

# Rapport de justification technique du Projet ENSEmenC

PEREZ Ambre & VEYRENT Emma

Elèves en Première année en cycle ingénieur  
Ecole Nationale Supérieure de Cognitique  
(ENSC-Bordeaux INP)

*à rendre le 23 mai 2025*



I. Introduction	3
II. Mode d'emploi	3
A) Lancement du jeu	3
B) Commandes de base	3
C) Types de plantes disponibles :	4
D) Temporalité	4
E) Conditions de croissance	4
F) Système économique	4
III. Classes de gestion des terrains	5
A) Terrain	5
B) TypeTerrain	6
C) ParcelleTerrain	6
III. Classes de gestion des plantes	7
A) Plante	7
B) TypePlante	7
IV. Classes de gestion des agents positifs et négatifs	8
A) Animal	8
B) Abeille	9
C) Taupe	9
D) Escargot	10
V. Fonctionnalités supplémentaires	10
A) Economie	10
B) Meteo	11
VI. Classes de gestion du jeu	11
A) Affichage	11
B) GestionJeu	12
C) Program.cs	13
VII. Bilan du projet et difficultés rencontrées	14

# I. Introduction

Ce rapport tient à présenter les justifications techniques portant sur le projet ENSemenc, projet concernant le module Programmation Avancée de l'année scolaire 2024-2025. Ce projet consiste en la réalisation d'un jeu numérique portant sur le thème du potager et s'affichant dans le terminal. Il a été développé en C#, dans l'éditeur VS Code et Github fut utilisé afin de permettre aux deux membres de l'équipe de travailler en collaboration. Ce jeu permet au joueur de gérer virtuellement un potager en plantant, récoltant et en s'occupant de plantes en tous genres. Les conditions de croissance des différents végétaux furent intégrées au projet ainsi que des animaux pouvant améliorer ou menacer les cultures. La météo et les conditions géologiques des différents terrains furent aussi implémentées.

Au niveau de la temporalité du jeu, un tour correspond à une semaine, et toutes les saisons défilent. Le but du joueur est d'incarner un jardinier et de maximiser le rendement de son jardin.

Ce rapport détaille ainsi l'organisation du code du projet avec les différentes classes implémentées qui assurent son fonctionnement. Mais avant de comprendre les justifications techniques du jeu, voici un petit mode d'emploi facilitant ainsi la prise en main du jeu en tant que joueur.







# II. Mode d'emploi




Vous incarnez donc un jardinier et devez gérer virtuellement votre potager en plantant, récoltant et vendant des produits pour maximiser vos profits et votre expérience !

## A) Lancement du jeu

- Lancez l'application
- Entrez votre nom de jardinier
- Vous commencez avec : 💰 10€ d'argent de départ, 2 graines de rose 🌹, 3 graines de tomate 🍅 et 2 graines de carotte 🥕.






## B) Commandes de base

Touche	Action	Description
P	 Planter	Planter une plante
A	 Arroser	Arroser une parcelle
S	 Soigner	Soigner la plante
R	 Récolter	Récolter les produits
E	 Enlever	Supprimer une plante
T	 Temps +1	Avancer d'une semaine

M	 Magasin	Acheter des graines
V	 Vendre	Vendre vos productions
Q	 Quitter	Quitter le jeu

### C) Types de plantes disponibles :

Lorsque vous plantez, vous devez posséder la graine adéquate. Une fois plantée, vous ne posséderez plus cette graine dans votre inventaire.










-  Rose (Ornementale) - Printemps - Terre fertile
-  Tomate (Consommable) - Été - Terre meuble
-  Carotte (Consommable) - Automne - Sable
-  Tournesol (Décoration) - Été - Terre fertile
-  Basilic (Aromatique) - Été - Terre meuble

### D) Temporalité

Dans le jeu une année correspond à 4 saisons (du printemps jusqu'à l'hiver) et comporte donc 4\*13 semaines. Les 2 premières semaines suivant une plantation correspondent à la phase de germination et la production commence à partir de la semaine 3. Il faudra donc être patient avant de pouvoir commencer à récolter !

### E) Conditions de croissance

Attention la couleur des plantes indique leur état de santé. Pour qu'elles produisent, il faut en prendre soin en les soignant les arrosant mais aussi en les protégeant des agents destructeurs que sont les escargots et les taupes, ou encore de mauvaises intempéries.

-  Abeille : Plante des roses gratuitement en se déplaçant
-  Coccinelle : Soigne toutes les plantes malades dans un rayon de 2 cases
-  Escargot : Réduit la santé des plantes adjacentes (-10%)
-  Taupe : Détruit complètement une plante adjacente
-  Pluie : Arrose automatiquement toutes les plantes
-  Sécheresse : Les plantes perdent plus de santé
-  Grêle : Dégâts aléatoires sur 30% des plantes
-  Gel : Les plantes sensibles au froid souffrent
-  Canicule : Stress hydrique généralisé

### F) Système économique

Chaque plante possède un prix de vente et un prix d'achat différent :

-  Rose : vente 8€ achat 3€

- 🍅 Tomate : vente 5€ achat 2€
- 🌻 Tournesol : vente 6€ achat 3€
- 🌿 Basilic : vente 4€ achat 2€
- 🥕 Carotte : vente 3€ achat 1 €

🌱 Bon jardinage et amusez-vous bien avec Potager et Cie ! 🌱

### III. Classes de gestion des terrains

#### A) Terrain

Dans ce premier point nous expliciterons la classe abstraite `Terrain` qui constitue la base de notre système de gestion territoriale. De cette classe dérive les propriétés communes à tous les types de terrains qui sont :

- nom, région et surface du terrain
- conditions environnementales (température, humidité, pH, ensoleillement)
- grille à 2 dimensions de parcelles (`ParcelleTerrain[, ]`)
- dimensions (hauteur, largeur)

Le constructeur définissant ces caractéristiques inclut aussi des valeurs par défaut pour le NiveauHumidité, NiveauSoleil, Temperature et pH. Diverses méthodes ont été définies dans cette classe. Nous y retrouvons plusieurs méthodes publiques virtuelles , à savoir :

- `PlanterPlante(Plante plante, int x, int y)`
- `ProgresserSemaine(string saison)`
- `ArroserParcelle(int x, int y)`
- `RecolterParcelle(int x, int y)`
- `SoignerPlante(int x, int y)`
- `EnleverPlante(int x, int y)`
- `ObtenirResume()`

Deux méthodes protégées et une méthode publique :

- `VerifierEspacement(Plante plante, int x, int y)`
- `AjusterConditionsSaisonnières(string saison)`
- `CalculerEspaceDisponible(int x, int y)`

Comme leurs noms l'indiquent, ces méthodes permettent la gestion des plantes, des actions de maintenance, l'implémentation de la simulation temporelle, des calculs et l'obtention d'informations.

## B) Type Terrain

À partir de la classe abstraite `Terrain`, nous avons développé trois types de terrains spécialisés qui héritent de cette classe parent :

**TerrainSable** : Cette classe simule un terrain sablonneux avec ses caractéristiques propres. Le niveau d'humidité par défaut est fixé à 30% car le sable est naturellement plus sec. Le pH est défini à 6.5, proche de la neutralité mais légèrement acide. La particularité de ce terrain réside dans sa surcharge de la méthode `AjusterConditionsSaisonnières()` qui reflète le fait que le sable draine rapidement l'eau, réduisant l'humidité de 10% à chaque ajustement saisonnier avec un minimum de 20%.

**TerrainArgileux** : Ce type de terrain modélise les propriétés de l'argile qui retient davantage l'eau. L'humidité par défaut est ainsi fixée à 70% et le pH à 7.5, légèrement basique. La méthode `AjusterConditionsSaisonnières()` surchargée augmente l'humidité de 10% chaque saison avec un plafond de 90%, simulant la capacité de l'argile à conserver l'eau.

**TerrainMarecageux** : Le terrain marécageux présente des caractéristiques particulières avec une humidité très élevée (90%) et un pH acide (5.5). Sa spécificité majeure réside dans l'implémentation d'un système de propagation des maladies avec 15% de chance d'apparition chaque semaine. Deux méthodes privées gèrent cette fonctionnalité : `PropagerMaladie()` qui choisit aléatoirement une parcelle pour commencer l'infection, et `PropagerMaladieAuxVoisins()` qui utilise la récursivité pour propager l'infection aux parcelles adjacentes avec une probabilité décroissante pour éviter une contamination excessive.

## C) Parcelle Terrain

Cette classe représente une case individuelle du terrain et constitue l'unité de base de notre grille de jeu. Chaque parcelle peut contenir une plante (`PlanteCourante`) et temporairement un animal (`AnimalCourant`). Le constructeur prend en paramètre le terrain parent et initialise la parcelle à vide. Les méthodes importantes de cette classe incluent :

- `EstVide()` : Vérifie si la parcelle ne contient aucune plante
- `PlanterPlante(Plante plante)` : Permet de planter une plante si la parcelle est libre
- `ProgresserSemaine()` : Fait évoluer la plante selon les conditions transmises
- `Arroser()`, `Soigner()`, `Recolter()` : Actions de maintenance sur la plante
- `EnleverPlante()` : Suppression de la plante courante
- `ObtenirVisuel()` : Gestion de l'affichage avec priorité aux animaux
- `ObtenirInfoPlante()` : Retourne les détails de la plante présente
- `Infecter()` : Méthode permettant de rendre malade la plante

### III. Classes de gestion des plantes

#### A) Plante

La classe abstraite `Plante` modélise de manière réaliste les besoins et caractéristiques des végétaux selon les exigences du cahier des charges. Cette classe encapsule toutes les propriétés biologiques nécessaires à la simulation :

##### Caractéristiques biologiques statiques :

- `Nom` et `Nature` (consommable, ornementale, etc.)
- `SaisonOptimale` de croissance et `TypePref` de terrain
- `EspacementEntre2` et `PlaceNecessaire` pour le développement optimal
- `VitessePousse`, `EauNecessaire`, `EnsoleillementNecessaire`
- `TemperaturePref`, `NiveauHumiditePref`
- `MaladiePotentielle`, `EsperanceVie`, `NombreProduction`
- `Visuel` (emoji représentant la plante)

##### États dynamiques évolutifs :

- `TailleActuelle` et `SanteActuelle` (pourcentage entre 0-100%)
- `EstMalade` (statut booléen de maladie)
- `Age` en semaines et `ProductionActuelle`
- `DatePlantation` pour le suivi temporel

Le système de développement repose sur un calcul sophistiqué via la méthode `CalculerCoefficientDeveloppement()` qui évalue 6 critères pondérés :

- Compatibilité du type de terrain (20%)
- Adéquation de l'humidité (20%)
- Niveau d'ensoleillement optimal (20%)
- État de santé de la plante (10%)
- Température appropriée (15%)
- Espace disponible suffisant (15%)

La méthode `Progresser()` utilise ainsi ce coefficient afin de déterminer l'évolution hebdomadaire de la plante. Si le coefficient est inférieur à 50%, la plante perd de la santé, sinon elle grandit proportionnellement. Le système inclut également une probabilité de maladie de 5% par semaine et gère le vieillissement naturel.

#### B) TypePlante

À partir de la classe abstraite `Plante`, nous avons développé cinq types de plantes spécialisées qui héritent de cette classe parent et implémentent leurs caractéristiques spécifiques :



**Rose** : Cette classe modélise une plante ornementale de saison printanière. Elle préfère la terre fertile avec un niveau d'humidité de 65% et une température optimale de 20°C. Son espérance de vie s'étend sur 24 semaines avec une vitesse de pousse de 3 cm par période. La rose nécessite un ensoleillement de 70% et peut souffrir d'oïdium. Son espacement requis est de 2 parcelles entre chaque plant pour un développement optimal.

**Tomate** : Plante consommable estivale adaptée à la terre meuble. Avec une préférence pour 60% d'humidité et 25°C, elle produit jusqu'à 5 fruits par pied sur une durée de vie de 12 semaines. Sa croissance de 2 cm par période nécessite 80% d'ensoleillement et elle peut être affectée par le mildiou. L'espacement minimal entre plants est d'une parcelle.

**Carotte** : Cette plante consommable automnale prospère dans le sable avec seulement 50% d'humidité et une température fraîche de 15°C. Sa croissance lente de 1,5 cm par période s'étend sur 16 semaines pour produire une racine. Elle nécessite 60% d'ensoleillement et peut subir l'attaque de la mouche de la carotte. Son espacement compact de 0,5 parcelle permet une culture dense.

**Tournesol** : Plante ornementale estivale majestueuse nécessitant 2 parcelles d'espace et préférant la terre fertile. Avec une croissance rapide de 4 cm par période, elle atteint sa maturité en 20 semaines sous 90% d'ensoleillement optimal. Sa température préférée de 28°C et son besoin d'humidité de 55% reflètent ses origines méditerranéennes. Elle peut être affectée par le sclérotinia.

**Basilic** : Herbe aromatique estivale adaptée à la terre meuble, nécessitant 65% d'humidité et 24°C pour sa croissance optimale de 2,5 cm par période. Sur 15 semaines de vie, elle peut produire 3 récoltes successives sous 75% d'ensoleillement. L'espacement d'une parcelle entre plantes permet une culture organisée, mais elle reste vulnérable à la fusariose, sa maladie associée.

## IV. Classes de gestion des agents positifs et négatifs

### A) Animal

La classe abstraite **Animal** constitue la base commune de notre système d'agents mobiles dans le potager. Cette classe définit les propriétés essentielles partagées par tous les animaux : **Nom**, **Type**, **Comportement** et **Visuel** pour l'identification et l'affichage, ainsi que les coordonnées **X** et **Y** pour le positionnement dans la grille. Chaque animal maintient une référence vers le **Terrain** sur lequel il évolue et vers la **GestionJeu** pour interagir avec l'état global du jeu. Le constructeur protégé garantit que seules les classes dérivées peuvent être instanciées, respectant le principe d'abstraction.


La méthode abstraite **Agir()** force chaque type d'animal à implémenter son comportement spécifique, illustrant parfaitement le polymorphisme. Les méthodes utilitaires



protégées facilitent la navigation : `CasesAdjacentes()` retourne les 8 cases entourant une position, `CasesAdjacentesVides()` filtre celles disponibles pour le déplacement, et `PositionAleatoireVide()` trouve une case libre sur tout le terrain. Cette architecture modulaire permet d'ajouter facilement de nouveaux types d'animaux en héritant de cette classe de base.

## B) Abeille

La classe `Abeille` hérite de la classe abstraite `Animal` et implémente un agent bénéfique pour l'écosystème du potager. Cette classe illustre parfaitement le concept d'héritage et de polymorphisme dans notre architecture.

Au niveau de ses caractéristiques, l'abeille est définie avec le nom "Abeille", le type "Insecte", le comportement "Pollinisateur" et le visuel . Elle possède une `distanceMax` de 5 cases, limitant son rayon d'action lors de chaque activation.

Son comportement se décline suivant la méthode `Agir()` surchargée qui implémente un comportement complexe sur plusieurs étapes :


1. Déplacement aléatoire jusqu'à 5 cases maximum par action
2. 40% de chance de planter une rose sur chaque case visitée
3. Mise à jour de sa position avec libération de l'ancienne case
4. Affichage en temps réel du déplacement avec animation (400ms entre chaque mouvement)
5. Actualisation visuelle du terrain via `Jeu.AfficherTerrainConsole()`

Cette implémentation simule la pollinisation naturelle et enrichit l'écosystème du jeu de manière autonome.



## C) Taupe

La classe `Taupe` représente l'agent nuisible du système et démontre l'équilibre nécessaire dans notre simulation écologique.

Elle a une caractéristique destructive, c'est à dire que, définie avec le nom "Taupe", le type "Fouisseur", le comportement "Mangeuse" et le visuel , elle symbolise les dégâts causés aux cultures. En effet, sa méthode `Agir()` implémente une logique simple mais efficace :

1. Recherche systématique de plantes dans toutes les cases adjacentes (8 directions)
2. Suppression immédiate de la première plante trouvée via `EnleverPlante()`
3. Auto-destruction après destruction pour simuler le départ de l'animal
4. Affichage du visuel "trou" temporaire indiquant les dégâts

L'équilibre du jeu est maintenu par une apparition aléatoire (1 chance sur 6) lors de la progression hebdomadaire, créant un défi constant sans être excessive.

## D) Escargot

La classe `Escargot` complète notre écosystème d'agents nuisibles en représentant un ravageur modéré mais persistant dans notre simulation.

Ce petit chenapan est défini avec le nom "Escargot", le type "Mollusque", le comportement "Grignoteur" et le visuel 🐌, cet agent symbolise les dégâts graduels causés aux cultures. Sa méthode `Agir()` implémente une stratégie de dégradation progressive :

1. Identification systématique de toutes les plantes dans les cases adjacentes (8 directions)
2. Réduction de 10% de la santé de chaque plante trouvée via modification directe de `SanteActuelle`
3. Calcul et affichage des dégâts infligés avec suivi détaillé de l'évolution sanitaire
4. Auto-destruction après action pour simuler le départ naturel du ravageur

Comme avec la taupe, l'équilibre écologique est préservé par une apparition aléatoire (1 chance sur 6) lors de la progression hebdomadaire. Contrairement à la taupe qui détruit immédiatement, l'escargot affaiblit progressivement les cultures, créant un défi de gestion à long terme qui nécessite une surveillance constante et des soins réguliers pour maintenir la productivité du potager.

# V. Fonctionnalités supplémentaires

## A) Economie

Le système économique du jeu repose sur la classe `Economie` qui gère les transactions financières et l'équilibre commercial du potager. Cette classe encapsule la logique monétaire avec un capital de départ de 10€ et des dictionnaires de prix différenciés pour l'achat et la vente.

Le système de tarification reflète la réalité économique agricole avec des marges bénéficiaires variables selon les produits. Les roses génèrent le profit le plus élevé (achat 3€, vente 8€) tandis que les carottes offrent une marge plus modeste (achat 1€, vente 3€). Cette différenciation encourage la diversification des cultures et la planification stratégique.

Les méthodes `AcheterGraine()` et `VendreProduit()` gèrent respectivement les transactions d'approvisionnement et de commercialisation. La première vérifie la solvabilité du joueur avant d'autoriser l'achat, tandis que la seconde calcule automatiquement les gains

et met à jour le capital. Cette mécanique incite à une gestion prudente des ressources et à l'optimisation des cycles de production.

## B) Meteo

Le module météorologique, implémenté dans la classe statique `Meteo`, simule les aléas climatiques et leur impact sur les cultures. Cette fonctionnalité enrichit considérablement la dimension réaliste du jeu en introduisant des variables externes incontrôlables.

L'énumération `TypeIntemperie` définit six conditions climatiques : Normale, Pluie, Sécheresse, Grêle, Gel et Canicule. La méthode `GenererIntemperie()` utilise des probabilités saisonnières réalistes - par exemple, 25% de chance de gel en hiver contre 15% de canicule en été - pour créer une variabilité climatique crédible.

La méthode `AppliquerIntemperie()` traduit ces événements en effets concrets sur le terrain. La pluie arrose automatiquement toutes les plantes, économisant du temps et des efforts au joueur. À l'inverse, la grêle inflige des dégâts aléatoires à 30% des cultures, nécessitant des interventions correctives. Cette mécanique force le joueur à s'adapter aux conditions changeantes et ajoute une dimension stratégique à la planification des cultures selon les saisons.

# VI. Classes de gestion du jeu

## A) Affichage

La classe statique `AffichageTerrain` constitue la couche de présentation visuelle de notre application et centralise toutes les fonctionnalités d'affichage en mode console. Cette architecture modulaire sépare clairement la logique métier de la présentation, respectant le principe de séparation des préoccupations.

La classe définit également plusieurs constantes pour les couleurs des différents types de terrain (`CouleurSable`, `CouleurArgile`, `CouleurMarecage`) permettant une identification immédiate du type de sol. Le système de colorisation des plantes utilise un code couleur intuitif basé sur leur état de santé : vert pour une santé optimale (>75%), jaune pour un état moyen (50-75%), et rouge pour les plantes malades ou en détresse (<25%). Nous avons ensuite 3 méthodes d'affichage principales:

- `AfficherTerrain()` : Méthode centrale qui orchestre l'affichage complet avec effacement de l'écran, affichage de l'en-tête, du cadre décoratif, de la grille des parcelles et des conditions environnementales
- `AfficherCadreTerrain()` : Génère un cadre ASCII artistique avec des caractères Unicode ( `▬` | `└` ) pour délimiter visuellement l'espace de jeu

- `AfficherConditionsActuelles()` : Présente les paramètres environnementaux avec une colorisation contextuelle (bleu pour le froid, rouge pour la chaleur)

L'implémentation gère intelligemment la superposition des éléments visuels en donnant la priorité aux animaux sur les plantes, permettant de visualiser les interactions écologiques en temps réel. Cette hiérarchisation garantit que les événements dynamiques (déplacement d'abeilles, apparition de taupes) restent visibles par-dessus l'état statique des cultures.

Au niveau de l'interface utilisateur, la méthode `AfficherMenuPrincipal()` génère un menu ASCII encadré présentant toutes les actions disponibles avec leurs raccourcis clavier, offrant une navigation intuitive et ergonomique.

## B) GestionJeu

La classe `GestionJeu` représente le cœur de l'application et orchestre l'ensemble des mécaniques ludiques. Elle encapsule l'état global du jeu et coordonne les interactions entre tous les sous-systèmes. Nous aurions pu la diviser en plusieurs classes afin de simplifier la navigation car le programme de cette classe est assez long. Mais au fur à mesure du développement du projet nous ne l'avons pas fait, et le faire en fin de projet nous semblait trop risqué et compliqué.

- Au niveau de l'**architecture des données**, cette classe maintient l'état complet de la session de jeu avec le terrain actuel, la temporalité (saison, semaine, année), l'inventaire du joueur, l'économie, et les statistiques. L'utilisation de propriétés en lecture seule garantit l'encapsulation tout en permettant l'accès aux données depuis les autres modules.
- Est aussi présent le **système temporel** qui simule un calendrier réaliste avec 4 saisons de 13 semaines chacune. La méthode `PasserSemaine()` orchestre la progression temporelle en déclenchant successivement la génération météorologique, l'évolution des plantes, et l'apparition aléatoire d'animaux (probabilité de 1/6 pour chaque espèce).
- Pour l'**interface de la console**, la méthode `LancerInterfaceConsole()` implémente une boucle de jeu complète avec gestion des commandes clavier. Chaque action (A pour arroser, P pour planter, T pour le temps) déclenche des animations et des feedbacks visuels appropriés. La méthode `ExecuterActionSurParcelle()` centralise la logique de sélection de coordonnées et la validation des actions.
- De plus, la classe `GestionJeu` paramètre le **système économique** intégré au jeu avec les méthodes `AfficherMagasin()` et `VendreProduits()` qui implémentent un système commercial complet avec gestion des stocks, vérification de solvabilité, et mise à jour automatique de l'inventaire. Cette intégration économique ajoute une dimension stratégique à la gestion des ressources.
- Enfin, la classe inclut de nombreuses méthodes utilitaires pour la **génération d'emojis** contextuels (saisons, terrains, plantes, animaux) qui enrichissent l'expérience visuelle. La méthode `ObtenirResume()` génère un **rapport détaillé** de la session incluant les statistiques d'apparition des animaux et l'état économique.

## C) Program.cs

Le fichier `Program.cs` constitue le point d'entrée de l'application et gère l'initialisation, la présentation, et la séquence de démarrage du jeu.

- **initialisation** : le programme commence par configurer l'environnement console avec l'encodage UTF-8 pour supporter les caractères Unicode et les emojis, puis ajuste la taille du buffer pour éviter les problèmes d'affichage. Cette configuration garantit une expérience visuelle optimale sur différents systèmes d'exploitation.
- **écran d'accueil artistique** : La fonction `DessinerTitre()` génère un logo ASCII artistique avec animation progressive (délai de 100ms par ligne) créant un effet d'apparition spectaculaire. Cette présentation professionnelle établit immédiatement l'identité visuelle du jeu.
- **décoration** : Les fonctions de dessin (`DessinerTournesol()`, `DessinerRose()`, `DessinerTulipe()`, etc.) créent un paysage décoratif ASCII avec positionnement précis et colorisation contextuelle. Ces éléments visuels utilisent des caractères spéciaux pour simuler la profondeur et créer une ambiance bucolique.

Déroulé : le programme suit une séquence logique claire : écran d'accueil → saisie du nom du joueur → lancement de l'interface de jeu → affichage du résumé final. Cette structure garantit une expérience utilisateur fluide et cohérente. À la fermeture du jeu, le programme affiche automatiquement un résumé complet de la partie incluant les statistiques de performance, l'état du terrain, et les accomplissements du joueur. Cette fonctionnalité encourage la rejouabilité en permettant de mesurer les progrès entre les sessions.

## VII. Bilan du projet et difficultés rencontrées

Ce projet était challengeant, tant à la fois sur le plan technique en lui-même, notamment dans la prise en main de la programmation orientée objet, qu'au niveau de sa longueur et de toutes les différentes caractéristiques demandées. La gestion des nombreuses classes nous a demandé beaucoup de travail et de persévérance car les lier entre elles, notamment dans les classes GestionJeu et Affichage, était compliqué. Au niveau des améliorations, nous avons choisi de ne pas développer de mode urgence mais de se concentrer plutôt sur une diversification des commandes, notamment avec une météo développée et un système économique, et d'améliorer le graphisme et la clarté du jeu. Nous aurions pu également comme précisé au début du paragraphe VI) B) créer plusieurs fichiers cs différents afin de séparer la classe gestion Jeu qui est très importante, mais cette idée ne nous est venue que trop tard dans la chronologie du projet et de telles modifications n'étaient pas possibles dans le temps imparti.

Nous retenons néanmoins de ce projet, la liberté créative qui nous a permis de nous amuser tout en progressant, et les nouvelles notions de POO qui furent bien assimilées. Travailler en binôme grâce à GitHub nous a permis encore une fois d'apprendre ensemble, d'apprendre l'une à l'autre et de se partager les tâches.

