action et retour sur l'état du potager. Le code est structuré selon une architecture orientée objet, permettant une séparation claire des responsabilités. L'utilisation de GitHub a facilité la collaboration, la révision croisée du code et le suivi de version de manière rigoureuse. L'ensemble a été conçu pour être lisible, maintenable et potentiellement réutilisable.

Malgré cette complétion technique et fonctionnelle, certaines limites subsistent. L'interface utilisateur reste réduite à une console texte, sans gestion de sauvegarde ni affichage graphique. Le mode "urgence" mériterait d'être approfondi, par exemple en localisant les invasions sur des terrains spécifiques pour nuancer leurs impacts. Sur le plan ergonomique, le jeu gagnerait à filtrer automatiquement les semis disponibles en fonction des saisons, à proposer une visualisation des fiches plantes avant plantation, ou encore à permettre le choix de la quantité d'eau à verser lors d'un arrosage. La notion de surface cultivable par terrain, bien que présente dans les attributs des plantes, pourrait aussi être exploitée pour limiter le nombre de plantations par parcelle. D'un point de vue technique, la gestion des exceptions pourrait être systématisée afin de renforcer la stabilité du programme.

Ces constats ouvrent la voie à plusieurs perspectives d'évolution du projet. L'ajout d'un système de sauvegarde/restauration permettrait de prolonger les parties sur plusieurs sessions. L'intégration d'une couche graphique dans une interface en mode fenêtre améliorerait l'expérience utilisateur. De nouveaux types de plantes et terrains pourraient être ajoutés, avec un calibrage plus fin de leurs caractéristiques pour simuler plus fidèlement le réalisme agronomique. Une composante économique pourrait également permettre d'enrichir la stratégie du joueur à travers un système de vente, troc ou extension du potager. Enfin, un mode compétitif ou collaboratif en ligne ouvrirait la voie à des dynamiques multijoueurs et renforcerait la rejouabilité. Toutes ces évolutions pourraient s'intégrer en capitalisant sur la structure objet déjà en place, rendant ENSemenC pérenne et extensible pour de futurs développements ou projets étudiants.