

BALACHANDRAN Vanathy
SURGET Lou-Ann



CO6SFPAO
Programmation Avancée

Document de justification technique
ENSEmenC



Pr. Angélique TETELIN

23 mai 2025

SOMMAIRE

Introduction.....	2
I. Univers du jeu.....	2
A. Contexte géographique et climat.....	2
B. Types de terrains et biodiversité.....	3
II - Déroulement d'une partie.....	4
A. Mode classique : gestion du potager semaine après semaine.....	4
B. Mode urgence : événements aléatoires.....	5
• L'invasion d'éléphants.....	6
C. Interface utilisateur et navigation.....	7
III - Explication des classes.....	7
A. Accueil.....	7
B. Plantes et ses classes dérivées.....	8
C. Jardin, Terrain et Case.....	8
D. JardinCurseur.....	9
E. Joueur et Inventaire.....	10
F. Boutique.....	11
G. Meteo.....	11
Bilan.....	12
Outil d'organisation.....	12

Introduction

Dans le cadre du projet du Semestre 6 en Programmation Avancée, nous avons réalisé un projet de simulation de potager en console, intitulé **ENSEMENC**. Ce jeu, entièrement codé en C#, plonge le joueur dans la gestion d'un jardin virtuel où il doit faire pousser différentes plantes tout en tenant compte des conditions de météo, de terrain et de croissance propres à chaque espèce.

Nous avons décidé d'ajouter un objectif principal à ce jeu : **partir de zéro et atteindre un certain nombre de pièces** en cultivant intelligemment, en récoltant les plantes au bon moment, et en vendant les produits dans une boutique intégrée au jeu. Pour enrichir l'expérience, nous avons intégré des mécaniques réalistes comme la météo dynamique, les préférences de terrain, les besoins en eau et la croissance progressive des plantes selon leur environnement.

Nous avons travaillé en binôme, en répartissant les tâches de manière équitable. Nous avons collaboré tout au long du projet, en se répartissant le développement des classes principales, la gestion de l'affichage console et l'implémentation de la logique de jeu. Grâce à une organisation fluide et à un partage clair des responsabilités, nous avons pu construire un jeu fonctionnel, lisible et conforme aux attentes pédagogiques.

Ce rapport détaille l'univers du jeu, les fonctionnalités développées, la modélisation objet utilisée, ainsi que notre démarche de test et notre retour critique sur le projet.

I. Univers du jeu

A. Contexte géographique et climat

Notre jeu, ENSEMENC, prend place dans un univers tropical inspiré du **Sri Lanka**, île réputée pour sa biodiversité, sa végétation luxuriante et son climat alternant moussons et saisons sèches. Ce choix d'univers donne une saveur réaliste et pédagogique à notre simulateur de potager tout en nous permettant d'intégrer des contraintes climatiques concrètes et variées.

Au début de la partie, le joueur choisit la saison de départ parmi les quatre saisons symboliques : Printemps, Été, Automne ou Hiver. Ce choix impacte immédiatement le comportement du jeu, notamment la météo qui évolue au fil des semaines. La température, l'humidité, le vent et la probabilité d'intempéries sont générés dynamiquement selon cette saison choisie, influençant directement la croissance des plantes.

Le Sri Lanka, situé dans une zone équatoriale, présente des conditions agricoles contrastées selon les périodes de l'année. Nous avons cherché à reproduire cela via des plages de température différentes, des taux d'humidité variables, et des occurrences plus ou moins fréquentes de pluie.

Cette dimension climatique impose au joueur de choisir ses plantations intelligemment en fonction des saisons et des conditions environnementales. Il doit surveiller les besoins spécifiques de chaque plante (en eau, en chaleur, en type de sol), et s'adapter pour maximiser ses récoltes tout en évitant que ses plantes ne dépérissent.

B. Types de terrains et biodiversité

Le potager se compose de six parcelles de terrains de types différents (sable, terre, argile), distribuées aléatoirement au lancement de la partie. Chaque type de plante a un terrain préféré, et si elle est plantée ailleurs, elle poussera plus lentement ou pourra mourir si les conditions sont trop défavorables.

Notre jeu propose plusieurs types de plantes à cultiver, chacune avec des caractéristiques uniques, des besoins spécifiques, et des comportements différents selon leur environnement.

Parmi les espèces disponibles dans notre version du jeu, on trouve :

- **Tomate** . 🌱🍅 : plante annuelle comestible, à croissance rapide, nécessitant un terrain de terre et beaucoup de soleil.
- **Aubergine** . 🌱🍆 : plante comestible occupant plusieurs cases, adaptée à l'argile, à planter en automne.
- **Mangue** . 🌱🌳🥭 : arbre fruitier vivace qui prend beaucoup de place (9 cases), adore les températures élevées et le soleil direct.
- **Hibiscus** . 🌱🌺 : plante ornementale, non comestible mais vendable, qui pousse sur du sable quelle que soit la saison.
- **Thé** . 🌱💧 : plante vivace demandant beaucoup d'eau, avec un cycle de croissance très long.

Chaque plante a :

- une **ou plusieurs saisons de semis** idéales,
- un **terrain préféré** (terre, argile, sable),
- des **besoins hydriques** précis,
- une **espérance de vie** et une **vitesse de croissance**,
- et une **taille** (nombre de cases qu'elle occupe) précisée au moment de semer la graine.

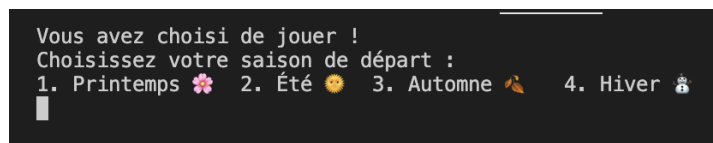
Le joueur doit donc gérer judicieusement son espace, ses ressources, et le calendrier des semis pour optimiser ses récoltes et atteindre l'objectif fixé.

II - Déroulement d'une partie

A. Mode classique : gestion du potager semaine après semaine

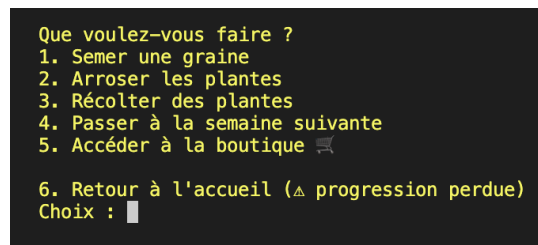
La méthode `Jouer()` permet à l'utilisateur de lancer une partie en initialisant la saison, créant les objets `Meteo`, `Jardin`, `JardinCurseur`, `Joueur` (avec un inventaire prérempli) et `Urgence`, puis entrant dans une boucle se répétant tant que le joueur n'a pas accumulé 100 pièces (`objectifArgent`).

Chaque partie du jeu débute par le choix d'une saison (printemps, été, automne ou hiver), qui influencera directement les conditions météorologiques durant la partie.



Le jeu se déroule ensuite jusqu'à atteinte de l'objectif, sur plusieurs semaines, découpées en cycles saisonniers (4 semaines par saison).

À chaque semaine, une météo est générée aléatoirement selon la saison courante, avec des variations de température, humidité, vent et conditions (pluie, nuage, soleil).



Le joueur peut parcourir le jardin, voir les caractéristiques d'une plante (croissance, température, terrain, hydratation) et dispose de plusieurs actions stratégiques possibles à chaque tour :

- **Semer une graine**

Lorsque le joueur appuie sur **1**, l'inventaire est filtré pour ne conserver que les objets « graine », puis chaque type de graine disponible est affiché avec sa quantité. Le joueur saisit le numéro correspondant à la graine qu'il souhaite planter. Une fois la graine sélectionnée, le joueur déplace librement le curseur pour choisir un emplacement.



Si `curseur.PeutPlanter(planteChoisie)` renvoie faux, un message d'erreur détaille le nombre de cases nécessaires et l'on revient à la sélection d'emplacement.

Quand l'emplacement est validé, `curseur.Planter(planteChoisie)` place l'objet au bon endroit, et l'inventaire du joueur est mis à jour via `joueur.Inventaire.SemerGraine`, décrémentant le stock de graines.

- **Arroser une plante**

En pressant **2**, les plantes peuvent être arrosées : le joueur déplace à nouveau le curseur sur la case cible, puis si une plante s'y trouve, la méthode `plante.Arroser()` est invoquée pour augmenter son niveau d'hydratation, en précisant que toutes les cases occupées par la plante ont été arrosées. Si aucune plante n'est détectée sous le curseur, un message informe qu'il n'y a rien à arroser.

- **Récolter une ou plusieurs plantes devenues mures / nettoyer le jardin**

La touche **3** déclenche la récolte : la méthode `jardin.InventaireRecoltes(joueur.Inventaire)` parcourt toutes les cases de chaque terrain pour identifier les plantes mures, les enlève du jardin (la récolte) et ajoute les quantités correspondantes à l'inventaire du joueur et retourne un dictionnaire `nom→quantité`. Immédiatement après, `jardin.NettoyerPlantesMortes()` élimine du jardin les plantes dont la phase indique "morte", afin de libérer les cases.

- **Passer à la semaine suivante.**

Choisir **4** fait passer la semaine suivante : `jardin.ToutPousser(meteo, saison, 7)` fait évoluer chaque plante selon les paramètres climatiques, tandis que `BaisserHydratationPlantes(jardin)` soustrait 20 points d'eau à chacune (évaporation). Enfin, le compteur de semaines est incrémenté et la boucle interne se termine pour revenir afficher un nouvel état.

- **Accéder à la boutique** pour vendre ses récoltes ou acheter de nouvelles graines,

La touche **5** ouvre la boutique via `new Boutique().Afficher(joueur)`, donnant accès à l'achat de graines ou d'outils et à la vente des récoltes, avec mise à jour instantanée de l'argent et de l'inventaire. (cf. III - F)

Toutes ces actions ont un impact direct sur la progression vers l'objectif final.

Dès que le solde du joueur atteint ou dépasse 100 pièces, la boucle s'interrompt, un message de félicitations s'affiche en vert, puis l'application revient automatiquement à l'accueil, prête à lancer une nouvelle partie ou à se fermer.

Si le joueur souhaite abandonner sa partie en cours, il peut appuyer sur **6** : un avertissement s'affiche avant un retour vers l'écran d'accueil (`AfficherPageAccueil()`), supprimant toute progression.

B. Mode urgence : événements aléatoires

Entre deux semaines, un événement spécial peut survenir sous forme de mode urgence. Ce mode est déclenché de manière aléatoire (1 chance sur 7) et se traduit par un affichage clignotant à l'écran.

Les événements sont gérés par une classe `Urgence`, qui contient les méthodes pour exécuter les différents événements mais également la méthode `AfficherPageUrgence()` afin d'afficher le titre clignotant du mode URGENCE.




- **L'invasion d'éléphants**

Des éléphants viennent d'entrer sur votre propriété !!

Le joueur est alors confronté à un choix risqué :

- Tenter de faire fuir les éléphants en leur faisant peur (50 % de chance de réussite et si échec l'ensemble du potager est détruit),
- Ou les laisser passer, auquel cas entre 25 % et 50 % de ses plantes seront piétinées.

Cet événement est géré par deux méthodes : `Elephant()` qui affiche un dialogue avec le joueur, gère les probabilités et modifie dynamiquement l'état du jardin et

`AnimationElephant()` qui, comme son nom l'indique, permet d'animer le déplacement de l'éléphant , en écrasant visuellement les plantes en temps réel.

`Elephant()`

Le joueur est interrogé (oui/non) via la console. S'il répond "oui", un `Random.Next(1, 3)` décide du sort :

```
Des éléphants ont réussi à entrer sur votre propriété
Ils vont écraser certains de vos plants !! Voulez vous lui faire peur et ainsi sauvez vos plants ??
(Si oui vous prenez le risque qu'ils s'énervent et écrasent TOUTES vos plantations !)
```

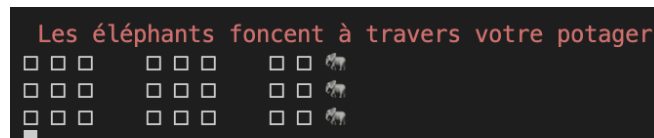
- **Succès** : tous les plants sont sauvés.
- **Échec** : toutes les plantes sont détruites (`AnimationElephant()` est appelée).

S'il répond "non", seules quelques plantes sont détruites. Une carte des plantes est construite (`Dictionary<Plantes, List<coordonnées>>`) et un nombre aléatoire de plantes (25–50 %) est choisi puis supprimé du jardin.

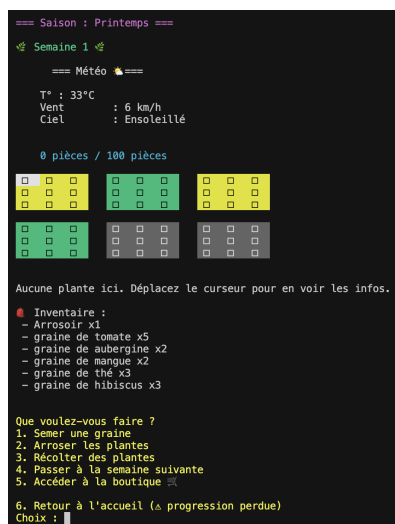
Ce traitement se fait en profondeur, via un accès direct aux **Terrain.Cases**.

`AnimationElephant()`

Elle balaye tous les `Terrains` et toutes les `Cases` et efface les plantes au passage. Un effet de mouvement est créé en effaçant et réaffichant la console (`Thread.Sleep(400)`). Cela donne un effet visuel dynamique, directement synchronisé avec les conséquences sur les données du jeu.



C. Interface utilisateur et navigation



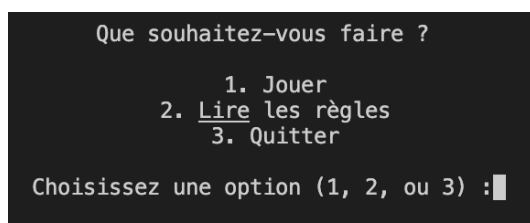
Le jeu repose sur une interface console intuitive, centrée sur la lisibilité et l'accessibilité.

Le potager est affiché grâce à des cases et le joueur peut déplacer un curseur grâce aux flèches du clavier. Nous avons également ajouté des messages informatifs et des retours visuels colorés.

À chaque tour, le jeu affiche le potager, la météo de la semaine, l'argent du joueur, l'objectif global et le menu d'actions.

III - Explication des classes

A. Accueil



La classe **Accueil** prend entièrement en charge l'interface utilisateur et le déroulement du jeu dès le lancement : la méthode `AfficherPageAccueil` propose à l'utilisateur de commencer une partie (`Jouer()`) lire les règles (`LireRegles()`) ou bien de quitter l'application (`Quitter()`).

`Jouer()` cf. II - A

À chaque semaine, `GenererMeteo` fixe température, humidité, vent et condition (Ensoleillé, Nuageux, Pluie) selon des plages et probabilités propres à la saison calculée par `ObtenirSaison`.

Dans la boucle principale, l'écran est rafraîchi pour afficher la saison, le numéro de semaine, la météo via `meteo.Afficher()`, le solde et l'inventaire du joueur.

`LireRegles()` présente en texte clair l'ensemble des mécanismes (saisons, potager, semis, phases de croissance, météo, actions disponibles et urgences) puis permet de retourner à l'écran d'accueil

`Quitter()` efface l'écran, remercie le joueur et ferme l'application

B. Plantes et ses classes dérivées

La classe abstraite `Plantes` définit le socle commun à toutes les espèces cultivables : elle regroupe à la fois les paramètres statiques (nom, saisons et terrains préférés, cycle de vie, besoins en eau et température, surface occupée via la propriété abstraite `Occupation`) et les états dynamiques (phase de croissance, jours cumulés depuis le semis, niveau d'hydratation, cycles de repousse pour les vivaces).

Le constructeur initialise la phase à « Graine » et le niveau d'eau à 0, tandis que la méthode protégée `InitCycles()` calcule, si la plante est vivace, le nombre de récoltes possibles à partir de l'espérance de vie et du temps de maturité.

Les sous-classes concrètes comme `Tomate`, `Mangue`, `Aubergine`, `Thé` et `Hibiscus` définissent les valeurs spécifiques à chaque type de plante.

`Grandir()` :

Méthode centrale qui simule la progression de la plante sur plusieurs jours. Elle tient compte de la météo (saisons, pluie, température), de l'hydratation de la plante, du terrain et des stades de croissance (en fonction des jours écoulés).

`Arroser()` : augmente l'hydratation des plantes

`Recolter()` :

Permet de déclencher un nouveau cycle de croissance pour les plantes vivaces. Elle réinitialise partiellement la plante si elle peut repousser.

`Croissance()` :

Propriété visuelle utilisée pour l'affichage des différents émojis (phases de la plantes)

C. Jardin, Terrain et Case

La classe `Jardin` contient une grille 2D de `Terrain` (2×3), chacun contenant une grille 3×3 de `Case`. Chaque `Case` peut contenir une `Plante`.

- `Terrain` a un `TypeTerrain` (enum : `Sable`, `Terre`, `Argile`)
- `Case` contient une plante ou est vide (`null`). Elle renvoie le symbole "□" si `Plante == null`, sinon la chaîne `Plante.Croissance`
- `Jardin` gère la pousse (`ToutPousser`) et la récolte (`InventaireRecoltes`)

Le constructeur de la classe `Jardin` mélange aléatoirement deux terrains de chaque type pour varier la disposition et construit les terrains.

`ObtenirToutesLesPlantes()` récupère la liste de toutes les instances de `Plantes` distinctes présentes dans le potager.

`ToutPousser(Meteo meteo, string saison, int jours = 1)` parcourt chaque `Terrain`, chaque `Case` et, si `Case.Plante != null`, appelle `Grandir(...)` avec la météo, la saison, le type de sol et le nombre de jours à simuler entre chaque passage de semaine

`InventaireRecoltes(Inventaire inventaire) : Dictionary<string,int>` parcourt chaque `Terrain` et chaque `Case`

- Pour chaque plante mature et non encore récoltée, vérifie qu'on en est à l'origine (case de référence selon `Occupation`)
- Ajoute `Occupation.Count` exemplaires dans l'`inventaire` et dans le dictionnaire de retour
- Appelle `plante.Recolter()` :
 - si `false`, supprime définitivement la plante de toutes ses cases (annuelles ou épuisées)
 - si `true`, la laisse repasser en phase "En croissance" (vivaces)

`NettoyerPlantesMortes()` parcourt tous les terrains et cases ; si `Plante.Phase == "Morte"`, remplace `Case.Plante` par `null` pour libérer la case

D. JardinCurseur

Cette classe permet au joueur de naviguer dans le jardin à l'aide des flèches du clavier et de sélectionner des cases pour planter, arroser ou examiner les plantes. Elle affiche visuellement le potager avec des couleurs de fond selon le type de terrain.

Elle instancie les champs `jardin` (de la classe `Jardin`, en référence au potager), les indices du terrain (`terrain X`, `terrain Y`) ainsi que les coordonnées à l'intérieur du terrain (`caseX`, `caseY`).



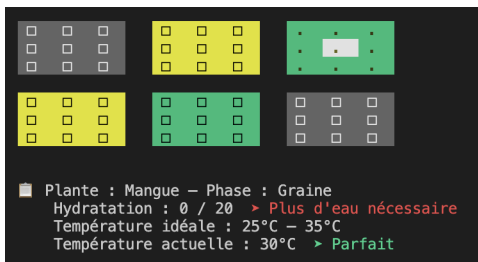
Le constructeur associé à la classe initialise le curseur en haut à gauche et lie l'instance de `Jardin`

`Deplacer(bool instructions, Plantes? plante)`

Lance une boucle de navigation où l'utilisateur déplace le curseur avec les flèches et valide par Entrée. Si `instructions=true`, affiche aussi un texte explicatif adapté à l'espèce (surface occupée, sol préféré).

`DeplacerUneFois(ConsoleKey key)`

Fait avancer le curseur d'une case selon la flèche appuyée, en sautant automatiquement d'un terrain à l'autre lorsqu'on dépasse un bord.



`AfficherInfosSousCurseur()`

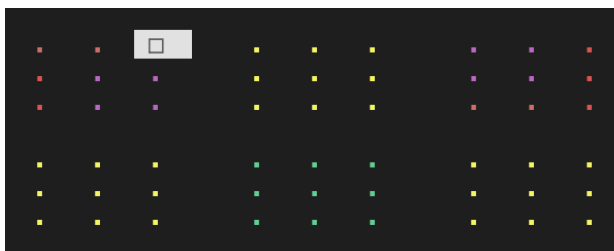
Lit la plante (ou l'absence) sous le curseur puis, si elle existe, affiche son nom, sa phase, son hydratation et la comparaison température idéale vs actuelle.

`BoucleDeplacement(Action afficher, ConsoleKey toucheQuitter)`

rafraîchit en continu l'affichage (`afficher()`), la lecture de touches (flèches ou touche de sortie) et la mise à jour des coordonnées jusqu'à la validation.

`Afficher()` parcourt tous les terrains et leurs 3x3 cases pour dessiner la grille en console :

- couleur de fond selon le type de terrain : **Sable**, **Terre** et **Argile**



couleur du texte selon la plante (définie dans les classes `Plantes`),

Tomate, **Aubergine**, **Hibiscus**, **Thé** et **Mangue**.

surlignage gris de la case sous le curseur,

- espacement uniforme grâce à `FormatCellule`.

`ObtenirCase()` retourne directement l'objet `Case` situé sous la position actuelle du curseur.

`ObtenirPlante()` renvoie la `Plantes?` attachée à la case active (ou `null` si aucune).

`PeutPlanter(Plantes plante)` vérifie que toutes les cases ciblées par `plante.Occupation` sont dans les limites du terrain et inoccupées, afin d'autoriser la plantation.

`Planter(Plantes plante)` : si `PeutPlanter` renvoie vrai, positionne l'objet `plante` sur chacune des cases définies par sa liste `Occupation`.

E. Joueur et Inventaire

La classe `Joueur` représente l'acteur principal du jeu : le jardinier. Il possède un inventaire d'objets (`Objet`) et une certaine somme d'argent.

Les attributs principaux sont :

- `Argent` : quantité d'argent possédée, initialisée à 0
- `Inventaire` : objet de type `Inventaire`, qui stocke toutes les graines, outils et produits récoltés.

```
Inventaire :  
- Arrosoir x1  
- graine de tomate x5  
- graine de aubergine x1  
- graine de mangue x1  
- graine de thé x3  
- graine de hibiscus x3
```

Lorsque le joueur est dans la boutique `AjouterArgent()` et `DepenserArgent()` sont appelés lors des transactions.

Le joueur peut donc gérer sa stratégie d'investissement : planter pour vendre, acheter pour diversifier, ou économiser pour atteindre les 100 pièces nécessaires à la victoire.

F. Boutique

La classe `Boutique` permet au joueur d'acheter des graines et de vendre ses récoltes, ce qui constitue la mécanique économique centrale du jeu. Elle offre une interface console stylisée et intuitive.

```
🛒 BOUTIQUE DU JEU  
💰 Votre argent : 0 pièces  
Que souhaitez-vous faire ?  
1 Acheter  
2 Vendre  
3 Revenir au jeu  
Votre choix : 1
```

`Acheter()` : Un catalogue statique propose plusieurs graines avec leurs prix, ainsi que l'argent que possède le joueur et un retour visuel en couleur. `Vendre()` : Le joueur choisit une récolte (toutes sauf les graines) et la quantité à vendre.

Le prix unitaire est fixé par type de plante (plus la plante est produite (ex: la mangue se récolte par 9), plus le prix est faible). L'argent gagné est ajouté via `joueur.AjouterArgent(prix*quantité)`.

La boutique utilise une mise en forme ASCII stylisée (cadres et émojis), ainsi que des couleurs pour les succès et erreurs. Les choix sont clairs grâce aux indices (12←) afin de faciliter la navigation.

G. Meteo

La météo influence la pousse des plantes. Elle est composée de plusieurs attributs : Condition (Ensoleillé, Nuageux, Pluie), Température, Vent et Humidité. Elle change chaque semaine selon la saison via `GenererMeteo()`.

Bilan

Outil d'organisation

Le travail collaboratif s'est fait grâce à Git, la plateforme GitHub et nous n'avons pas utilisé de branches contrairement au projet du S5.

Répartition des tâches dans le groupe

Lors des premières séances de TP en classe, nous nous sommes mis d'accord sur le pays où se trouvera notre potager, les différentes plantes que nous utiliserons et les différentes mécaniques du jeu (des exemples d'urgence, etc...).


Par la suite, nous nous sommes divisé le travail. Vanathy s'est occupée de toute la partie sur les plantes : la classe Plante ainsi que toutes ses classes dérivées et Lou-Ann a commencé les classes Jardin, Terrains etc...

Une fois que ces deux parties ont été réalisées, nous avons pu continuer sur la simulation du jeu et l'affichage générale (Lou-Ann), l'implantation des différentes méthodes comme semer, arroser, récolter (Vanathy), la croissance des plantes (Vanathy), ou encore la création du mode urgence (Lou-Ann).

Enfin, nous avons pu coder la partie météo, saisons et finir la croissance des plantes (Vanathy) et coder la boutique et la gestion de l'argent (Lou-Ann).

Par la suite, nous nous sommes occupés équitablement du visuel du jeu (couleurs, dessins ASCII, émojis), de la rédaction des règles, de l'affichage et de l'organisation du code. Dans les jours précédant la date butoire, nous avons commencé la rédaction de ce rapport et l'avons modifié progressivement en fonction des changements dans le code.

La répartition plus détaillée est visible dans notre matrice d'implication :

 MI Projet ENSEMENC

Date	Tâche réalisée
11/04/2025	Configuration de GitHub
14/04/2025	Création des classes
	Première grille du potager
22/04/2025	Classes Jardin et Plantes
05/05/2025	Suppression et réinitiation du git avec .gitignore
	Classes Plantes et Météo
	Page d'Accueil
	Création Inventaire
	Nouvelle grille du potager (curseur) + déplacement avec flèches et ZSDQ
13/05/2025	Déplacement avec flèches
	Croissance des plantes en fonction des phases
	Méthode Planter
	Présentation de la page d'accueil
	Début Mode Urgence
16/05/2025	Affichage du potager corrigé
	Méthode Récolter + ajout Inventaire
17/05/2025	Urgence Éléphant
	Méthode Arroser
	Plusieurs actions par semaine pour le joueur

18/05/2025	Affichage info plante avec curseur
	Niveau d'hydratation + Croissance en fonction de l'hydratation
À partir du 19/05/2025	Rédaction du rapport
	Mise à jour des messages d'affichage
	Correction Croissance Plantes
23/05/2025	Création Boutique
	Implémentation Jeu avec pièces et objectif
	Type de Terrains + Saisons + Croissance adaptée aux deux
	Gestion Plantes Vivaces et plantes mortes
	Finalisation rapport

Bien que le jeu soit déjà fonctionnel, plusieurs pistes d'amélioration ont été identifiées pour enrichir le gameplay, augmenter la rejouabilité, et améliorer l'ergonomie générale.

- **Sauvegarde de la partie** : L'ajout d'un système de sauvegarde et de chargement permettrait aux joueurs de poursuivre leur progression sur plusieurs sessions.
- **Ajout de nouvelles urgences** : Aujourd'hui, seule une urgence aléatoire liée à des éléphants est implémentée. Ce système pourrait être considérablement enrichi par l'introduction de nouveaux événements extrêmes comme des moussons, sécheresses, vents violents (perte aléatoire de plantes), ou des cyclones (destruction des plantes).
- **Gestion de plantes malades** : certaines plantes pourraient tomber malades selon la météo ou les excès/déficits d'eau. Cela impliquerait des soins ou traitements spécifiques et rendrait l'entretien du jardin plus complexe. Cela pourrait aussi introduire de nouveaux objets à utiliser (engrais, remèdes...).
- **Objectifs secondaires et complexification du but principal** : Le jeu actuel repose sur un seul objectif : atteindre 100 pièces. Pour enrichir le challenge, on pourrait introduire des défis hebdomadaires, des succès à débloquent, ajouter un chronomètre (qui atteint l'objectif le plus rapidement ?)

