Projet ENSemenCe

Soconusco, Mexique

Hokkaido, Japon

Carcassonne, France

Camille Vignes et Thiméo Tonon

Table des matières

Introduction

Qui n'a jamais rêvé de gérer un jardin abondant ou une ferme fructueuse ? Grâce à notre adaptation du projet de programmation ENSemenC, nous rendons cela accessible à tous. En effet, notre jeux codé en C# vous permet de gérer votre potager aux trois coins du monde : en France à Carcassonne , au Mexique à Soconusco et au Japon à Hokkaido.

Dans le cadre de ce projet, nous avons veillé à respecter au mieux les consignes définies dans le cahier des charges fourni par Mme Tételin, tant sur le plan de la structure des classes que sur celui des fonctionnalités attendues. Le développement de notre jeu de potager a été réalisé en langage C#, en adoptant une approche de programmation orientée objet. Conformément aux directives, nous avons utilisé la plateforme GitHub afin d'assurer une collaboration efficace, en fusionnant régulièrement nos différentes versions du code et en suivant les évolutions du projet de manière rigoureuse. Durant les deux mois impartis, nous nous sommes investis pleinement et de manière équilibrée, mettant tout en œuvre pour donner vie à ce jeu. Si certaines fonctionnalités ambitieuses n'ont pas pu être intégrées, faute de temps ou en raison de leur complexité, nous sommes néanmoins très fiers du travail accompli. Ce projet représente le fruit de notre engagement, de notre apprentissage progressif et de notre envie de produire un résultat à la hauteur des attentes.

Dans ce rapport, nous allons vous présenter notre gestion du projet, les spécificités de notre univers et notre interprétation des consignes. Puis, nous allons détailler plus en profondeur notre code. Enfin, nous ferons un bilan de ce projet.

Gestion de projet

Afin de mener à bien ce projet, nous avons commencé par faire de multiples réunions pour dépoussiérer les consignes, pour clarifier nos objectifs et pour être sûr de savoir quelle direction prendre concernant l'affichage et notre gestion de classes. Afin de rassembler toutes nos idées, notre interprétation des consignes, nos pense-bêtes et les caractéristiques propres aux villes sélectionnées, nous avons utilisé Google Drive.

Nous avons également utilisé GitHub comme plateforme centrale pour la collaboration et la gestion des versions tout au long du projet. Cet outil a permis de documenter clairement les modifications apportées au code. En effet, nous avons particulièrement fait attention aux commit. Grâce à GitHub, nous avons suivi les différentes étapes de développement, et nous nous sommes assurées d'une synchronisation efficace entre les contributions individuelles de chaque membre de l'équipe. La fonctionnalité de branches a été essentielle pour tester des fonctionnalités avant leur fusion dans la version principale "origin main". Nous avons utilisé les branches Camille (modifiée par Camille) et Thiméo (modifiée par Thiméo). Nous avons également utilisé les outils intégrés pour la résolution des conflits lors de merge trop trépidents. Grâce à la revue de code directement dans l'interface, nous avons renforcé la qualité et la stabilité de notre programme.

Nous nous sommes répartis le travail en plusieurs phases. Nous avons débuté par une grande phase de recherche et de réflexion (3 semaines). Puis, nous avons commencé à coder l'interface de jeu et l'affichage. Ensuite, une fois l'affichage décidé et abouti, nous nous sommes concentré sur le mode normal et nous avons chacun codé des classes différentes pour fluidifier le travail. Nous avons fait en sorte que le nom de nos classes et de nos méthodes soit le plus clair possible pour se comprendre le plus facilement et pour faciliter la tâche aux correcteurs. Lors de chaque mise en commun, nous nous fixions des objectifs à réaliser pour la prochaine fois. Enfin, nous aboutissons au rendu final du projet en codant le mode urgence, dérivé du mode normal.

Voici notre matrice d'implication qui retrace chaque investissement des membres du groupe. Nous aboutissons à un total approximatif de 50% chacun. Nous avons en effet veillé à ce que le travail soit équitablement réparti entre nous.

MATRICE D'IMPLICATION : PROJET IPROG				
projet : ENSemenCe			version:	date: 22/05/2025
MEMBRES DE L'EQUIPE: Camille VIGNES, Thiméo Tonon				
n°	Tâche/Fonctionnalité/Fonction	Nom du/des Codeurs	Pourcentage de participation si tâche partagée	Pourcentage du Projet Global
1	Affichage et zones d'affichage	Camille, Thiméo	30%/70%	
2	Terrain	Thiméo	100%	
3	Plantes	Camille, Thiméo	70%/30%	
4	Parcelle	Camille, Thiméo	40%/60%	
5	Session de jeu et joueur	Camille, Thiméo	20%/80%	
6	Meteo	Camille, Thiméo	80%/20%	
7	Nuisibles	Camille	100%	
8	Outils	Camille, Thiméo	80%/20%	
9	Plantes	Camille, Thiméo	60%/40%	
10	Recoltes	Camille/Thiméo	30%/70%	
11	Interface	Thiméo	100%	
12	Réunions de clarification et d'avancement	Camille/Thiméo	50%/50%	
13	Rapport	Camille/Thiméo	70%/30%	
14				
15				
REMARQUES:				
Total : Camille : 48% // Thiméo : 52%				

Univers et Spécificité de notre code

🚨

Notre jeu a été conçu pour être joué sur un terminal prenant en compte l'affichage des emojis, vérifiez que le votre les affiche bien

🚨

Notre jeu possède un grand nombre de spécificités liées à son univers. Nous avons d'abord voulu rendre un travail le plus réaliste possible tout en se restreignant à cause de la limite temporelle induite. En effet, le joueur peut faire son potager dans trois villes correspondantes à trois régions du monde très différentes par leur spécificité. En effet, un grand nombre de paramètres vari selon les régions : la température, les saisons, l'humidité, les plantes, les nuisibles,... Nous avons approfondi au maximum nos recherches afin de faire correspondre au mieux les spécificités de chaque ville dans notre jeu. Par exemple, nous sommes allés chercher des bases de données retraçant la température de nos régions de 2009 à aujourd'hui.

Nous avons décidé que le joueur avait la possibilité de planter entre 6 et 9 espèces différentes par région du monde. Nous voulions un nombre intermédiaire pour pouvoir gérer leur

caractéristiques facilement. Ces espèces sont rangées en quatres différents types : arbre fruitier, céréale, légumes et fleur. Ces types ont chacun leur lot de nuisibles et de spécificités pour faciliter les choses.

Notre jeu s'actualise de semaine en semaine. Pour faire avancer le jeu, le joueur peut passer à la semaine suivante lorsqu'il a fini ses actions sur la semaine précédente. En passant à la semaine suivante, la météo, l'état des plantes, l'état du terrain, ainsi que les nuisibles qui sont passés par là s'actualisent. Le joueur doit donc gérer ces nouveaux paramètres et faire en sorte que ses plantes gardent un taux de santé assez haut pour survivre et faire assez de fruits.

La météo de la semaine peut être pluvieuse, couverte ou ensoleillée. En fonction, le taux d'humidité et d'ensoleillement changent. Le joueur doit faire attention à ces paramètres pour prendre soin de ses plantes.

Pour chaque nuisible embêtant la parcelle, le joueur dispose de plusieurs outils pour en venir à bout. Chaque semaine, au moins un nuisible vient embêter une plante. Le joueur doit réagir rapidement s'il ne veut pas que la santé de sa plante se détériore. Il existe un grand nombre de nuisibles : chenilles, pucerons, oiseaux, maladies, champignons et lapins. Il existe également un grand nombre d'outils et de solutions pour pallier cela : sécateur, cd, traitement, coccinelle, fermier en colère, serre et paillage.

Structure du code

Nous allons maintenant rentrer dans une description plus fine de notre code et de ses fonctionnalités.

Le jeu se présente sous la forme d'une application console. Ce dernier est programmé de manière à permettre une taille de fenêtre flexible. L'affichage s'ajuste donc dynamiquement selon la taille de la fenêtre en début de partie. La fenêtre comporte une taille d'affichage minimum, si cette dernière est trop réduite, l'écran indique à l'utilisateur de la réajuster. Lors du réajustement de la taille de l'affichage, l'utilisateur reçoit un retour lui disant que l'interface se réajuste, de façon à ne pas entraîner de saccade dans le réajustement des éléments, et ainsi de ne pas brouiller le joueur.
La page d'accueil comporte un titre principal et une image formée en caractères ASCII.
À côté de ces deux éléments d'interfaces, un menu interactif permet de choisir entre créer une partie et continuer la précédente.

L'interface du jeu comporte 3 parties principales. Le volet supérieur regroupant les informations générales sur l'état du jeu et renseignant sur l'avancement actuel du jeu. On y trouve la date, la météo, le lieu et le mode de jeu dans lequel le joueur se trouve. Le volet intermédiaire permettant une visualisation des récoltes et permettant d'interagir avec. Le volet inférieur comprenant les menus de jeu et des commandes d'actions (sélectionner un outil, prendre une décision, etc...). Un volet latéral permettant de présenter les différentes caractéristiques selon la plante sélectionnée dans le champ.

Possibilités du joueur

Le joueur est l'acteur central de notre jeu, c'est à partir de ses actions qu'il avance dans celui-ci. En effet, à chaque début de partie, une instanciation de la classe Joueur est réalisée. Cette classe a été pensée de manière à regrouper toutes les informations nécessaires à la sauvegarde d'une session de jeu. Il faudrait uniquement sauvegarder les informations rassemblées dans chacun des attributs de cette classe et nous aurions un réel gestionnaire de sauvegarde grâce à cette dernière.

Les données du joueur interagissent et sont donc manipulées avec le reste du jeu (orchestré par la classe SessionJeu).

La navigation du jeu se fait à l'aide des touches du clavier. Nous la retrouvons dans la classe SessionJeu qui centralise la logique d'une session de jeu et assure la coordination entre les différents modules du projet (joueur, météo, interface, inventaire, potager, etc.). Elle est conçue comme un contrôleur principal qui initialise les zones d'affichage (ZoneEcranJeu, InterfaceAccueil). Elle construit dynamiquement les menus à travers la méthode ConstruireMenus() et pilote les transitions entre les écrans. L'interaction utilisateur repose sur la capture du clavier via Console.ReadKey(true) stockée dans un objet ConsoleKeyInfo. Cette approche permet une navigation fluide entièrement textuelle, sans interface graphique, où chaque touche déclenche une action. Par exemple, les flèches directionnelles manipulent la position du curseur dans le menu actif (ZoneActive), Enter valide une sélection, Escape ou Backspace revient en arrière, et des touches spécifiques (I, M, J, P, etc.) basculent entre les modules fonctionnels du jeu (Inventaire, Magasin, Journal, Potager). La méthode Naviguer() agit comme une boucle de traitement des entrées utilisateur et oriente l'interface en conséquence. SessionJeu gère également les actions métier (achat, vente, plantation, utilisation d'outil), l'actualisation de l'affichage (dialogues, météo, argent, état du potager), ainsi que l'évolution temporelle du jeu via PasserSemaineSuivante(), qui applique les effets de la météo et la croissance des plantes. L'architecture repose donc sur une interaction temps réel clavier/console, couplée à une structuration en zones d'interface et à une logique métier isolée et réactive.

Le joueur dispose donc d'une parcelle qu'il peut cultiver comme bon lui semble en fonction des plantes à sa disposition que l'on retrouve dans Plante.cs. Nous avons déclaré 21 plantes différentes en faisant des classes filles de Plante. Le joueur doit faire attention au type de terrain pour que la plante soit en meilleure santé possible. Nous retrouvons les différents types de terrains, leur humidité et leur exposition dans Terrain.

Le joueur a la possibilité d'acheter et de vendre des plantes dans Magasin avec l'argent dont il dispose. Cela est codé dans SessionJeu. Le joueur peut également acheter des outils dans le magasin. On retrouve tous les outils dans Outil.cs , il y a le paillage, la serre, le fermier en colère, la coccinelle, le traitement, le fumier, les cd, le sécateur, le panier et l'irrigation d'urgence. Chaque outil a ses spécificités et ses actions possibles. Certain comme l'irrigation d'urgence peut se déclencher qu'en mode urgence lors d'une sécheresse.

Une fois les items achetés, le joueur les retrouve dans son inventaire et peut les utiliser selon la situation. Attention, une mauvaise utilisation des outils peut entraîner un impact négatif sur la santé de la plante.

Le joueur peut aller voir dans le menu journal les actualités sur l'état de ses plantes et de son terrain.
Journal, Inventaire et Magasin sont des zoneMenu. La classe ZoneMenu regroupe des zones dédiées à l'affichage d'un menu, permettant de naviguer dans son arborescence.

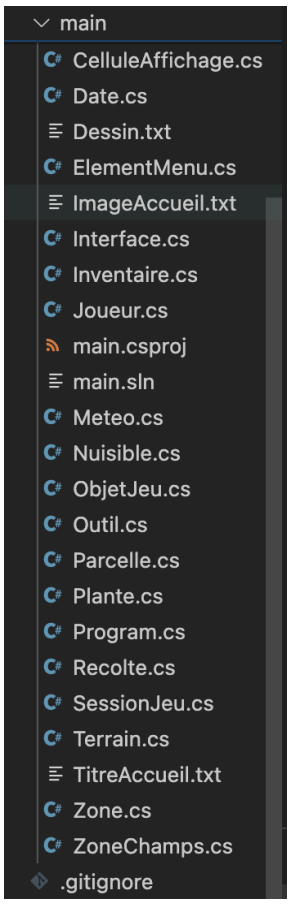
Déroulement de la partie

Une partie débute lorsque le joueur sélectionne une région parmi celles proposées. Cela déclenche l'instanciation d'un objet Joueur initialisé avec un potager vide, un inventaire de départ, un sol adapté à la région choisie, et un capital de 2000 crédits.

En mode normal, la simulation progresse de manière hebdomadaire, à partir de la semaine 1 de l'année 2009, jusqu'à ce que le joueur perde ou atteigne la limite des 10 années de jeu. Le joueur perd en n'ayant plus d'argent et en ayant toutes ses plantes mortes. Chaque semaine, la météo est générée dynamiquement via le GestionnaireMeteo, influençant l'état des plantes sur les parcelles selon la température, le temps (soleil, pluie, etc.), et les spécificités de chaque espèce (Plante). Le joueur peut interagir à tout moment avec son environnement via l'interface console, notamment en plantant (PlanterSemis), utilisant des outils (UtiliserOutil), vendant des récoltes, ou en passant à la semaine suivante. La perte intervient lorsque toutes les plantes sont mortes et que le joueur n'a plus les ressources nécessaires pour acheter de nouveaux plants. La victoire est atteinte au terme de 10 années complètes de jardinage, si le joueur parvient à maintenir son activité. Le jeu peut être quitté à tout moment en appuyant sur la touche "X", détectée via Console.ReadKey() dans la boucle principale de navigation (Naviguer()).

Le mode urgence se déclenche lorsqu'une catastrophe se déclenche comme du gel, une grande sécheresse ou un envahissement de nuisible. Le joueur dispose d'un certain nombre de solutions dans outil pour faire face à cela. Ces catastrophes détériorent très vite la santé de la plante, il faut donc réagir vite. Ces catastrophes sont tirées aléatoirement une fois par mois selon la saison.

Modélisation objet réalisée



Tout notre code est construit en programmation orientée objet avec le langage C#. Nous avons créé plusieurs fichiers afin de fluidifier et rendre plus accessible notre code. On retrouve cette arborescence là (figure à gauche).

Nous avons fait certains choix concernant notre code. Notamment, nous avons choisis de construire tous nos tableaux à double entrée ou ordres de paramètres en mettant toujours en premier les abscisses puis les ordonnées. Ce choix s'est imposé pour que la logique des méthodes déjà implémentées par la classe Console.Console.GetCursorPosition et Console.SetCursorPosition ,qui prennent en premier indice la largeur/colonne, et en second la hauteur/ligne, soit cohérente pour le reste du code.

Dans notre arborescence, nous avons essayé au mieux de séparer la gestion de l'affichage et les données, pour avoir une gestion claire des interactions entre ces dernères.

Affichage

Dans l'affichage, nous avons une première classe qui s'appelle Zone. Nous avons fait en sorte que toutes les sous classes de zones s'assemblent dans l'affichage de la console de façon responsive.

Le fichier Zone.cs définit l'architecture de l'interface utilisateur console du jeu à travers une hiérarchie de classes orientée objet centrée sur la notion de "**zone**". La classe abstraite Zone constitue la base commune de toutes les zones d'affichage.
Elle encapsule les dimensions, la position à l'écran et des méthodes utilitaires permettant d'écrire du texte (EcrireTexte), de gérer les retours à la ligne et d'effacer le contenu.

À partir de cette classe, différentes sous-classes spécialisent le comportement :

- ZoneTexte affiche un simple contenu textuel avec des couleurs personnalisables
- ZoneDessin permet d'afficher dans la console des illustrations ASCII en lisant des fichiers texte

- ZoneDialogue, qui en hérite, sert à afficher des messages dynamiques liés à la progression du jeu
- ZoneInteractive, également abstraite, introduit la notion de curseur de navigation qui peut être déplacé grâce à la méthode DéplacerCurseur()
- ZoneMenu est dérivée de ZoneInteractive, une classe dédiée à la création de menus arborescents contrôlés au clavier.

Cette dernière permet de naviguer dans les menus à l'aide des flèches directionnelles (Curseur), de sélectionner des éléments (ValiderSelection) ou de revenir en arrière (RetournerEnArriere). Elle contient un élément ElementMenu.

L'ensemble de ce système repose sur une utilisation avancée de la console C#, manipulant précisément le curseur, les couleurs et les lignes affichées pour simuler une interface utilisateur interactive et segmentée dans un environnement textuel.
Zone permet donc de délimiter une zone dans l'affichage. Chaque zone peut être affichée grâce à la méthode Afficher qui se décline en d'autres zones comme vue précédemment.

Nous avons un second fichier ZoneChamps.cs qui gère la classe ZoneChamps. Celle-ci est une zone-grille qui permet de gérer des cellules avec des méthodes spécifiques au champ.

Données

Concernant les données, nous avons créé une classe Joueur. La classe Joueur modélise l'ensemble des données liées à un joueur durant une partie. Elle centralise les informations essentielles à la simulation :

- le montant d'argent disponible (Argent)
- la date en jeu (DateActuelle)
- le lieu de la partie (Lieu)
- un inventaire structuré via la classe Repertoire
- un potager représenté sous forme de tableau à deux dimensions (Parcelle[,] Potager) de taille déterminée par la taille de la fenêtre en début de session.

Lors de l'instanciation d'un objet Joueur, le constructeur initialise par défaut la date et le capital du joueur, génère un potager vide en appelant la méthode privée CreerPotager(), et remplit l'inventaire avec quelques éléments de départ (semis et récoltes). La structure du potager repose sur une double boucle qui initialise chaque case avec une instance de Parcelle, configurée selon le lieu de jeu sélectionné. De manière symbolique, un Citronnier est automatiquement planté dans la parcelle située en (2,1). Cette classe joue un rôle fondamental dans la logique du jeu en tant que conteneur des ressources et des interactions du joueur avec le monde (achats, plantations, ventes, gestion des semis, etc.), et est systématiquement manipulée par les autres composants du programme.

Dans les données nous retrouvons également des données sur le jeu comme avec les classes Meteo, Outils, Plantes, Nuisibles, Recolte. La classe ObjetJeu est la classe mère.

Les classes dérivées de ObjetJeu forment la base des entités manipulables dans le jeu, qu'il s'agisse de plantes, d'outils ou de récoltes. ObjetJeu fournit des propriétés communes telles que le nom, un emoji de représentation visuelle, ainsi que les prix d'achat et de vente.

La classe abstraite Plante, qui en hérite, modélise les végétaux cultivables avec un ensemble riche de propriétés biologiques et mécaniques : types de terrain préférés, besoins en eau et en soleil, résistance au froid ou à la sécheresse, liste de nuisibles possibles, état de santé, stade de croissance, ainsi que rendement et saison de semis/récolte. Chaque sous-classe de Plante (ex. Pommier, Tomate, Riz, etc.) définit ses paramètres spécifiques via le constructeur et réimplémente la méthode Dupliquer() pour permettre une copie conforme du végétal. Les plantes sont la base du jeu. Chaque plante peut être achetée ou vendue au magasin et possède un prix d'achat et de vente. Lorsqu'une plante est plantée sur un terrain, elle possède un temps de croissance qui augmente chaque semaine jusqu'à ce qu'elle soit mature (si croissance==100 : devient mature). Tous ces facteurs influencent la santé d'une plante. La santé Sante d'une plante affecte son rendement et est déterminée sur 100: 80 correspond à un rendement maximum, 70 un rendement important, 60 un rendement modéré et 50 un rendement faible. Chaque plante possède son propre rendement. Concernant l'eau, Sante diminue de -20 si le besoin en eau n'est pas suffisant, de -20 s'il est trop important. Concernant le soleil, Sante diminue de -15 si la quantité de soleil n'est pas suffisante et de -5 si la quantité de soleil est à 100% (trop d'exposition tue l'exposition). Concernant la température, Sante diminue de -15 si la différence de température avec la température de préférence est > à 15°C.

À l'opposé, la classe Outil, également abstraite, représente des objets utilisables directement sur une parcelle (ex. Arrosoir, Fumier, Secateur), et impose l'implémentation de la méthode Actionner(Parcelle) qui applique un effet spécifique au sol, à la plante ou aux nuisibles. Certains outils sont conçus pour protéger (Serre, CD, Coccinelle), d'autres pour soigner ou optimiser la production (Traitement, Fumier, Panier).

Également, la classe Meteo renferme le temps que le joueur va rencontrer tout du long de sa partie. Les températures dans Temperatures ont été récupérées de données réelles et 3 types de ciel ont été modélisés : pluie, soleil et nuageux. Pluie donne de +10 à +25% d'humidité à l'ensemble des terrains. Soleil donne de +10 à +25% d'exposition à l'ensemble des terrains. Nuageux ne donne pas d'évolution de la pluie et du taux de soleil.

Aussi, la classe Nuisible, également abstraite, permet de référencer les différents parasites, animaux ou maladies que nous pouvons trouver sur les parcelles (ex. Chenille, Champignons, Lapins). Chaque semaine, un nuisible vient s'installer sur une des plantes On choisit pour une plante dans le champs un des nuisibles qu'il possède et on l'ajoute à la liste. Chaque catégorie de plante possède ses propres types de nuisibles. Chaque nuisible peut être retiré selon son outil.

Enfin, des objets comme les récoltes (non détaillées ici mais implicites via RecoltePommier, RecolteTomate, etc.) complètent cette architecture, servant d'unités économiques pour le système de vente/échange. Cette structure en héritage permet une forte extensibilité du gameplay, en assurant que chaque nouvel objet du jeu suive une logique uniforme tout en ayant des comportements spécialisés.

La classe Terrain gère l'exposition aux intempéries et aux nuisibles tandis que la plante fixe les limites. En effet, il existe 3 types de terrains selon la localisation où vous vous trouvez. Le terrain à Carcassonne est argileux. Un terrain possède TauxHumidite qui gère l'humidité du terrain et TauxExposition qui gère l'exposition au soleil. Le terrain peut être fertile (nombre qui peut ajouter jusqu'à 10 à la santé, diminue de 2 à chaque semaine) Le terrain possède un drainage qui est fixe et déterminé par le type de terrain.

La classe Temperature regroupe les données sur 15 ans des températures moyennes hebdomadaires pour un lieu-dit.
La classe Date permet d'actualiser la date semaine par semaine.

Lien

L'affichage et les données sont reliés par ElementMenu et SessionJeu .
SessionJeu est la classe principale qui lance la partie. Elle interagit donc avec l'affichage et les données. Nous l'avons déjà détaillée dans "Possibilité du joueur"

Le fichier ElementMenu.cs définit l'ensemble des structures de menus interactifs utilisés dans l'interface console du jeu.

C'est une des fonctionnalités principales du programme. En effet, cette classe permet de naviguer à travers une liste d'items, tous liés entre eux, de descendre et de remonter dans une arborescence ainsi que d'appliquer différentes fonctionnalités.

L'affichage de la liste d'items s'ajuste automatiquement en fonction de la hauteur de la zone d'affichage définie. Selon la taille, un nombre de pages est défini et lorsque l'utilisateur descend à la fin d'une page, la page suivante s'affiche avec le nombre d'items restants.

La classe principale ElementMenu représente une entrée de menu textuelle pouvant contenir des sous-éléments (Items), un lien vers un menu parent (Parent) pour remonter à celui-ci, et une référence à la zone graphique (ZoneMenu) dans laquelle elle s'affiche. Chaque élément peut être activé via la méthode Actionner(), qui permet soit de naviguer dans l'arborescence du menu, soit de déclencher une action concrète en jeu. Plusieurs sous-classes spécialisées héritent de ElementMenu pour couvrir différents contextes d'interaction :

- ElementMenuNouvellePartie lance une nouvelle session de jeu dans une ville sélectionnée
- ElementMenuMagasinAchatOutil
- ElementMenuMagasinAchatSemis
- ElementMenuMagasinVenteRecolte permettent d'acheter ou de vendre des objets,
- ElementMenuInventaireSemis
- ElementMenuInventaireOutil permettent d'utiliser les outils possédés par le joueur.

Ces éléments sont tous couplés à une instance de SessionJeu, ce qui garantit une interaction fluide entre les menus affichés à l'écran et la logique métier du jeu. Ce système de menus hiérarchiques est conçu pour fonctionner entièrement au clavier, en lien avec le module ZoneMenu, et repose sur un modèle orienté objet favorisant la modularité, la réutilisabilité et l'extension des comportements par

héritage. Dans ZoneMenu, ElementMenu peut être une Racine ou un NoeudActif (pour savoir dans quel noeud on se trouve actuellement).

Tests réalisés

Afin de vérifier le bon fonctionnement de l'application, nous avons réalisé plusieurs tests au fur et à mesure de notre création de classes dans Programme.cs . Cependant, ces tests se sont principalement passés sur le débogueur, qui a été une vraie plus value en termes d'outils disponibles. Ce projet nous a permis de voir plus amplement de quoi il était possible de faire avec (utilisation d'espions, très utiles !), mais la configuration pour pouvoir avoir le terminal intégré et les Méthodes de la Classe Console était peu évidente (même si nous avons réussi !)

Une fois la base du mode normal solide, nous avons testé différentes situations comme le mode urgence, celui de la situation perdu avec plus de plantes et plus d'argent et celle avec la dernière semaine de l'année pour voir si on passait bien d'année en année après la semaine 52. Ces tests sont intégrés au niveau des touches, avec la touche u pour déclencher le mode urgence et x pour arrêter le code directement.

Notre session démarre par un type de session où le joueur possède tous les outils ainsi qu'un nombre de plantes et une bourse plus importante a été ajouté au jeu de manière à pouvoir tester les différents outils et fonctionnalités sur les différentes plantes.

Limites

Nous avons pu réaliser un projet complet intégrant une gestion modulaire et flexible de l'affichage grâce aux classes zones et ElementMenu, qui façonnent l'interface. La navigation effectuée par les touches directionnelles, la touche entrée, echap et retour, ainsi que les raccourcis sont tant d'éléments qui permettent une navigation intuitive. Cependant, cet aspect nous a pris un temps important, car nous ne nous limitons pas à de simples Console.Write ou Console.WriteLine().

Grâce à la POO, nous pourrons réutiliser les composants d'affichage afin de créer à venir des zones de texte, et des éléments d'interface plus élaborés.

Nous avons pu implémenter des fonctionnalités telles que le magasin, la gestion de l'inventaire, la météo et la température, la gestion des nuisibles et du taux d'humidité, le tout interagissant avec chaque parcelle du champ et ainsi sur la santé de chacune des plantes.

Cependant, par manque de temps, nous n'avons pas pu implémenter une des fonctionnalités principales qui est le mode urgence. Des bouts de codes liés au commencement de cette implémentation sont présents, mais au vu du travail fourni et du temps pris, nous n'avons pas poursuivi la création de cette fonctionnalité. Nous avons toutes les compétences afin de réaliser celui-ci, avec un petit exemple d'an

Nous avons initialement aussi eu l'idée d'intégrer une image ASCII à partir d'un texte .txt, et de pouvoir importer les données d'un joueur et les exporter sur un csv, la classe joueur renfermant tous les éléments nécessaires à l'exécution d'une sauvegarde de la parte. Nous n'avons cependant pas eu le temps de le faire.

L'architecture abstract des plantes ne nous sert pas beaucoup car les plantes ont les mêmes fonctionnalités, avec plus de temps, nous aurions pu décliner en types de plantes les différents types

en arbre, fleur, et légume, qui auraient chacun eus leurs propres caractéristiques et actions. Même chose pour le terrain.

Une de nos incohérences dans le design du jeu a aussi été d'appliquer les options uniquement à la plante, alors que des outils tels que le paillage seraient plus destinés au sol.

Bilan

Ce projet a constitué une expérience à la fois riche sur le plan technique et formatrice en matière de gestion de projet collaboratif. En développant une application console complète en C#, nous avons mis en œuvre une architecture logicielle intégralement orientée objet, structurée autour d'une arborescence claire de classes (Plante, Outil, Joueur, Meteo, etc.), avec une séparation rigoureuse entre données de jeu et interface d'affichage.

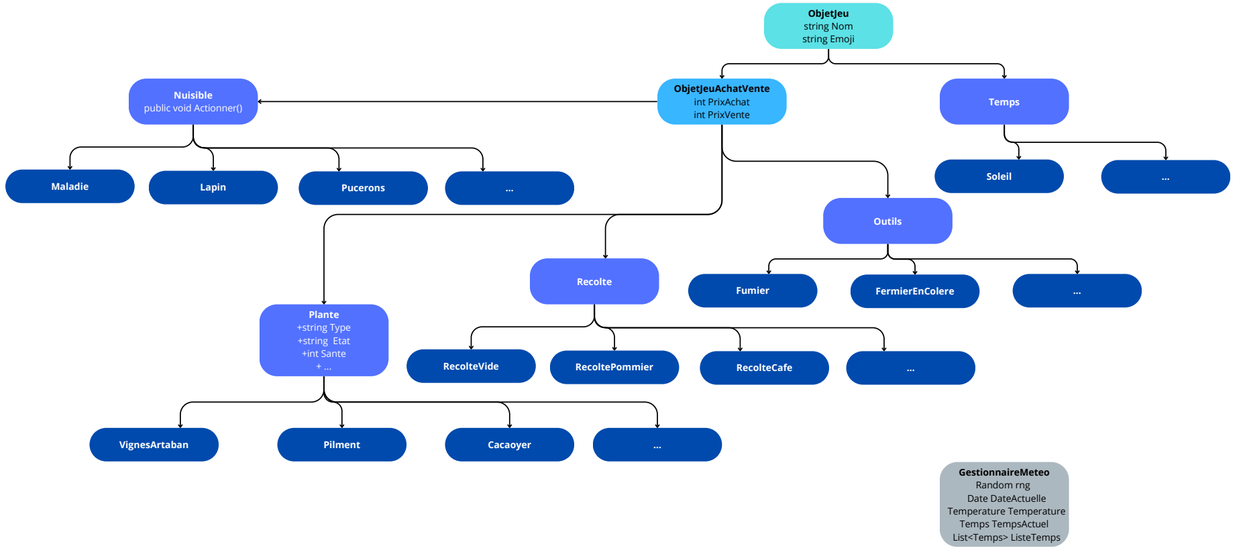
L'utilisation de GitHub nous a permis d'assurer une collaboration fluide, rigoureuse et bien documentée, notamment grâce à un usage méthodique des branches, commits et revues de versions précédentes du code. Grâce à ce projet, nous avons pu réellement développer nos compétences en gestion de versionnage et de branches, ainsi que pour la gestion de conflits.

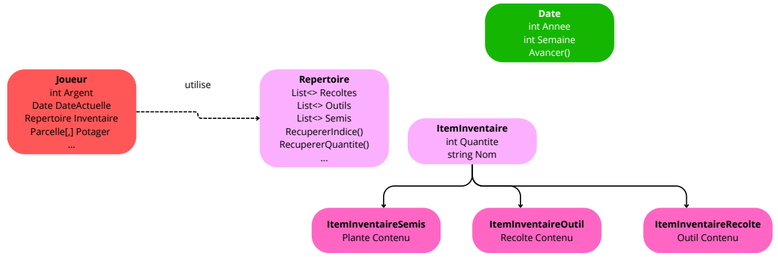
Sur le plan fonctionnel, nous avons réussi à implémenter un moteur de jeu complet, cohérent et immersif, dans lequel les mécaniques agricoles (croissance des plantes, gestion des outils, météo, nuisibles…) sont modulées par des règles inspirées du réel et adaptées aux contraintes de l'environnement console. Le système de menus dynamiques, l'affichage ASCII adaptatif, et l'évolution hebdomadaire du potager ont été des défis techniques que nous avons relevés avec succès. La diversité des classes dérivées d'ObjetJeu nous a permis de créer un univers riche, personnalisable par région, avec plus de vingt plantes uniques et une variété d'outils et d'événements.

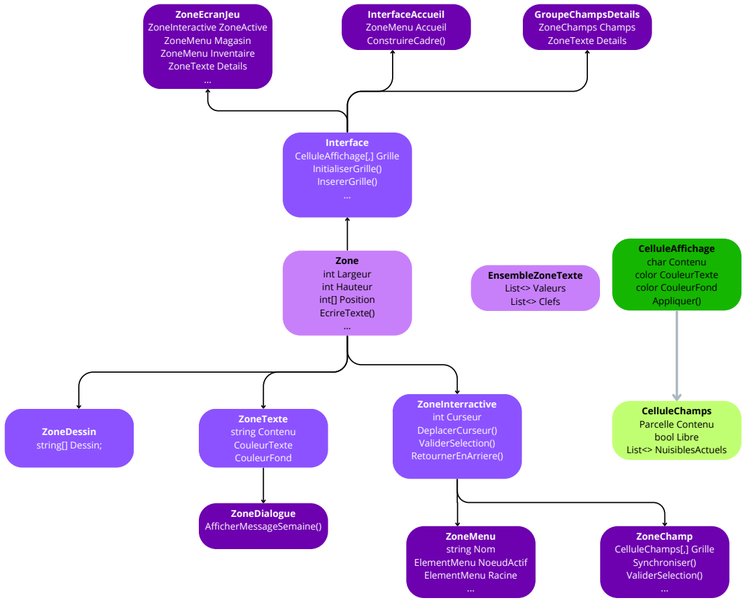
Malgré l'ampleur des fonctionnalités mises en œuvre, certaines idées supplémentaires comme l'introduction d'un mode multijoueur ont été écartées par souci de temps. Nous n'avons pas eu le temps non plus de coder en profondeur la région du Mexique et celle du Japon pour parfaire plutôt celle française. Néanmoins, le projet tel qu'il est aujourd'hui est complet, stable, et extensible, avec une logique modulaire facilitant l'ajout de nouvelles fonctionnalités.
Nous ressortons de ce projet fiers du travail accompli, avec une compréhension renforcée de la POO et des structures de données complexes. Ce jeu est pour nous un aboutissement technique, mais aussi un témoignage de notre capacité à transformer une idée en un produit fonctionnel, ludique et cohérent.

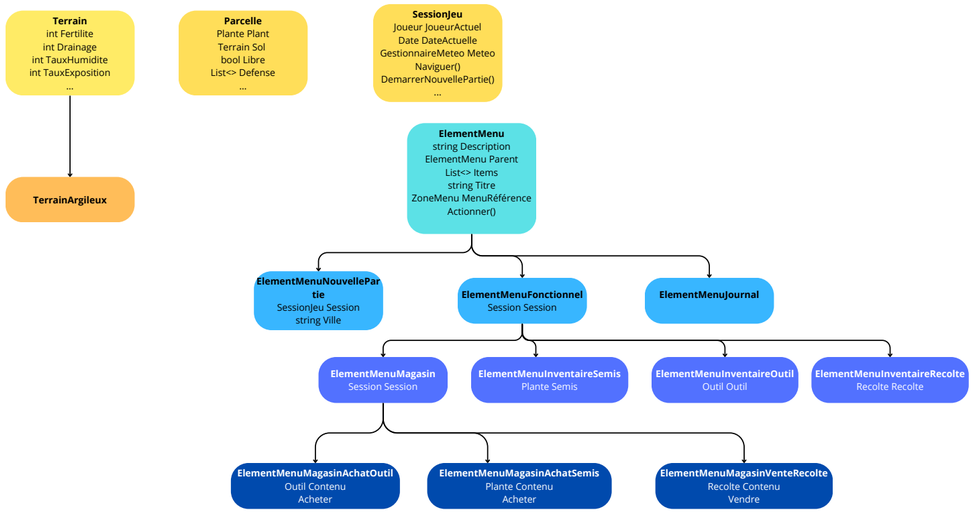
Annexe : architecture du projet

conseil : zoomez pour mieux voir en détail le texte car la qualité se dégrade en vue globale


ObjetJeu
string Nom
string Emoji
Nuisible
public void Actionner()
ObjetJeuAchatVente
int PrixAchat
int PrixVente
Temps
Maladie
Lapin
Pucerons
...
Soleil
...
Outils
Recolte
Fumier
FermierEnColere
...
Plante
+string Type
+string Etat
+int Sante
+ ...
RecolteVide
RecoltePommier
RecolteCafe
...
VignesArtaban
Pilment
Cacaoyer
...
GestionnaireMeteo
Random rng
Date DateActuelle
Temperature Temperature
Temps TempsActuel
List<Temps> ListeTemps


Date
int Annee
int Semaine
Avancer()
Joueur
int Argent
Date DateActuelle
Repertoire Inventaire
Parcelle[,] Potager
...
utilise
Repertoire
List<> Recoltes
List<> Outils
List<> Semis
RecupererIndice()
RecupererQuantite()
...
ItemInventaire
int Quantite
string Nom
ItemInventaireSemis
Plante Contenu
ItemInventaireOutil
Recolte Contenu
ItemInventaireRecolte
Outil Contenu


ZoneEcranJeu
ZoneInteractive ZoneActive
ZoneMenu Magasin
ZoneMenu Inventaire
ZoneTexte Details
...
InterfaceAccueil
ZoneMenu Accueil
ConstruireCadre()
GroupeChampsDetails
ZoneChamps Champs
ZoneTexte Details
Interface
CelluleAffichage[,] Grille
InitialiserGrille()
InsererGrille()
...
Zone
int Largeur
int Hauteur
int[] Position
EcrireTexte()
...
EnsembleZoneTexte
List<> Valeurs
List<> Clefs
CelluleAffichage
char Contenu
color CouleurTexte
color CouleurFond
Appliquer()
ZoneDessin
string[] Dessin;
ZoneTexte
string Contenu
CouleurTexte
CouleurFond
ZoneInterractive
int Curseur
DeplacerCurseur()
ValiderSelection()
RetournerEnArriere()
CelluleChamps
Parcelle Contenu
bool Libre
List<> NuisiblesActuels
ZoneDialogue
AfficherMessageSemaine()
ZoneMenu
string Nom
ElementMenu NoeudActif
ElementMenu Racine
...
ZoneChamp
CelluleChamps[,] Grille
Synchroniser()
ValiderSelection()
...


Terrain
int Fertilite
int Drainage
int TauxHumidite
int TauxExposition
...
TerrainArgileux
Parcelle
Plante Plant
Terrain Sol
bool Libre
List<> Defense
...
SessionJeu
Joueur JoueurActuel
Date DateActuelle
GestionnaireMeteo Meteo
Naviguer()
DemarrerNouvellePartie()
...
ElementMenu
string Description
ElementMenu Parent
List<> Items
string Titre
ZoneMenu MenuRéférence
Actionner()
ElementMenuNouvellePartie
SessionJeu Session
string Ville
ElementMenuFonctionnel
Session Session
ElementMenuJournal
ElementMenuMagasin
Session Session
ElementMenuInventaireSemis
Plante Semis
ElementMenuInventaireOutil
Outil Outil
ElementMenuInventaireRecolte
Recolte Recolte
ElementMenuMagasinAchatOutil
Outil Contenu
Acheter
ElementMenuMagasinAchatSemis
Plante Contenu
Acheter
ElementMenuMagasinVenteRecolte
Recolte Contenu
Vendre