Université Lyon 1

LIFAPOO

Printemps 2023

Rapport

A VEGETABLE GARDEN

AIT ALLAOUA Salah

BGHIEL Yassin

I. Description du Projet

Le projet consiste à simuler un potager en vue de dessus en respectant le modèle MVC. Les fonctionnalités seront développer par la suite mais jusqu'à présent l'utilisateur peut :

- Planter ou cueillir différente variétés.
- Les plantes dispose différente étapes de croissance.
- Jusqu'à présent on dispose de 4 variétés : Salade, Carotte, Pastèque et Aubergine.
- Le potager est soumis par la température, le temps mais aussi par différente saison.
- Ce dernier est aussi soumis à une menace particulière : les nuisibles.
- L'utilisateur peut, si il le désire, vendre ces plantes mais aussi acheter des graines, des cases pour planter d'autre plante, des engrais et des pesticides.
- La vitesse de simulation peut être paramétrer pour accélérer le temps.

On rappelle que le plateau est représenter par une grille de cases pour la vue et par une grille de case gérer par Swing pour le modèle.

II. L'organisation au sein du groupe

Tout d'abord il était important de définir les rôles et responsabilités qu'on devait apporter à ce projet. Cela permet de déterminer qui est en charge de chaque tâche et éviter tout chevauchement ou confusion.

Ensuite, il est important de définir les objectifs du projet et de planifier les étapes clés nécessaires pour atteindre ces objectifs. Nous avons définis un planning en fonction du temps libre pour définir un calendrier clair pour chaque étape du projet ainsi que les extensions afin de s'assurer que le projet est sur la bonne voie et que les délais sont respectés dans les temps.

La communication est également un aspect crucial. Nous devons communiquer régulièrement pour s'assurer que tout se passe bien afin d'être informée des progrès réalisés et des problèmes éventuels rencontrés. Il est important de mettre en place un moyen de communication clair et efficace, que ce soit par téléphone ou tout autre moyen de communication. Nous avons utilisé le logiciel de gestion GIT pour nous partager le travail

Enfin, il est essentiel de suivre les progrès du projet et de s'assurer que chaque étape est atteinte à temps. Si des retards sont constatés, nous devons impérativement réagir rapidement pour éviter tout retard supplémentaire.

III. Les fonctionnalités

Nous avons utilisé un certain nombre de classe implémentant l'interface « Runnable ». Pour rappel L'interface « Runnable » définit une opération qui ne renvoie pas de résultat. Elle est utilisée pour exécuter des tâches en parallèle avec le thread principal. Cette interface est une interface fonctionnelle, ce qui signifie qu'elle ne définit qu'une seule méthode : "run()". Cette méthode doit être implémentée par toute classe qui implémente l'interface "Runnable". Elle contient le code qui sera exécuté par le thread.

i. Croissance des plantes

La croissance des plantes est une méthode définit pour chaque légume. Ce code prend un paramètre entier nommé "heure". La méthode contient plusieurs instructions conditionnelles qui déterminent l'état de l'usine en fonction du temps écoulé (représenté par la différence entre le paramètre "heure" et le "départ", qui est vraisemblablement un temps initial).

La première instruction conditionnelle vérifie si la durée de vie de la plante ("durée_de_vie") est supérieure au temps restant jusqu'à la fin de la journée (24 moins "départ"). Si c'est le cas, il vérifie si l'heure actuelle ("heure") est inférieure à la durée de vie de la centrale. Si c'est le cas, la variable "TMP" est réglée sur la somme du temps restant jusqu'à la fin de la journée et de l'heure actuelle. Sinon, "TMP" est réglé sur la valeur absolue de la différence entre l'heure actuelle et l'heure de "départ".

Si la plante n'est pas morte, le code vérifie alors si "TMP" est supérieur au temps nécessaire à la récolte de la plante ("durée_pour_recolter"). Si c'est le cas, le champ "Etatrecolte" est mis à vrai. Le code vérifie ensuite si "TMP" est supérieur à la durée de vie de la plante. Si c'est le cas, le champ "Morte" est défini sur vrai. Enfin, si "TMP" est supérieur à la moitié du temps nécessaire à la récolte de la plante, le champ "Pousse" est mis à vrai. Ainsi les légume peuvent pousser avec des variable qui leur sont propre.

```
if(24-depart<durée_de_vie)
{
    if(heure<durée_de_vie)      You, hier • bouton test / a faire PROPAGATION infection + …
    {TMP = 24-depart + heure;}
    else
    {
    TMP = Math.abs((24-heure)-(24-depart));
    }
}
else{TMP = Math.abs(heure-depart);}
if(TMP>durée_pour_recolter&&assez_soleil==true&&zabi ==1){
    setEtatrecolte(pret:true);
    TMP1 = TMP;
    zabi += zabi;
}
if(TMP-durée_pour_recolter>durée_de_vie&&pretaetrerecolter==true){setMorte(etatvie:true);}

if(getInfecter()==true){
};
if(TMP>durée_pour_recolter/2){setPousse(etat:true);}
check_propagation(heure);
check_Ensoleil(heure);
if(infecter==true&&pretaetrerecolter==true){morte = true;}
```

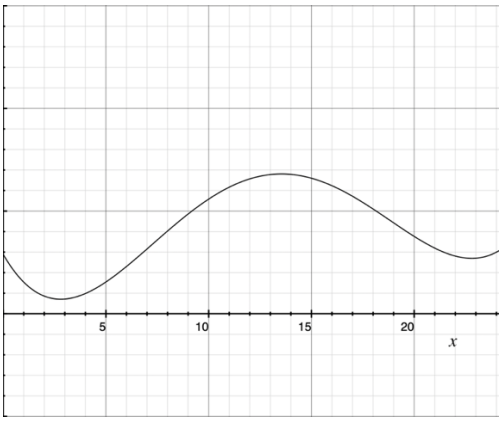
ii. Simulateur Météo

Nous avons décidé de mettre en place une simulation des plus réalistes, prenant en compte les saisons, ce qui implique une gestion particulière de l'évolution des températures, ainsi que la fréquence des précipitations, mais aussi les cycles d'inversion soleil-lune.

L'évolution des températures :

Nous avons pris comme année de référence 2022 pour la ville de Lyon, en choisissant un jour médian entre chaque saison pour être le plus fidèle possible (1er février...), et nous avons effectué une

interpolation sur les différentes températures au fur et à mesure de la journée, ce qui donne un graphe de cette forme-là :



Cependant, chaque graphe pour chaque saison est différent, car en raison de la présence du soleil, la chute de température diffère, même entre l'automne et le printemps, il existe une variation de l'évolution sur 24 heures.

```
switch(getSaison()){
case HIVER: result= (x - 24) * (((0.000504973 * (x - 6) - 0.00363683) * (x - 15) - 0.0269444) * x + 0.00416667) + 3; break;
case PRINTEMPS:result =  -0.0057356 * Math.pow(x, b:3) + 0.135262 * Math.pow(x, b:2) + 0.0615741 * x + 8;break;
case ÉTÉ: result= (x - 24) * (((-0.00774177 * (x - 15) - 0.124537) * x) + 0.0125) + 18.3;break;
case AUTOMNE: result=  (x - 24) * (((-0.00360082 * (x - 15) - 0.05) * x) + 0.0166667) + 13.4;break;
```

Fréquence de d'intempérie :

Nous nous sommes appuyés sur des données montrant les différentes variations de précipitations, que nous avons synthétisées en pourcentage pour évaluer le risque de pluie.

```
switch(getSaison())
{
    case HIVER: risquePrecipitation = 80;break;
    case PRINTEMPS: risquePrecipitation = 60;break;
    case ÉTÉ: risquePrecipitation = 10;break;
    case AUTOMNE: risquePrecipitation = 55; break;
}
Random r = new Random();
int risque = r.nextInt(bound:100);
if(risque<=risquePrecipitation){this.pluie=true;}
else{this.pluie = false;}
```

Cycle jour-nuit :

De la même manière, nous avons simplifié les cycles pour avoir une durée d'exposition au soleil plus importante selon les différentes saisons.

```
switch(getSaison())
{
    case HIVER: if(simPot.getHour()<8 || simPot.getHour()>16) ensolleillement = false; else{ensolleillement=true;} break;
    case PRINTEMPS:if(simPot.getHour()<6 || simPot.getHour()>20) ensolleillement = false;else{ensolleillement=true;} break;
    case ÉTÉ: if(simPot.getHour()<5 || simPot.getHour()>22) ensolleillement = false; else{ensolleillement=true;}break;
    case AUTOMNE: if(simPot.getHour()<6 || simPot.getHour()>20) ensolleillement = false;else{ensolleillement=true;} break;
}
```

Et pour chaque plante un temps d'ensoleillement est nécessaire à sa pousse.

Base de donnés et références ainsi que graphe médian :

https://prevision-meteo.ch/climat/journalier/lyon-bron/2022-01

https://www.researchgate.net/figure/Evolution-moyenne-au-cours-dune-journee-de-la-temperature-de-la-couche-seche_fig9_333746814

iii. Horloge

La classe "Clock" implémente l'interface "Runnable", ce qui signifie qu'elle peut être utilisée pour créer un nouveau thread qui s'exécutera en parallèle avec le thread principal.

Cette classe permet d'initialiser l'heure et les minutes de l'horloge en fonction de l'heure actuelle du système. La méthode "run()" est la méthode principale du thread. Elle tourne en boucle et incrémente les minutes de l'horloge toutes les secondes (ou à la fréquence définie par "pause"). Lorsque les minutes atteignent 60, l'heure est incrémentée de 1 et les minutes sont remises à 0. Lorsque l'heure atteint 24, elle est remise à 0. On peut donc avec cette implémentation et grâce à un accesseur, modifier la vitesse du temps donc grâce à ça on peut accélérer le temps.

iv. Simulateur Pesticides

La classe « SimulateurPesticide » implémente l'interface "Runnable", ce qui signifie qu'elle peut être exécutée comme un thread séparé. La classe possède un constructeur qui prend un objet "SimulateurPotager" en paramètre.

Cet objet représente un simulateur de jardin et contient un tableau 2D d'objets "Case" qui représentent les différentes cellules du jardin. La classe a une méthode appelée "Soigner" qui prend un objet "Case" comme paramètre.

Cette méthode vérifie si le cas est un "CaseCultivable" (une cellule cultivable dans le jardin) et si c'est le cas, elle récupère l'objet "Legume" (légume) planté dans la cellule et définit son attribut "infecter" sur « false », ce qui signifie que le légume est guéri de toute infection par les pesticides. La classe a une autre méthode appelée "ProccessusSoin" qui itère sur tous les cas du jardin et appelle la méthode "Soigner" sur chaque cellule cultivable.

Cette méthode est le processus principal d'application de la cure de pesticides sur les légumes du jardin. Enfin, la classe implémente la méthode "run" de l'interface "Runnable", qui appelle simplement la méthode "ProccessusSoin", la rendant exécutable en tant que thread séparé.

```java
public void Soigner(Case a){
    if(a instanceof CaseCultivable) {
        CaseCultivable cellule = (CaseCultivable) a;
        Legume legume = cellule.getLegume();
        if (legume != null){
            legume.setInfecter(contaminer:false);

        }
    }
}

public void ProccessusSoin(){
    for (int i = 0; i < grilleCases.length; i++) {
        for (int j = 0; j < grilleCases[i].length; j++) {
            if (grilleCases[i][j] instanceof CaseCultivable) {
                Soigner(grilleCases[i][j]);
            }
        }
    }
}
```

v. Simulateur Engrais

La classe « SimulateurEngrais contient une référence à un objet SimulateurPotager et une variable de type Legume. La méthode getPrix() permet d'obtenir le prix de l'engrais, qui est fixé à 15. La méthode Utilisation(Legume l) permet d'appliquer de l'engrais sur un objet Legume.

Si l'objet Legume en entrée n'est pas nul et n'a pas déjà été fertilisé, la méthode appelle la méthode Engrais() de l'objet Legume pour le fertiliser et met à jour son état de fertilisation en le passant à « true ». La méthode Engrais() va juste la diviser la durée de pousse par deux ce qui va accélérer la pousse.

```java
public void Utilisation(Legume l){
    if(l!=null){
        if (l.getEngrais() == false){
            l.Engrais();
            l.setEngrais(vite:true);
        }
    }
}
```

vi. Simulateur Nuisible

Dans la classe Nuisible il y a deux méthodes principale Infection et Propagation qui se caractérisent par :

Infection :

Nous générons un nombre aléatoire de cases à infecter afin de sélectionner au hasard ces cases à infecter, ce qui provoque une propagation aux cases adjacentes, mais peut aussi entraîner la mort de la plante à la fin de son cycle de pousse.

```
do {
    x = r.nextInt(sizeX);
    y = r.nextInt(sizeY);


} while (grille_cases[x][y] instanceof CaseNonCultivable);
CaseCultivable cellule = (CaseCultivable) grille_cases[x][y];
```

Propagation :

Cette méthode consiste à propager l'infection sur les cases avoisinantes pour obtenir un résultat réaliste en fonction des conditions réalistes que nous avons voulu implémenter, et elle se produit à chaque laps de temps défini.

```
if(legume.getPropagation()==true)
{
   if(grilleCases[i-1][j] instanceof CaseCultivable){CaseCultivable c1 = (CaseCultivable) grilleCases[i-1][j]; c1.setDepartPropagation();}
   if(grilleCases[i][j-1] instanceof CaseCultivable){CaseCultivable c1 = (CaseCultivable) grilleCases[i][j-1]; c1.setDepartPropagation();}
   if(grilleCases[i+1][j] instanceof CaseCultivable){CaseCultivable c1 = (CaseCultivable) grilleCases[i+1][j]; c1.setDepartPropagation();}
   if(grilleCases[i][j+1] instanceof CaseCultivable){CaseCultivable c1 = (CaseCultivable) grilleCases[i][j+1]; c1.setDepartPropagation();}
}
```

Mais heureusement cette dernière peut être stopper grâce au pesticides.

vii. Simulateur Potager

La classe « SimulateurPotager » crée une grille 2D de Caseobjets , qui représentent différents types de terres (cultivables ou non, avec ou sans légume qui y poussent). La classe définit également plusieurs autres sous-classes : Saison, Legume, Case_action et plusieurs autres classes de simulation (SimulateurMeteo, SimulateurPesticide, SimulateurEngrais, SimulateurNuisible, et SimulateurBoutique).

La principale fonctionnalité du simulateur est de simuler la croissance des légumes dans le jardin, en fonction de différents facteurs tels que la saison, la météo, la présence de parasites ou d'engrais, et les actions entreprises par l'utilisateur (telles que la plantation, la récolte, ou l'application de pesticides ou d'engrais).

La classe a plusieurs variables d'instance qui gardent une trace de diverses choses, telles que la taille de la grille, le nombre de semences et de pesticides disponibles, la somme d'argent dont dispose l'utilisateur et la vitesse à laquelle la simulation s'exécute. Il définit également plusieurs méthodes pour effectuer différentes actions, telles que planter des graines, appliquer des pesticides ou des engrais, récolter des légumes et acheter des articles dans un magasin virtuel.

IV. Diagramme UML