# Mini-Projet 2020 LU2IN002

Benedetti Corentin

# FARMERS

# Sommaire



# Introduction

Pour ce mini-projet consistant à créer une simulation de récolte se déroulant sur un terrain contenant des ressources que des agents doivent récolter, j'ai choisi de faire un petit jeu appelé "Farmers" demandant au joueur de rembourser une dette en vendant des récoltes produites par des paysans en le minimum de jours possibles.

Le projet a eu ainsi pour objectif de non seulement satisfaire les requis de l'exercice mais aussi de remplir sa fonction de jeu le mieux possible en imaginant toutes les possibilités à donner au joueur et en améliorant son ergonomie.

Ce qui devait être un travail d'environ 5h30 à duré bien plus de temps et en se voulant complet, s'est énormément complexifié (informatiquement parlant). Il compte donc 6 classes objets, dont la principale possède environ 700 lignes de code, et une classe de test à lancer par l'utilisateur. Ainsi, même si j'ai essayé de commenter le mieux possible, cela risque d'être long de tout lire, je souhaite donc bonne chance à l'examinateur de cet exercice.

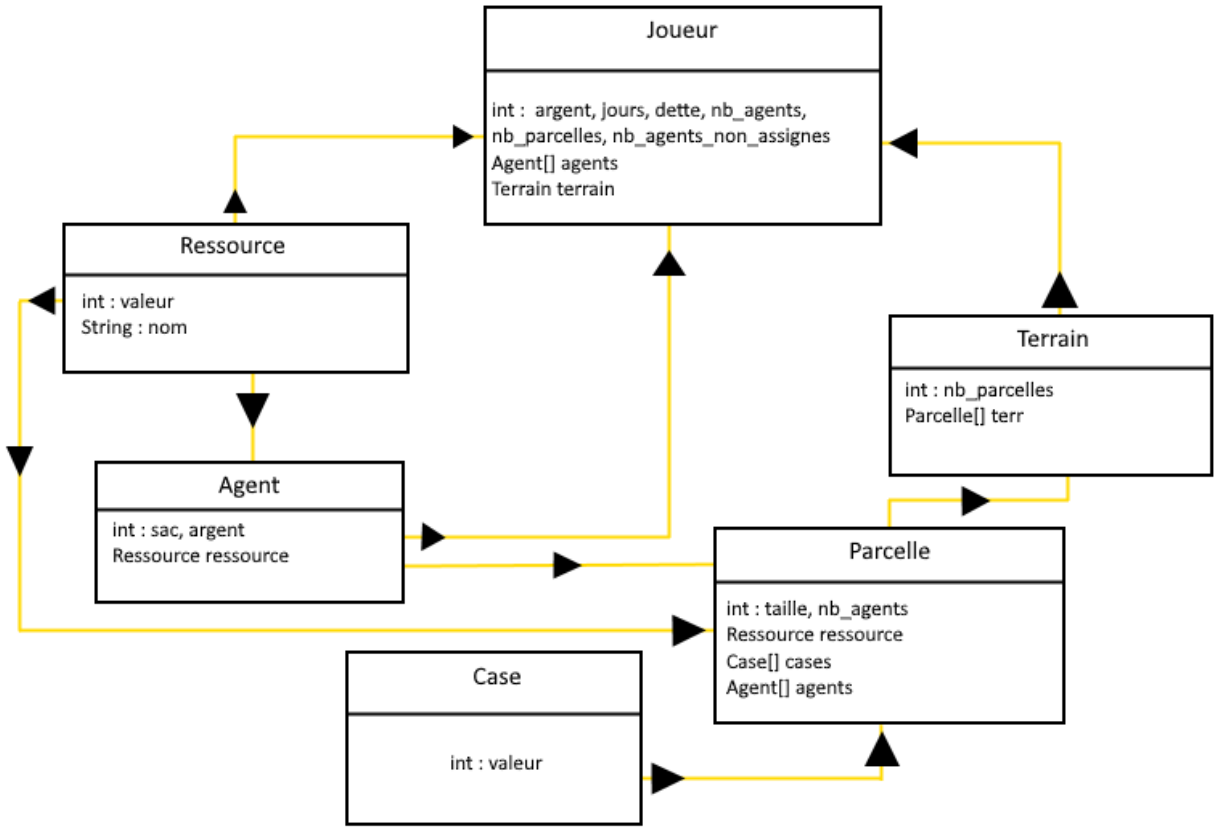
# Présentation

Le jeu se présente sous cette forme :

```
------------    Jour 1    -------------

Argent : 0 pieces
Dette : 10000 pieces
Nombre de paysans : 1
Nombre de parcelles : 2

 | 3 | 2 | 3 |
 | 2 | 2 | 1 |
 | 3 | 2 | 1 |

 | 2 | 3 | 2 |
 | 2 | 2 | 2 |
 | 3 | 3 | 3 |


Afficher votre terrain (1)          Boutique (2)          Vendre (3)          Payer votre dette (4)
```

Il faut naviguer dans les menus en mettant les numéros du clavier dans la console. Le terrain est affiché sous la forme de plusieurs matrices, chacunes représentant une parcelle.

# Diagramme UML



# Présentation des classes et fonctions

## Classe Objet Joueur

Classe la plus importante, elle fait le lien avec les autres et veille au bon déroulement de la partie. Elle contient la boucle principale du jeu mais aussi toutes les fonctions essentielles comme celles affichant les menus.

- Constucteurs : -`Joueur()` initialise les variables principales (`argent`, `dette`, `jours`, `nb_parcelles`, `nb_agents`) ainsi que les tableaux d'agents et de terrain. Il est important de rappeller que agent = paysan, terrain = parcelle, liste de terrain = terrain. Donc un terrain est une liste de parcelles et non un tableau à deux dimensions ; il est expliqué pourquoi dans la section "Problèmes".
  -`Joueur(int dette)` constructeur utilisé pour le tutoriel.

- Accesseurs : `getAr()` pour la variable `argent`, `getJour()` pour la variable `jours`, `getDette()` pour la variable `dette` et `toString()` donne un récapitulatif d'où en est le joueur.

- Fonctions standardes : -`vente()` permet de vendre les ressources récoltées par les paysans.
  -`boutique()` menu gérant la boutique et faisant appel aux autres sous-menus de la boutique (`amelioration()` et `amelioration2(int c)` gèrent les améliorations du joueur, `paysans()` permet d'acheter des paysans, `champs()` permet d'agrandir la taille de ses parcelles de terrain ou d'en acheter une supplémentaire et `ressources()` permet de débloquer des cultures supplémentaires).

- Fonctions Console : -`clearScreen()` nettoie la console d'une certaine façon (voir section "Problèmes"). -`titre()` affiche le titre.
  -`titre()` affiche une introduction.
  -`tutoriel()` affiche le tutoriel.
  -`afficheJour(int n)` affiche le jour `n`.
  -`findePartie()` renvoie un booléen indiquant si la dette est inférieure égale à 0. -`afficheTerr()` affiche le terrain ainsi qu'un menu pour interagir avec le terrain à l'aide de sous-menus (`afficheCult()` pour gérer les cultures et `clavier` pour gérer les paysans) ou pour récolter les cultures.
  -`affichePlan()` affiche le menu principal du jeu et permet d'accéder à la boutique, aux terrains, de vendre les récoltes ramassés, de rembourser une partie de la dette, de passer au jour suivant ou de quitter.
  -`resume()` affiche le nombre de jours pris pour rembourser la dette, seulement utilisé une fois à la fin du programme.
  -`jourSuivant()` effectue toutes les modifications liés au changement de jour (augmente la variable jours, fait pousser les cultures, augmente la dette avec un intérêt de 1 % et appelle clearScreen()).
  -`partie()` est la fonction contenant la boucle principale du jeu, celle qui appelle les autres fonctions et vérifie que le joueur n'a pas gagné avec la fonction findePartie().

## Classe Objet Terrain

Classe représentant les cultures du joueur et permettant d'effectuer tous les changements liés à celles-ci.

- Constucteurs : `-Terrain()` est le constructeur principal, il initialise le tableau de parcelles (contenant au maximum 9 parcelles) ainsi que la variable nb_parcelles comptant les parcelles ajoutés au terrain.
  `-Terrain(int nb_parcelles, Parcelle [] terr)` et `Terrain(int nb_parcelles)` sont des constructeurs utilisés seulement pour des tests et ne sont jamais appelés.

- Accesseurs : `-getTaille()` pour connaître le nombre de parcelles et `getAgent(int n)` pour connaître le nombre d'agents attribués à une parcelle.
  `-estVide(int n)` pour savoir si une parcelle est vide.
  `-toString()` pour afficher le terrain.
  `-nonAssigne(int n)` renvoie le nombre d'agents non assignés à une parcelle.

- Fonctions standardes : `-ajoutPar(int taille)` ajoute une parcelle et vérifie que le nombre maximum n'est pas atteint.
  `-remplir()` remplit le terrain aléatoirement au début du jeu.
  `-pousse(boolean arrosoir)` fait pousser les cultures plus ou moins vite si l'amélioration arrosoir+ a été prise.
  `-recolte(boolean paysan)` enlève des cultures du terrain et remplit le sac des agents. Résultat multiplié par 2 si l'amélioration paysan+ a été acheté.
  `-augmente ()` augmente la taille des parcelles.

- Fonctions appels, servant seulement à faire le lien entre Joueur et Parcelle et expliquées dans la partie sur l'objet Parcelle : `affichePar(int n)`, `retourneRess(int n)`, `changeRess(int n, int culture)`, `assigne(int n)`, `retire(int n)`.

## Classe Objet Parcelle

Classe faisant le lien entre la classe Terrain et la classe Case. Elle possède ses agents et un type de ressource.

- Constructeurs : `-Parcelle()` initialise la taille (nombre de cases) à 9, une nouvelle ressource, un tableau de cases, le nombre d'agents et un tableau d'agents.
- `-Parcelle (int taille, Ressource ressource)` et `Parcelle(int taille)` permettent de modifier certains paramètres.

- Accesseurs : `-getTaille()` renvoie la taille de la parcelle, `getRess()` renvoie la ressource, `getAgent()` renvoie le nombre de paysans assignés à la parcelle.
  `-toString()` affiche la parcelle dans la console (`affichePar(int n)`).

- Fonctions standardes : `-remplir()` remplit visuellement la parcelle.
  `-recolte(boolean paysan)` recolte les ressources.
  `-agentPlus()` ajoute un agent à la parcelle (`assigne(int n)`).
  `-agentMoins()` retire un agent à la parcelle(`retire(int n)`).
  `-pousse(boolean arrosoir)` fait pousser les cultures de la parcelle.
  `-augmente()` agrandit la parcelle
  `-changeRess(int n)` change la ressource attribué à la parcelle.

## Classe Objet Case

Cette classe représente une culture d'une parcelle du terrain.

- Constructeurs : `-Case()` initialise la valeur de la case à 0.
  `-Case(int valeur)` attribue la valeur donné en paramètre à la case.

- Accesseur : `-getVal()` retourne la valeur.

- Fonctions standardes : `-arrosoir` donne aléatoirement une valeur comprise entre 1 et 4 à une case.
  `-ramassage(boolean paysan, Agent a)` diminue la valeur d'une case pour simuler une récolte.
  `-pousseCase(boolean arrosoir)` augmente la valeur d'une case pour simuler la pousse.

## Classe Objet Ressource

Cette classe a pour but de changer la valeur des récoltes afin que le joueur gagne plus d'argent plus facilement. Elle possède un nom et une valeur.

- Constructeurs : `-Ressource()` initialise le nom et la valeur.
  `-Ressource(int valeur, String nom)` attribue le nom et la valeur donnés en paramètres.

- Accesseurs : `getNomR()` renvoie le nom et `getValeur()` renvoie la valeur.

- Fonctions standardes : `ble()`, `avoine()`, `orge()` et `mais()` changent la valeur et le nom de la ressource en fonction de la fonction.

## Classe Objet Agent

Classe très petite permettant simplement d'augmenter l'argent du joueur lors de la vente de récoltes. Elle lui est attribuée deux variables numériques : sac et argent ainsi qu'une ressource.

- Constructeurs : `-Agent()` initialise le sac du paysan à 0 et une nouvelle ressource.
  `-Agent(Ressource r)` initialise le sac du paysan à 0 et attribue la ressource donnée en paramètre.

- Accesseur : `getSac()` renvoie la valeur du sac.

- Fonctions standardes : `-changeRess(Ressource r)` change la ressource.
  `-remplitSac(int s)` remplit le sac du paysan.
  `-recolteAgent()` convertit le sac du paysan en argent pour le joueur.

Classe Test TestJoueur

Initialise simplement le Terrain et le Joueur afin de pouvoir lancer la partie. Elle fait appel aux fonctions `titre()`, `introduction()`, `tutoriel()` et `partie()`.

# Problèmes Rencontrés

Durant ce projet, certains problèmes ont survenus sans que je puisse trouver une solution me convenant rééllement, ce sont de ces problèmes que je vais parler ici.

**I – Le clearScreen** : Si vous avez testé le jeu vous pouvez voir que rien n'est rééllement effacé pendant une partie ; seulement de un bon nombre de sauts de ligne séparent les journées dans la console. Des propositions trouvés en ligne tels que flush() ou system("CLS") ne marchent pas puisque je travaille sur l'IDE Eclipse (Windows). La seule solution que j'ai trouvé est donc de faire une boucle de \n ce qui a pour seul avantage le fait de permettre au joueur de voir ce qu'il a fait précédemment.

**II – Les accents** : Je tenais juste à dire que s'il n'y a pas d'accents dans le jeu c'est pour éviter une mauvaise interprétation de ces derniers sur les environnements Linux et Max.

**III – Les consignes** : Certaines consignes du sujet n'ont pas été respectées. Je ne saurai dire si c'est une mauvaise lecture ou une mauvaise compréhension mais je me suis tellement investi dans ce projet que je n'ai pas eu le courage de recommencer quand je m'en suis rendu compte.

**IV – L'interface** : C'est moche. Je n'ai pas grand-chose à dire sur ça à part que je n'ai pas trouvé d'autre solution que l'interface textuel de la console.

**V – L'agencement des parcelles** : Au maximum, le joueur peut posséder 9 parcelles dans son terrain. Malheureusement je n'ai trouvé d'autre moyen pour les afficher que de le faire les unes à la suite des autres. J'aurais voulu que ça fasse un carré de 3x3 (ou 3x2 si il y en a 6) mais puisque c'est une interface textuel je ne peux rajouter toute une colonne.

**VI – Le scanner clavier** : La fonction scanner m'a permis de faire voyager le joueur dans les menus. Le seul problème est que si le joueur tape autre chose qu'un numéro, le programme s'arrête pour une erreur de type et tout comme le clearScreen je n'ai pas trouvé de fonction me permettant de récupérer le type du scanner afin de demander au joueur de changer sa valeur.