BONNET Émile
MARCHAIS Mattieu

CO5SFIN0 - Programmation avancée S6

Compte-Rendu - Projet jeu C#

“ENSemenC”

Pr. TETELIN Angélique

2024-2025

Table des matières

I. Introduction

A. Demande initiale

Pour ce projet, il nous a été demandé de développer un jeu en C# en utilisant le framework .NET 8. Le thème imposé est celui d'un simulateur de potager, où le joueur doit gérer ses plantations au fil des jours. Dès le début, le joueur a accès à différents types de semis, qu'il peut planter selon des paramètres variés comme le terrain, la luminosité ou l'humidité. Chaque plante pousse plus ou moins rapidement selon que ses besoins sont respectés ou non.

En parallèle, le joueur doit rester attentif à ce qu'il se passe dans son jardin : régulièrement un mode urgence s'active automatiquement. Le joueur doit alors intervenir rapidement pour éviter de perdre ses cultures. Ce projet a donc pour but de combiner une gestion stratégique au tour par tour avec des événements aléatoires nécessitant des réactions rapides, le tout dans une interface console.

II. Partie créative

Dû aux consignes de ce projet focalisé sur un aspect technique, cela laisse place à beaucoup d'interprétation, il est donc difficile de déterminer quels aspects de notre code sont créatifs et lesquels sont des fonctionnalités requises. Certaines parties de notre code ne sont sans aucun doute des parties créatives, comme par exemple la capacité au joueur d'acheter des améliorations pour la plupart des actions qui sont présentes dans sa barre d'action. Par exemple, pour 50 pièces, il peut acheter la capacité de labourer toutes les cases autour de lui en plus de celle sur lequel il se trouve. Cependant, aucune consigne n'était donnée sur comment nous devons coder la capacité au joueur de planter et de jardiner. Nous avons donc opté pour représenter le joueur physiquement sur la carte et ainsi permettre un certain confort de gameplay, reproduisant les codes des jeux-vidéos.

Pour être plus exhaustif sur notre code, voici certaines fonctionnalités qui sont explicitement créatives:

- La présence d'ASCII dans notre affichage pour rendre le tout plus esthétique.
- Un menu qui permet d'aller consulter les règles du jeu.
- Une boutique qui permet d'acheter des nouvelles plantes, mais aussi des améliorations et la victoire.
- Un affichage de notre grille qui est de couleur, avec chaque type de terrain représenté par des couleurs différentes.
- Nous avons subdivisé les actions pour notre jardin en labourage d'abord, puis planter, ensuite arroser les plantes et enfin les récolter.
- Les maladies se propagent plus facilement à des voisines de la même espèce de plante.

Pour ce qui est des fonctionnalités qui ne sont pas explicitement créatives mais plutôt des choix artistiques, nous avons:

- Quand le joueur ne fait aucune action, le temps passe quand même.
- Les actions des joueurs ne font pas pousser les plantes, il faut explicitement qu'il attende.
- Nous avons essayé au maximum d'éviter au joueur d' avoir à taper avec son clavier pour faire quelque chose.

Nous allons maintenant vous présenter le code dans sa globalité pour que vous puissiez comprendre nos choix créatifs et comment nous avons construit sur les fonctionnalités attendues.

III. Présentation du programme

A. Structure des classes

Si nous prenons un point de vue global sur le jeu, il est composé de 7 classes plus le Program.cs:

- EspaceDeJeu.cs
- Joueur.cs
- MainJeu.cs
- Maladie.cs
- Plante.cs
- Rongeur.cs
- Terrain.cs

Ces 7 classes composent l'entièreté de notre jeu, et chacune à un rôle bien défini.
EspaceDeJeu est responsable de tout ce qui concerne le terrain de jeu sur lequel le joueur évolue. Cela comprend la taille de grille tout comme le conteur des jours ou encore les variables d'affichage de l'inventaire.
Joueur est explicite dans son fonctionnement car il comprend et est responsable de tout ce qui concerne le joueur, que cela soit comment il bouge mais également toutes les actions qu'il peut effectuer.
MainJeu est composé principalement de 2 parties, la première étant celle de faire tourner le jeu en lui-même. Il n'appelle pas toutes les autres classes car ce n'est pas nécessaire. Cependant, c'est la seule classe appelée dans Program.cs. La deuxième partie concerne le menu et tout ce qui concerne les ASCII, ainsi que les affichages de victoire ou de défaite.
Maladie est relativement simple, car elle n'est responsable que de créer une maladie unique à chaque plante mais également des fonctions qui permettent de propager ces maladies.
Plante est, tout comme Joueur, explicite sur ces responsabilités. Il concerne toutes les caractéristiques des plantes et l'affichage de leur statistiques au Joueur.
Rongeur n'est composé seulement des fonctions pour permettre au rongeur de manger les plantes et de se déplacer.
Et pour finir, Terrain est simplement composé d'une variable qui représente le type de terrain présent, la majorité des informations ayant été défini dans EspaceDeJeu.
Nous allons maintenant entrer plus en détail sur chaque classe et les fonctions qui y sont présentes.

B. EspaceDeJeu

1. Présentation de la classe et de ses variables

Cette classe comprend toutes les actions se déroulant dans l'espace de jeu du joueur. C'est pour cela qu'elle comprend de nombreuses variables contenant des informations sur le gameplay. Cependant, elle possède aussi des variables pour gérer l'affichage de données importantes pour l'interface graphique.

Variables pour le gameplay :

```
public Terrain[,] CarteTerrains { get; private set; }
public string[,] Grille { get; private set; }
public int TailleX { get; private set; }
public int TailleY { get; private set; }
public int NombrePlanteMorte { get; set; }
public int Jours { get; set; } = 0;
public int luminosité { get; set; } = 0;
public bool ModeUrgence { get; set; } = false;
public bool Retraite { get; set; } = false;
public bool[,] EstLaboure { get; set; }
public Plante Plantenull { get; }
public List<Plante> Plantes = new List<Plante>();
public Joueur Joueur { get; set; }
```

Variables pour l'affichage :

```
public int SelectInventaire
public int SelectPlante
public List<string> PlantesDispo { get; set; } = new List<string> {"|Carotte|", "|Tomate|", "|Radis|",
"|Salade|"};

public List<string> ObjetBoutique { get; set; } = new List<string> { "Piment", "Melon", "Citrouille", "Fraise",
"Retraite" };
public List<string> Inventaire { get; set; } = new List<string> {"|Labourer| ", "|Planter| ", "|Récolter| ",
"|Arroser| ", "|Frapper| ", "|Boutique| "};
```

2. Méthodes et procédures

```
  public void InitialiserGrille()
/*Fonction servant à initialiser la grille de départ dans l'espace de jeu*/
```

Paramètres d'entrée :

Description : La fonction remplit la carte des terrains de différents types de terrain en créant des patch de terrains. De plus, chaque case de la grille est initialisée à trois espaces vides.

Paramètres de sortie :

```
public void DefinirGrille(int x, int y)
/*Fonction servant à redéfinir la grille à chaque étape*/
```

Paramètres d'entrée : Deux entier x, y correspondant respectivement à l'abscisse et l'ordonnée de la grille

Description : La fonction définit la grille en fonction des différents changements qui ont possiblement eu lieu. Cela comprend donc :

- Les cases Labourées
- Les plantes
- Le joueur

L'ordre d'importance de l'affichage se fait du bas vers le haut de la liste.

Paramètres de sortie :

```
public void AfficherJeu()
/*Fonction servant à afficher tout l'espace de jeu*/
```

Paramètres d'entrée :

Description : La fonction affiche tout l'espace de jeu. Elle comprend donc une double boucle pour afficher la grille. En même temps, elle change la couleur de fond et la couleur de toutes les instances présentes dans la grille. Elle fait ensuite des informations importantes pour le gameplay et fait appelle aux fonctions permettant d'afficher l'inventaire et les statistiques de la plante sur laquelle le joueur se trouve.

Paramètres de sortie :

```
public void AfficherInventaire(int selection)
/*Fonction pour afficher l'inventaire et surtout l'action que l'on séléctionne*/
```

Paramètres d'entrée : Un entier qui permet de savoir quelle action on sélectionne pour afficher l'inventaire en conséquence

Description : Cette fonction affiche l'inventaire avec en rouge l'action actuellement sélectionnée.

Paramètres de sortie :

```
public void UpdatePlantes()
```

Paramètres d'entrée :

Description : Cette fonction permet de mettre à jours toute les plantes de l'espace de jeu (variable List<Plante> Plantes) en utilisant la fonction MetAJours de la classe Plante

Paramètres de sortie :

```
public Plante SelectionnerPlante(int x, int y)
```

Paramètres d'entrée : Deux entier x,y correspondant à une position

Description : Cette fonction parcourt toute la liste de plantes de l'espace de jeu et compare toutes les positions pour pouvoir sélectionner la plante voulu. S'il n'y a aucune plante la fonction renvoie la variable PlanteNull créée au début de la classe.

Paramètres de sortie : Une plante situé à la position (x, y), ou la PlanteNull si aucune plante ne se trouve en (x, y)

C. Joueur

1. Présentation de la classe et de ses variables

Cette classe doit gérer toutes les composantes du joueur ainsi que lui permettre de se déplacer et de réaliser toutes les actions qui sont sur la barre d'action.

Voici ses variables:

```
public int[] TableauRecolte { get; set; } // Tableau de récolte en foction de la plante
public int PrixRetraite { get; set; } = 1000; // Prix de la victoire
public int PositionX { get; set; } // Position du joueur sur la grille
public int PositionY { get; set; } // Position du joueur sur la grille
public int Argent { get; set; } = 5;

public HashSet<string> AmeliorationsAchetées { get; set; } = new HashSet<string>(); // Liste des
améliorations achetées
public bool AFrappe { get; set; } // Indique si le joueur a frappé un rongeur
public EspaceDeJeu Grille { get; set; }
public string Affichage { get; set; } = " J ";
```

2. Méthodes et procédures

```
public void MoveJoueur(int tempsBoucle) // Déplace le joueur sur la grille
```

Paramètres d'entrée : Un entier, qui représente le temps avant le jeu continue de tourner si le joueur ne rentre pas d'input.

Description : La fonction est découpée en 2 parties, la première étant le while loop qui avance en fonction soit de l'input du joueur soit du temps qui passe. Ensuite, la seconde partie est la capacité du joueur de se déplacer ou de réaliser une action. Si le joueur n'as rien fait alors cette partie ne fait aucune action, mais s'il a entré une commande, alors la variable action va déterminer ce qu'il se passe.

Paramètres de sortie :

```
public void Action(int selection) // Action du joueur en fonction de la sélection
```

Paramètres d'entrée : Un entier, qui représente l'action qui a été choisie.

Description : La fonction dépend de si le mode urgence est activé. Si c'est le cas, alors seul l'action frapper fait quelque chose. Sinon cette fonction appelle les autres fonctions pertinentes pour chaque action.

Paramètres de sortie :

```
public void Labourer()
```

Paramètres d'entrée :

Description : La fonction détermine simplement que la case sur lequel est le joueur est labourée. Si le joueur possède l'amélioration pertinente alors les 9 cases autour sont également labouré.

Paramètres de sortie :

```
public void PlacePlante(Plante plante)
```

Paramètres d'entrée : Une plante

Description : La fonction ajoute simplement la plante choisie à la position du joueur. Il y a également une interaction avec le labourage dans le cas où le joueur aurait acheté l'amélioration correspondante , celle où la case reste labourée même après que la plante meurt/soit récolté.

Paramètres de sortie :

```
public void Recolter()
```

Paramètres d'entrée :

Description : Cette fonction, malgré sa longueur, est relativement simple. Elle permet simplement de gagner de l'argent en récoltant les plantes qui sont prêtes. Une amélioration peut également affecter le montant d'argent récupéré.

Paramètres de sortie :

```
public Plante ChoixPlante()
```

Paramètres d'entrée :

Description : Cette fonction est là où tous les différents types de plantes sont initialisés.

Paramètres de sortie : Une plante est renvoyée avec ces caractéristiques.

```
public void Arroser()
```

Paramètres d'entrée :

Description : Cette fonction augmente l'hydratation de la plante à la position du Joueur.

Paramètres de sortie :

```
public void Frapper()
```

Paramètres d'entrée :

Description : Cette fonction rend la variable booléenne AFrappe vrai si le joueur à fait cette action sur la position du rongeur.

Paramètres de sortie :

```
public void AccederBoutique()
```

Paramètres d'entrée :

Description : Cette fonction très longue est responsable du magasin où le joueur peut acheter des nouvelles plantes, des améliorations et la victoire. Beaucoup de ce qui rend cette fonction longue sont les modalités d'affichage que nous avons rajouter pour améliorer l'esthétique. Elle initialise d'abord les dictionnaires contenant tous les achats possibles ainsi que leur description, et affiche ensuite dans le bon ordre et la bonne couleur chaque achat possible, variable en fonction de ce qui a été acheté précédemment.

Paramètres de sortie :

```
public override string ToString() // Affichage du joueur en fontion du caractère qui le représente
```

Paramètres d'entrée :

Description : Cette fonction permet de déterminer à quoi ressemble le Joueur.

Paramètres de sortie : Renvoie le string du Joueur ("J").

D. Plantes

1. Présentation de la classe et de ses variables

La classe Plante représente chaque plante présente dans le jardin. Elle contient toutes les caractéristiques biologiques, visuelles et de progression de croissance. Elle gère également la logique de maladie, de croissance et d'environnement.

Variables des plantes:

```
Random rnd = new Random();
public int PositionX { get; set; }
public int PositionY { get; set; }
public int BesoinEau { get; set; }
public int Hydratation { get; set; }
public int BesoinLuminosite { get; set; }
public int EsperanceDeVie { get; set; }
public int Progression = 0;
public double TauxCroissance { get; set; }
public Maladie? MaladieActuelle { get; set; }
public string Type { get; set; }
public string Affichage // Affichage de la plante en fonction de son âge
{
 get
 {
 if (Type == "Plantenull") return " . ";
 return Type switch
 {
  "Carotte" => Progression < 100 ? "c" : "C",
  "Tomate" => Progression < 100 ? "t" : "T",
  "Radis" => Progression < 100 ? "r" : "R",
   "Salade" => Progression < 100 ? "s" : "S",
   "Piment" => Progression < 100 ? "p" : "P",
```

```
    "Melon" => Progression < 100 ? "m" : "M",
    "Citrouille" => Progression < 100 ? "h" : "H",
     "Fraise" => Progression < 100 ? "f" : "F",
     _ => "?"
    };
  }
}
public EspaceDeJeu Grille { get; set; }
```

2. Méthodes et procédures

```
public void MetAJour()
```

Paramètres d'entrée :

Description : Fonction appelée à chaque tour pour réduire l'hydratation et l'espérance de vie, appliquer les effets des maladies, recalculer le taux de croissance selon environnement et maladie, tenter de faire croître la plante, déterminer sa mort si insatisfaction.

Paramètres de sortie :

```
private void RecalculerTauxCroissance()
```

Paramètres d'entrée :

Description : Fonction qui calcule dynamiquement le TauxCroissance en fonction de : l'écart entre la luminosité actuelle et le besoin, l'écart d'hydratation, l'état de santé (maladie), et le terrain sur lequel la plante est posée.

Paramètres de sortie :

```
public void AfficherPlanteStatistique()
```

Paramètres d'entrée :

Description : Fonction qui affiche l'ensemble des statistiques de la plante au joueur quand il se trouve sur une case ayant une plante.

Paramètres de sortie :

E. Maladie

1. Présentation de la classe et de ses variables

La classe Maladie sert principalement à gérer les maladies que peuvent contracter les plantes, les maladies affectent les plantes pour une certaine durée et avec une certaine gravité d'infection. Chaque plante peut aussi propager sa maladie à ses voisines.

```
public string Type { get; set; }
public int Duree { get; set; }
public int Gravite { get; set; }
```

2. Méthodes et procédures

```
public void ContracterMaladie(Plante plante)
/*Fonction qui applique une maladie selon le type de la plante*/
```

Paramètres d'entrée : Une plante (celle qui va contracter la maladie)

Description : La fonction utilise simplement un switch case sur le type de la plante pour créer une nouvelle maladie unique à chaque type de plante

Paramètres de sortie :

```
public void AppliquerEffet(Plante plante)
/*Applique les effets d'une maladie (affecte le taux de croissance et permet la guérison */
```

Paramètres d'entrée : Une plante (celle qui va être affectée par la maladie)

Description : La fonction fait simplement diminuer la durée de la maladie de la plante d'une étape lorsqu'elle est appelée. Elle applique de plus un malus sur le taux de croissance de la plante en fonction de la gravité de la maladie.

Paramètres de sortie :

```
public void PropagerMaladie(Plante plante, Random rnd, EspaceDeJeu grille)
/*Fonction permettant d'appliquer une maladie selon une certaine probabilité aux plantes voisine*/
```

Paramètres d'entrée : Une plante (celle qui va propager la maladie), un espace de jeu (celui dans lequel est la plante) et un objet Random.

Description : La fonction regarde si la plante possède des voisine, si c'est le cas elle fait contracter à ses voisine la même maladie selon une certaine probabilité

Paramètres de sortie :

F. Rongeur

1. Présentation de la classe et de ses variables

En ce qui concerne la classe rongeur, elle est utile uniquement lors du mode urgence ou pendant lequel un rongeur va apparaître. Il va pouvoir bouger, dans la grille et il va falloir le frapper pour faire tomber ses PV à 0 et ainsi revenir au mode classique pour tranquillement faire pousser son potager.

```
public EspaceDeJeu EspaceDeJeu { get; set; }
public int PositionX { get; set; }
public int PositionY { get; set; }
public int PV {get; set;}
public string Affichage { get; set; } = " E ";
```

2. Méthodes et procédures

```
public void MoveRongeur()
/*Fonction permmettant de faire bouger aléatoirement le rongeur*/
```

Paramètres d'entrée :

Description : La fonction tire un entier aléatoire entre 1 et 4 et fait bouger le rongeur en fonction en changeant sa position après avoir vérifié que le mouvement était possible

Paramètres de sortie :

```
public void MangerPlante()
/*Fonction supprimant les plantes sur lesquelles le rongeur est passé*/
```

Paramètres d'entrée :

Description : Lorsque le rongeur se déplace sur une plante ou une case labourée, la fonction supprime la plante de la liste des plantes présentes dans l'espace de jeu, ou bien modifie la variable indiquant l'état de la case pour signaler qu'elle n'est plus labourée (false).

Paramètres de sortie :

G. Terrain

1. Présentation de la classe et de ses variables

Cette classe peut paraître vide aux premiers abords. En effet, actuellement un terrain n'est qu'une simple chaîne de caractère (son type). Cependant il nous a semblé intéressant de créer une classe pour les terrains dans un objectif d'amélioration future ou de reprise de code par d'autres personnes.

```
public string Type {get; set;}
```

2. Méthodes et procédures

Comme expliqué précédemment, un terrain n'est pour l'instant rien d'autre qu'une chaîne de caractère. Il n'y a donc pas de méthode associée à cette classe.

H. MainJeu

1. Présentation de la classe et de ses variables

Cette classe regroupe tout ce qui concerne la boucle principale du jeu ainsi que les affichages des menus et d'introduction ainsi que les écrans de victoire et défaite. C'est également la seule classe appelée par Program.cs . Elle constitue le cœur de l'exécution du jeu.

Variables:

```
Random rnd = new Random();
public Joueur Joueur { get; set; }
public EspaceDeJeu EspaceDeJeu { get; set; }
public Rongeur Rongeur { get; set; }
public int ConditionFinDeJeu { get; set; }
public string nextPrint { get; set; } = "";
public int selectNumber = 1; // -> Permet de savoir quel écran de selection est affiché
```

2. Méthodes et procédures

```
public void StartGame()
```

Paramètres d'entrée :

Description : Déclenche la boucle principale du jeu. Tourne selon un while loop qui lui même se subdivise en 2 while loop qui détermine dans quel mode de jeu le joueur est actuellement. Acheter sa retraite ou faire mourir 50 plantes arrête le jeu et déclenche l'écran de victoire/défaite pertinent.

Paramètres de sortie :

```
public void AfficherFinJeu(int condition)
```

Paramètres d'entrée : un int qui représente si le joueur à gagné ou perdu

Description : Affiche l'écran de victoire ou de défaite selon la condition passée. Montre l'ASCII art correspondant et récapitule la récolte en cas de victoire.

Paramètres de sortie :

```
public void PrintIntro()//-> Affiche l'introduction du jeu
```

Paramètres d'entrée :

Description : Affiche l'introduction du jeu (titre, contexte narratif, ASCII art de fleur). Utilise Console.SetCursorPosition pour centrer les éléments à l'écran.

Paramètres de sortie :

```
public void FleurAscii() // -> Affiche une fleur imposante pour l'écran d'accueil
```

Paramètres d'entrée :

Description : Affiche un grand ASCII art de fleur (graphisme d'accueil).

Paramètres de sortie :

```
public void PrintSelectScreen(int x)//-> Affiche l'écran de selection en fonction de l'entier x
```

Paramètres d'entrée : int x qui représente la position du joueur dans le menu

Description : Affiche le menu de sélection (jouer ou règles) avec surbrillance selon la sélection.

Paramètres de sortie :

```
public void SelectScreen()//-> Permet la selection des différents écrans de selection
```

Paramètres d'entrée :

Description : Permet au joueur de naviguer entre les différentes options du menu avec les touches z, s, espace.

Paramètres de sortie :

```
public void PrintColoredText(string input) // -> Permet d'afficher le texte en couleur
```

Paramètres d'entrée : string qui est le texte que l'on souhaite colorier

Description : Affiche le texte passé en argument avec mise en couleur de certains mots-clés.

Paramètres de sortie :

```
public void Regles() // -> Affiche les règles du jeu
```

Paramètres d'entrée :

Description : Affiche l'ensemble des règles du jeu à l'utilisateur avec du texte coloré, centré et structuré.

Paramètres de sortie :

```
public void WinAscii() // -> Affiche un ascii art de la victoire
```

Paramètres d'entrée :

Description : Affichent un ASCII art de victoire.

Paramètres de sortie :

```
public void LoseAscii() // -> Affiche un ascii art de la défaite
```

Paramètres d'entrée :

Description : Affichent un ASCII art de défaite.

Paramètres de sortie :

```
public void InterfaceJeu()
```

Paramètres d'entrée :

Description : Gère l'interface de lancement et de navigation du menu. Appelle SelectScreen() puis agit selon le choix du joueur.

Paramètres de sortie :

I. Program.cs

Ce fichier ne contient pas beaucoup de code. En effet, il ne sert qu'à appeler notre classe qui crée les boucles de gameplay (MainJeu). Et lance deux fonctions de ce dernier qui sont le lancement du jeu ainsi que l'affichage de l'écran de fin de jeu.

IV. Bilan du projet

A. Gestion de projets

Mise à part pendant les vacances d'avril et la journée avant le rendu, nous ne travaillons que très peu en dehors des séances de TD. En effet, chaque séance était suffisamment longue pour que notre projet avance correctement. En général, à chaque séance nous organisions un petit bilan pour se répartir les tâches à effectuer pendant la séance. Ce qui a permis tout au long du projet d'avoir des objectifs clairs et de ne jamais nous sentir en retard sur le travail à rendre.

B. Acquis pédagogiques

Tout au long de ce projet, nous avons pu développer différentes connaissances et compétences. Tel qu'une meilleure appréhension de la programmation orientée objet. Le développement de notre gestion de projet. Nous avons aussi pu nous améliorer en logique et algorithmique grâce à ce projet

C. Rétrospective

Maintenant que ce rapport et notre jeu est terminé, nous pouvons donner quelques retour sur ce projet. Tout d'abord, nous sommes tous les deux d'accord pour dire que la réalisation de ce projet à été plaisante pour nous. Bien que le nombre conséquent d'autres projets à rendre pendant la même période a pu nous ralentir, nous sommes très satisfait du travail que nous avons fait. Toutefois, nous nous sommes quand même senti limités par la contrainte de coder un jeu uniquement dans la console. Bien que nous comprenions la portée pédagogique de cette contrainte, cette dernière nous à restreint dans nos ambitions (peut-être à raison au vu du temps dont nous disposions) au début du projet. En contrepartie, la découverte de la programmation orientée objet nous a laissé beaucoup plus de liberté dans la réalisation de ce projet que celui du semestre précédent, ce que nous avons trouvé très enrichissant.