université PARIS-SACLAY

LICENCE INFORMATIQUE

Programmation Concurrente et Interfaces Interactives

Flowercraft 

Enseignants:

Valentin FOUILLARD

Nicolas SABOURET

Groupe 4 :

Ngoc Anh NGUYEN

Xue YANG

Ariane ZHANG

Zhengtian DING

L3 INFO 2022 - 2023

Table des matières

I. INTRODUCTION

1. But du jeu

La thématique du jeu proposée est celle d'une jardinerie : le joueur contrôle des jardiniers qui plantent et récoltent des fleurs de différents types pour composer des bouquets. Les bouquets rapportent de l'argent qui permettent d'embaucher plus de jardiniers ou d'acheter de nouvelles graines pour faire plus de bouquets. Le problème est que des lapins se baladent dans le jardin en mangeant les fleurs. Le but du jeu est donc de gagner le plus d'argent en évitant de faire faillite.

2. Règle du jeu

Le joueur peut acheter des graines,faire des bouquets et gagner de l'argent grâce aux bouquet vendu dans le bâtiment. Perd quand le joueur n'a plus d'argent

Le jeu est composé de plusieurs unités avec chacune leurs actions et objectifs :

- Les jardiniers sont des unités contrôlable par le joueur par clic de souris. Ils peuvent planter et récolter des fleurs, se déplacer dans le jardin. Toutes ces actions prennent du temps. En plus de cela, les jardiniers peuvent acheter des graines dans la cabane du jardinier, vendre des bouquets et chasser les lapins. Ces actions sont instantanées. Un nouveau jardinier est créé si un jardinier paie une somme d'argent à la cabane du jardinier. Cette action prend un certain temps.

- Les lapins sont des unités au nombre fixé à l'avance non contrôlable par le joueur qui se déplace sur les fleurs les plus proches. Ils apparaissent au bout d'un certain temps dans le jardin(pas au début du jeu). Ils peuvent manger les fleurs qui ne sont pas déjà mangées par d'autres lapins. Cette action prend un certain temps.

- La cabane est un bâtiment situé au milieu du jardin. Cette cabane vend des graines à prix fixe et achète des bouquet de fleurs à prix fixe. De plus, il permet de produire des jardiniers à prix linéaire au nombre de jardiniers présents sur le jardin. Enfin, ce bâtiment permet de produire des bouquet de fleurs.

II. ANALYSE GLOBALE

Fonctionnalités

Nous décollons plusieurs fonctionnalités des règles du jeu décrit précédemment :

- Interface graphique avec un jardin avec un bâtiment au milieu et une partie de contrôle qui contient tous les boutons d'action.
- Création d'une grille en deux dimensions pour le jardin
- Implémenter le déplacement des unités.
- Implémenter l'unité lapin, afficher les lapins et prendre en compte toutes ses actions :
    1. apparaître dans le jardin(créé après quelque minutes,le nombre est augmenté tous les cinq minutes.Les lapins vont se déplacer mais aléatoirement)
    2. rester autour des fleurs(les autres lapins ne peuvent pas rester autour le même fleur s'il est occupé déjà)
    3. mangez le fleurs(détruire cette fleur et continuer à se déplacer vers une autre case)
- Implémenter l'unité jardinier, afficher le jardinier et prendre en compte toutes ses actions :
    1. planter des fleurs instantanément sur une zone du jardin libre
    2. récolter des fleurs instantanément
    3. acheter des graines dans la cabane du jardinier instantanément lorsque le jardinier est à l'emplacement de la cabane
    4. faire fuire les lapins instantanément
    5. vendre les bouquet dans la cabane du jardinier instantanément lorsque le jardinier est à l'emplacement de la cabane
    6. recruter des jardiniers dans la cabane du jardinier instantanément lorsque le jardinier est à l'emplacement de la cabane
- Implémenter l'unité bâtiment, afficher le bâtiment au milieu de jardin et prendre en compte toutes ses actions :
    1. calculer le prix des graines en fonction du type
    2. faire des bouquets en utilisant les fleurs de jardinier
    3. calculer le prix des bouquets en fonction du type de bouquet
- Implémenter l'unité bouquet
    1. représenter un bouquet de fleurs à 4 types différents: rose, lys, tulipe, mixtes.
- Implémenter l'unité fleur et afficher les fleurs dans le jardin.
    1. grandir au cours du temps
    2. mourir si un lapin le mange

De plus, nous devons intégrer toutes les actions relatives aux joueurs. Le jardinier apparaît sur la carte, il représente le joueur. Les actions du jardinier se font via le panneau de contrôle. Le joueur a un montant d'argent initial. Il peut effectuer toutes les actions sur le panneau de contrôle. Le joueur a besoin d'argent pour acheter les graines et il peut gagner l'argent grâce à la vente des fleurs, des bouquets. Le joueur perd quand il n'a plus d'argent.

III. PLAN DE DÉVELOPPEMENT

Liste des tâches nécessaires :

- Création de la fenêtre principale
- Création et affichage de la grille
- Création des unités et ses actions
- Implémentation la croissance du fleurs
- Déplacement du jardinier en clic
- Création des boutons des actions
- Création d'interface des unités et du contrôleur liant l'interface et l'état.
- Ajouter des images
- Déplacement avancé des unités en implémentant l'algorithme A*
- Rédaction du rapport
- Documentation du code

Organisation du travail :

- Ngoc Anh NGUYEN : s'occupe de la grille, le fleur dynamique, le contrôleur liant les interfaces des unités et l'état.

- Xue YANG : se concentre sur les fonctionnalités du bâtiment et l'interface du fleur et celle du bâtiment.

- Ariane ZHANG : implémente le jardinier et le lapin et leurs actions. Elle s'occupe aussi l'interface principale.

- Zhengtian DING : se charge du déplacement avancé du jardinier et des lapins en utilisant l'algorithme A*.

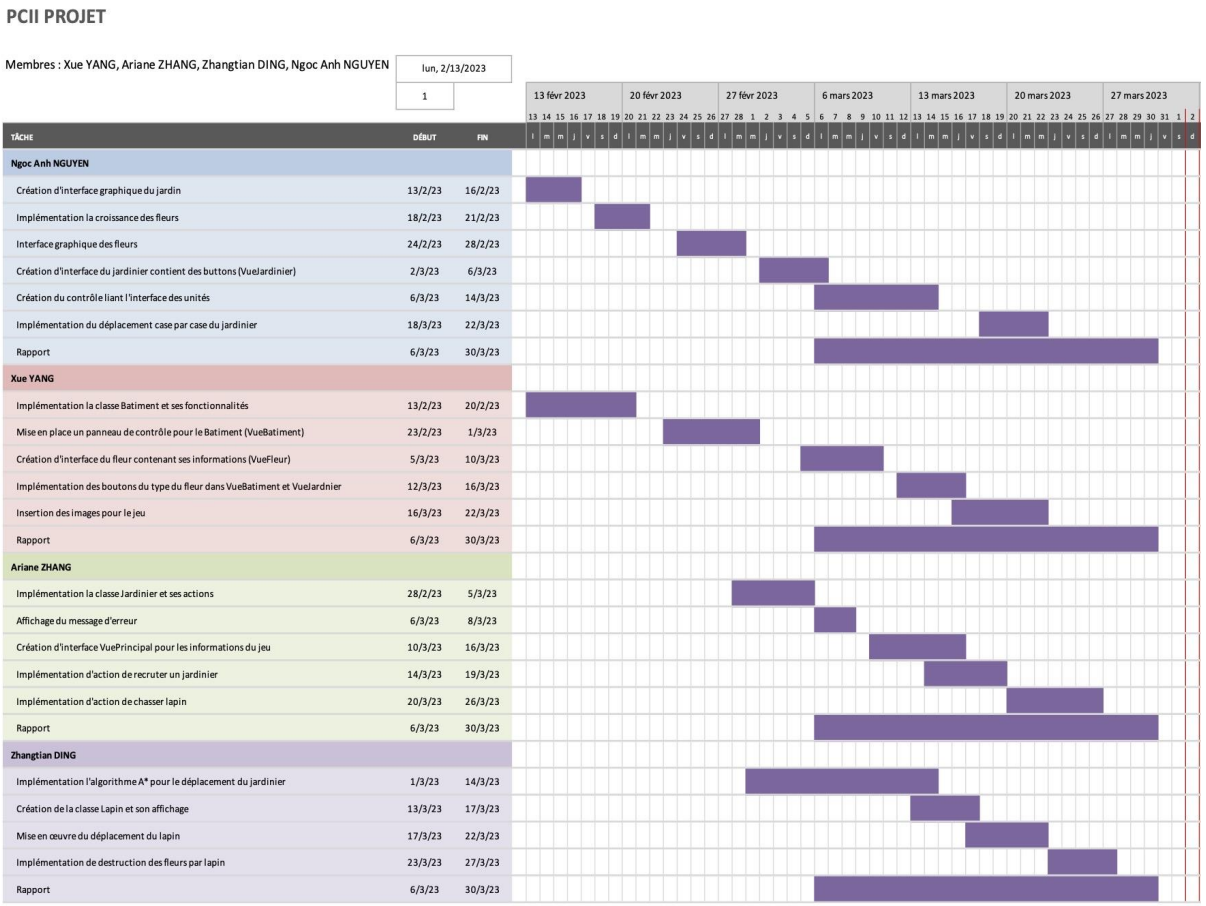


Figure 1 – Diagramme de Gantt de plan de développement

IV. CONCEPTION GÉNÉRALE

Nous avons adopté le motif MVC pour le développement de notre interface graphique. En effet, le code est divisé en trois parties dont le modèle, la vue et le contrôle. Le package modèle implémente toutes les données du jeu, c'est à dire les classe jardin, batiment, fleur, jardinier, lapin, bouquet, message et la classe abstrait unité. Le package control gère toutes les actions que peut effectuer le jardinier en appelant des méthodes implémentées dans le modèle. Et enfin, le package vue affiche toute l'interface du jeu. Elle est divisée en plusieurs parties, la vue du jardin, vue principal représentant les informations générales du jeu, vue de la fleur, vue du panneau de contrôle au bâtiment,la vue du panneau de contrôle du jardinier.

Cette séparation facilite la gestion du code pour les développeurs. Elles permettent aux développeurs de se concentrer sur ces tâches spécifiques sans interférer avec d'autres parties de l'application. Elle permet également de mieux se repérer. Par exemple, lorsqu'il y a un bug, on peut facilement localiser les erreurs et les corriger rapidemment.

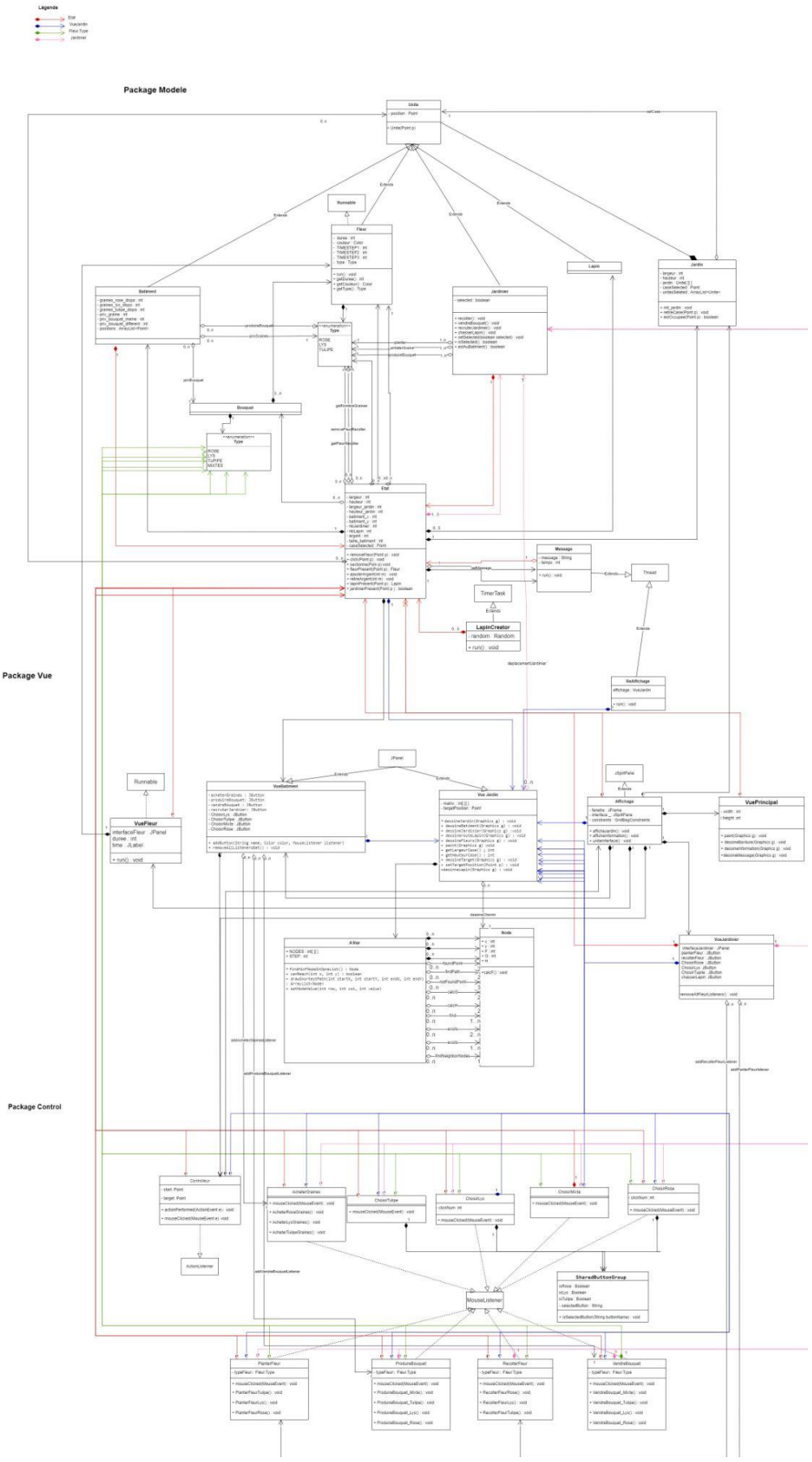


Figure 2 – Diagramme de classe

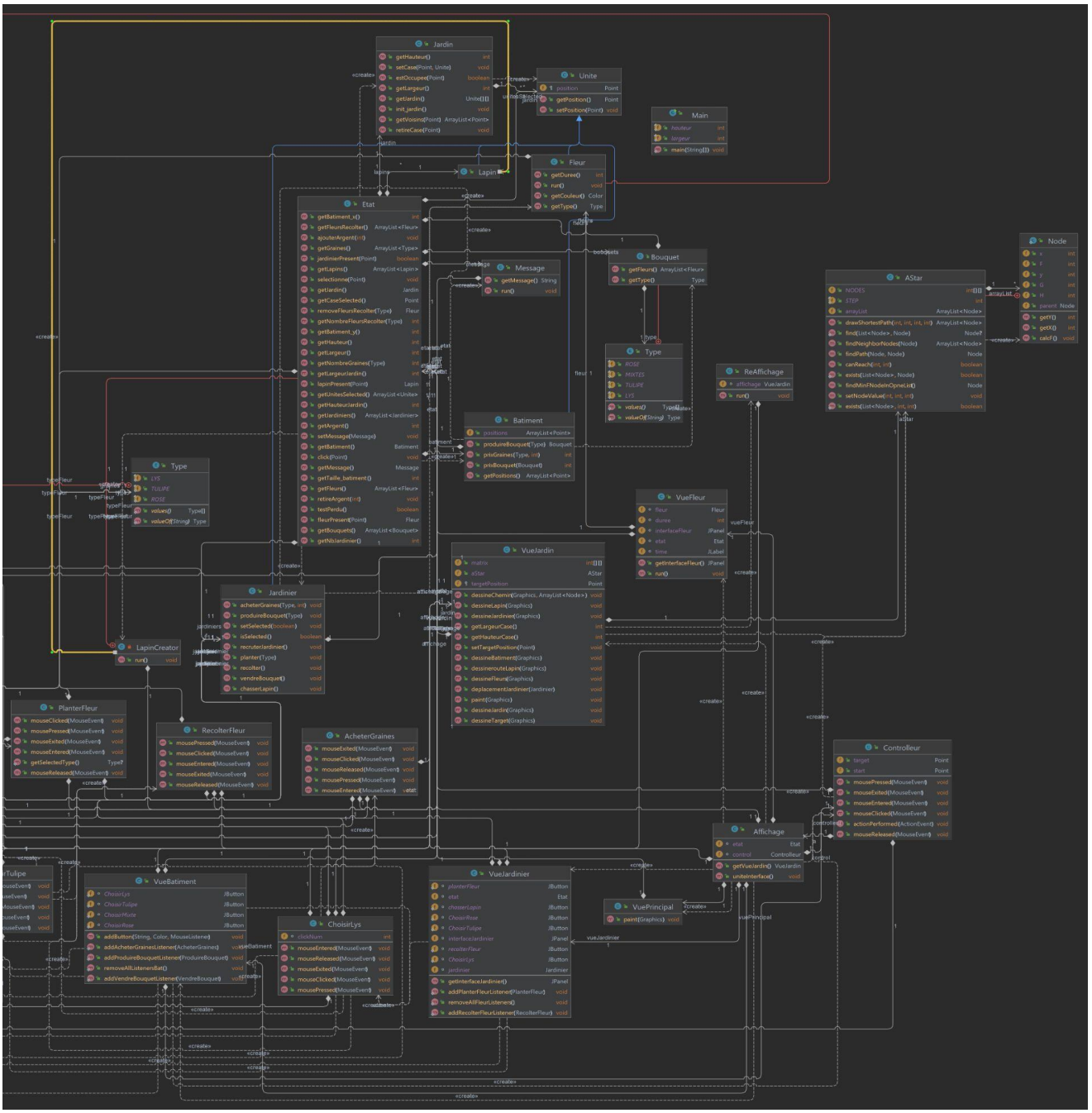


Figure 3 - Diagramme de classe généré par l'application

V. CONCEPTION DÉTAILLÉE

1. Création d'une grille pour le jardin

Le terrain est représenté par la classe <Jardin>. Elle contient deux attributs *largeur* et *longueur* qui sont la taille du tableau de deux dimensions. Le tableau initial possède la valeur null indiquant que la case est vide. Après la valeur des cases peuvent être remplacée par les unités (<Fleur>, <Batiment>, <Jardinier>, etc) à l'aide de la méthode "setCase" ou retirer une unité dans une case avec la méthode "retirerCase". La classe <Jardin> possède également un attribut *caseSelected* qui conserve en mémoire la dernière unité que le joueur a sélectionnée. Ainsi l'interface de cet unité reste affichée à l'écran (la partie à droite d'écran) et le joueur peut toujours lui transmettre des ordres.

2. Création des unités

La classe <Unite> représente les éléments dans le jardin <Jardin>. Elle contient un attribut pour la position de cette unité sur le jardin et les méthodes setter, getter.
L'objectif de cette classe est pour tous les élements (<Fleur>, <Jardinier>, <Batiment>, etc) qui sont sur le jardin ont la même type <Unite>.

<Fleur> est une unité dynamique, ça veut dire qu'elle évolue avec le temps par le biais d'un thread. Elle ont 3 types : ROSE, LYS, TULIPE et contient *duree* et *couleur* qui représentent son évolution.

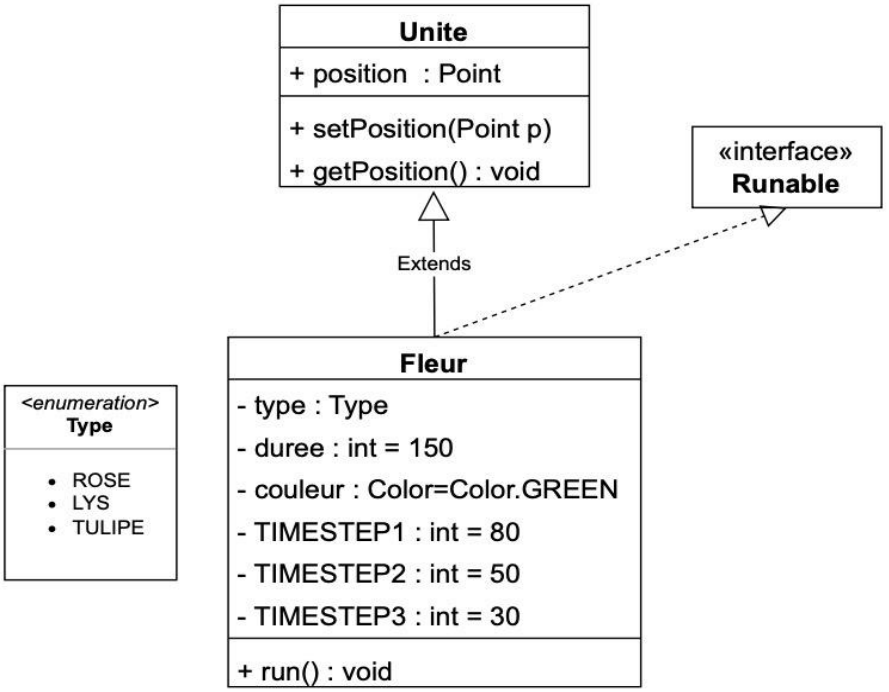


Figure 4 – Classe Fluer et Unité

La classe <Jardinier> est une unité contrôlable par le joueur par clic de souris. Il est est composé de deux attributs selected pour savoir si le jardinier est sélectionné par le joueur avec un click de souris. Et etat qui permettent de récupérer les informations générales du jeu, puisque le jeu peut avoir plusieurs jardiniers et l'information doit être commune à tous les jardiniers. Les informations générales sont les fleurs et lapins présentes dans le jardin, les fleurs récoltées, les graines en stock, les bouquets de fleurs, le solde actuel etc. Cette unité est capable de réaliser les actions suivantes : planter une fleur, récolter, acheter des graines, chasser des lapins, vendre des bouquets, produire des bouquets, recruter un jardinier.

<Batiment> gère les actions relatives aux bâtiments dans le jeu, notamment la vente de graines (prixGraines), la production et la vente de bouquets de fleurs (prixBouquet, produireBouquet) et le suivi de la position des bâtiments sur la carte (getPositions). Il enregistre le nombre de graines disponibles et fournit des méthodes pour vendre ces graines et acheter des bouquets de fleurs. Il permet également de produire des bouquets de fleurs et de les ajouter à une liste. La classe hérite de la classe Unite, qui enregistre la position de toutes les unités du jeu, y compris les bâtiments. Lorsque le joueur déplace le jardinier dans le bâtiment, il peut interagir avec celui-ci en utilisant les opérations fournies par le bâtiment. Par exemple, il/elle peut acheter des graines disponibles dans le bâtiment, produire et vendre des bouquets de fleurs. Le joueur peut exécuter ces opérations en appelant les méthodes fournies par le bâtiment.

3. Affichage de la grille

<VueJardin> : Cette classe permet cet l'affichage des interfaces d'unité et celui du jardin. Elle contient les fonctions "dessineJardin" pour la grille, "dessineBatiment" ,"dessinieJardinier", "dessineFleurs" et "dessineLapin" pour toutes les unités.

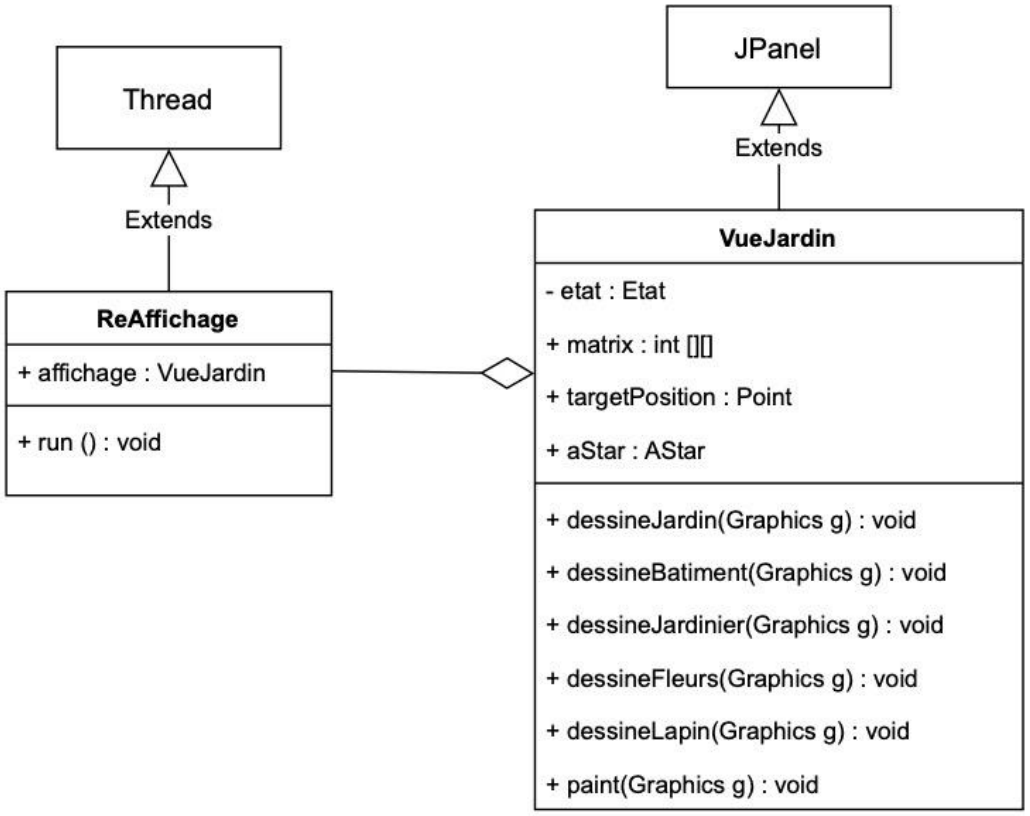


Figure 5 – Classe VueJardin et ReAffichage

On utilisera pour cela la fonction "paint", qui sera appelé naturellement par Swing, qui appellera toutes les fonctions ci-dessus qui affichent respectivement le <Jardin> et les <Unite>.

<ReAffichage> s'occupe de la mise à jour via un thread appelant régulièrement la méthode de mise à jour de <VueJardinier>, permettant ainsi l'affichage d'interfaces des unités dynamiques et changeantes. Par exemple: l'évaluation des fleurs.

4. Affichage des interfaces des unités

<VueJardinier> contient des boutons représentant les actions que le jardinier peut effectuer :

- Planter un fleur
- Récolter un fleur
- Chasser un lapin

<VueFleur> affiche les informations telles que les types de fleur et le temps restant avant la récolte sur la case qu'on a sélectionnée.

<VueBatiment> contient des boutons représentant les actions que le bâtiment peut effectuer :

- AcheterGraines
- ProduireBouquet
- VendreBouquet
- RecruterJardinier

Il y a aussi trois autres boutons pour choisir le type de fleur ou celui de bouquet.

<VuePrincipal> est pour les informations générales et aussi les messages pour le joueur

- Argent
- Nombre des graines de chaque type du fleur
- Nombre des fleurs que le joueur a eu
- Nombre des bouquets que le joueur a eu

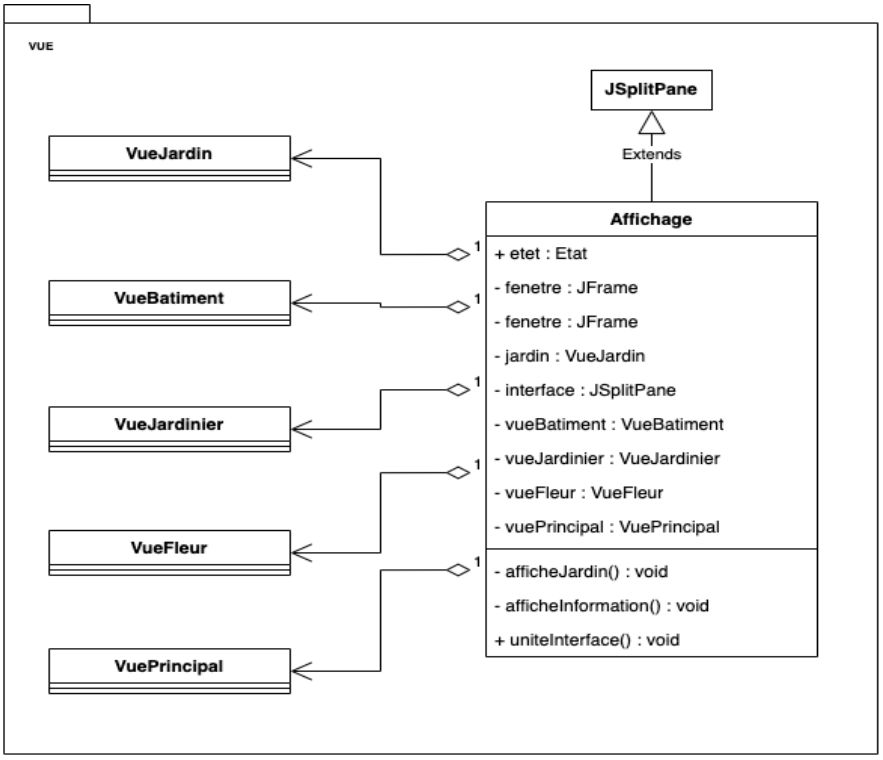


Figure 6 – Classe Affichage

<Affichage> s'occupe d'afficher la fenêtre du jeu. La classe va aussi changer l'interface de la partie contrôle (à droite de la fenêtre) correspondante à l'unité qu'on a sélectionnée grâce à les fonctions dans la classe <Etat>.

5. Planter des fleurs

Cette fonctionnalité permet de planter une fleur à la position du jardinier instantanément. Si la position actuelle ne présente aucune fleur, ni sur l'emplacement du bâtiment et qu'on a des stock de graines du type de graine demandé.
La fonctionnalité est implémentée dans la classe <Jardinier> et prend en paramètre un type de fleur "Fleur.Type" (un type énuméré de la classe <Fleur>). Elle utilise la méthode "fleurPresent(Point p)" pour tester si une fleur est déjà présente à une position donnée et "getGraines()" afin savoir si le joueur possède des graines du type demandé. Ces méthodes appartiennent à la classe <Etat>. Il y a également une méthode "getPositions()" dans bâtiment qui permet d'obtenir la liste des case du jardin occupé par le bâtiment. Si un test ne passe pas alors on ajoute un message d'erreur en utilisant la méthode "setMessage" de la class <Etat>. Au contraire, si tous les tests sont validés, le jardinier ajoute une fleur avec à la liste des fleurs présent dans le jardin et retire une graine de la liste de graine grâce à "getFleurs()" et "getGraines()" de la class <Etat>.

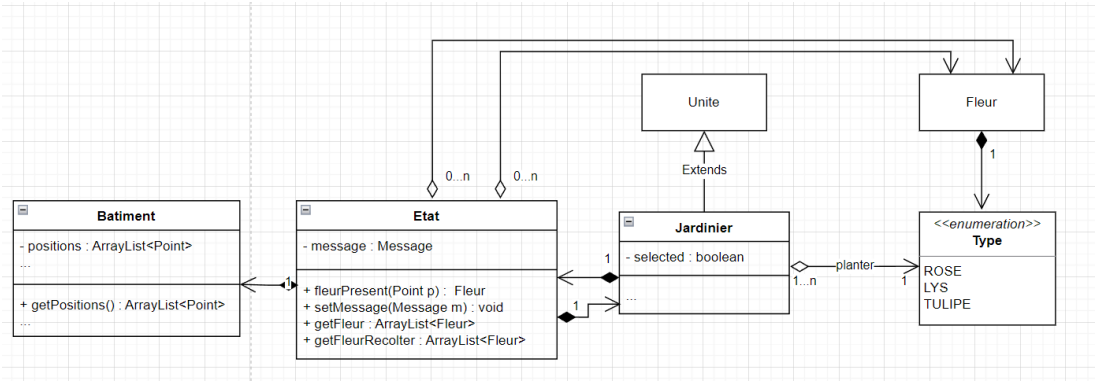


Figure 7 – Des classes concernant pour planter des fleurs

6. Récolter des fleurs

Cette fonctionnalité implémentée dans la classe <Jardinier> permet de récolter une fleur présente à la position du jardinier instantanément. On teste de la même manière que la fonctionnalité "Planter", si on est pas au bâtiment et qu'il a une fleurs la position actuelle. Si une fleur est présente, on utilise "getDuree()" de la classe <Fleur> pour tester si la fleur est prête à être récolter. Si un test ne passe pas alors on ajoute un message d'erreur en utilisant la méthode "setMessage" de <Etat>. A l'inverse, le jardinier retire la fleur de la liste "fleurs" et ajoute la fleur à la liste "fleursRecolter" grâce à "getFleurs()" et "getFleursRecolter()". On met a jour aussi le jardin en retirant cette unité fleur avec "retireCase(position)" de la classe Jardin>.

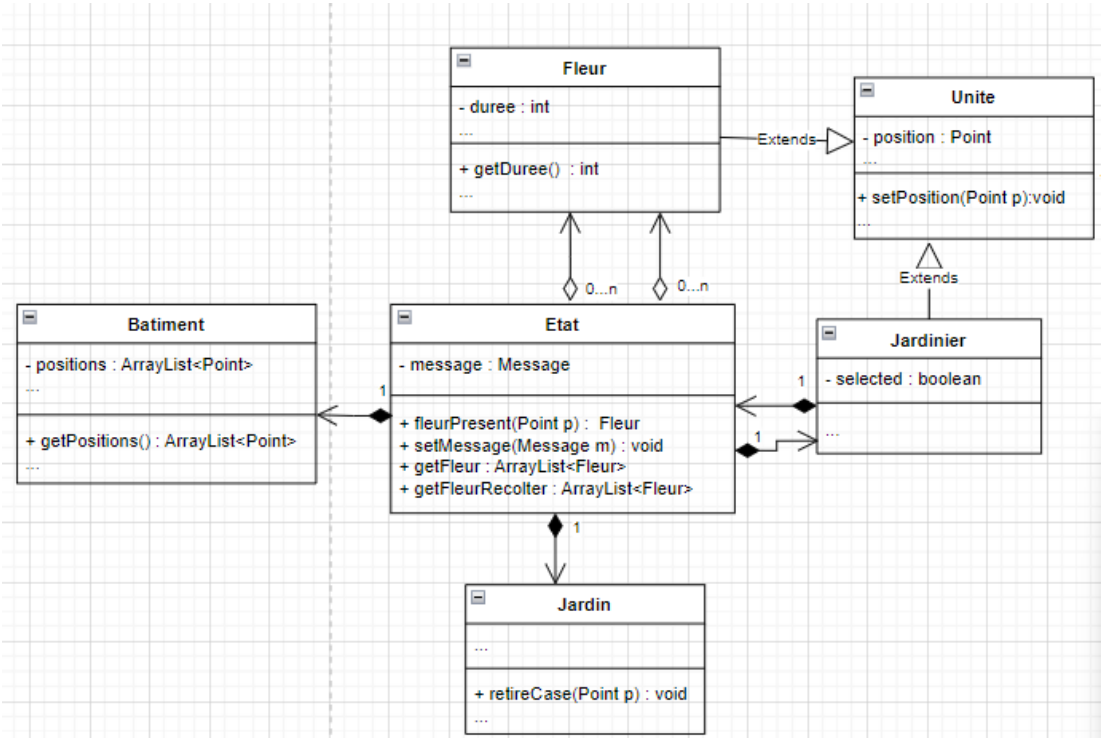


Figure 8 – Des classes concernant pour récolter des fleurs

7. Acheter des graines

Fonctionnalité implémenté dans la classe <Jardinier>. Elle prend en paramètre un type de graine Fleur “Fleur.Type” et la quantité de graines à acheter. On vérifie que le jardinier est bien à l'emplacement du bâtiment. Après vérification, le jardinier fait appel à la méthode “prixGraine(type, n)” de la classe <Batiment> pour calculer le prix. On vérifie que le joueur ait assez d’argent avec “getArgent()”. Et enfin on ajoute la graine à la liste des graines et paie avec la fonction “retireArgent()” de la classe <Etat>.

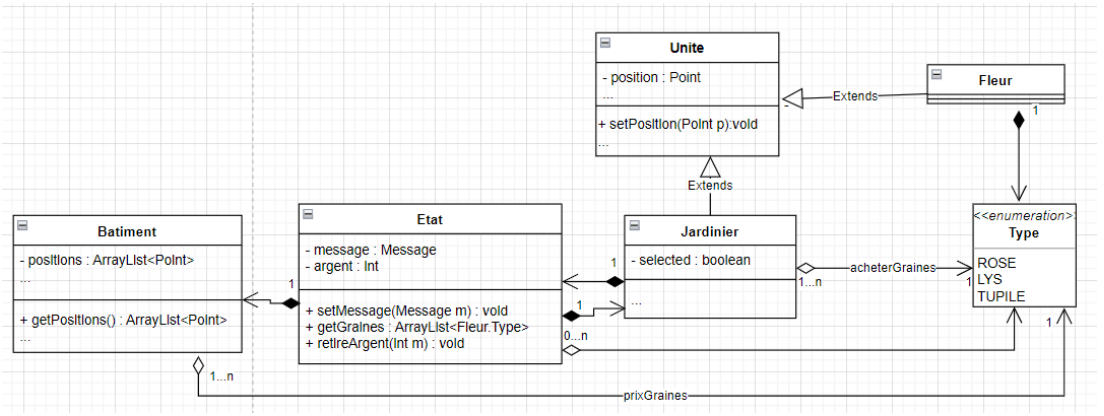


Figure 9 – Des classes concernant pour acheter des graines

8. Chasser les lapin

Cette fonctionnalité est implémentée dans la classe <Jardiner> et fait un appel à la méthode “lapinPresent()” afin de vérifier si un lapin est à la position du jardinier. De la même manière que les fonctionnalités précédentes, on retire le lapin avec “getLapins()”, “retireCase()” et affiche un message avec “setMessage()” lorsque le test est validé. Sinon un message d’erreur.

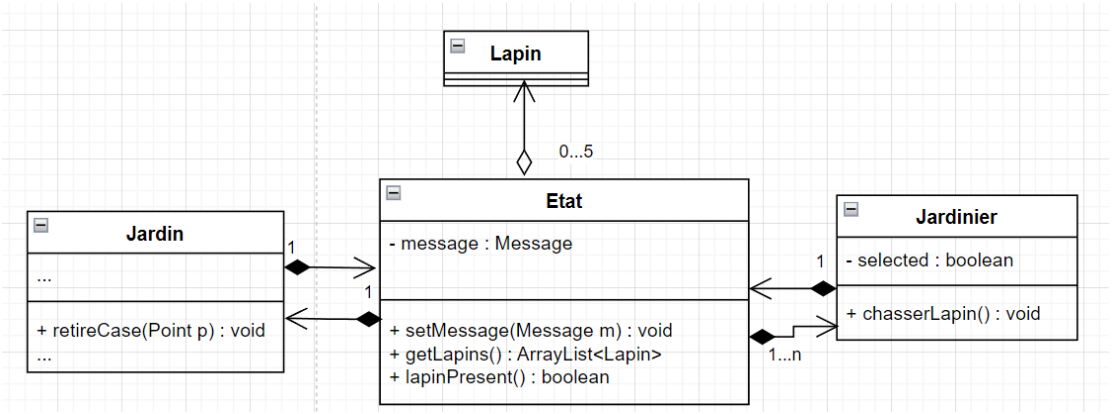


Figure 10 – Des classes concernant pour chasser les lapins

9. Vendre des bouquet

Cette fonctionnalité est implémentée dans la class <Jardiner> et vérifie les conditions de l’emplacement du jardinier et la quantité de bouquet en stock avec “getBouquet()”. Si le joueur a un bouquet en stock, on fait appel à la fonction “prixBouquet()” de la classe <Batiment>. Ensuite, le jardinier enlève le bouquet de la liste en stock et reçoit l’argent avec la fonction “ajouterArgent(prix)” de la classe <Etat>.

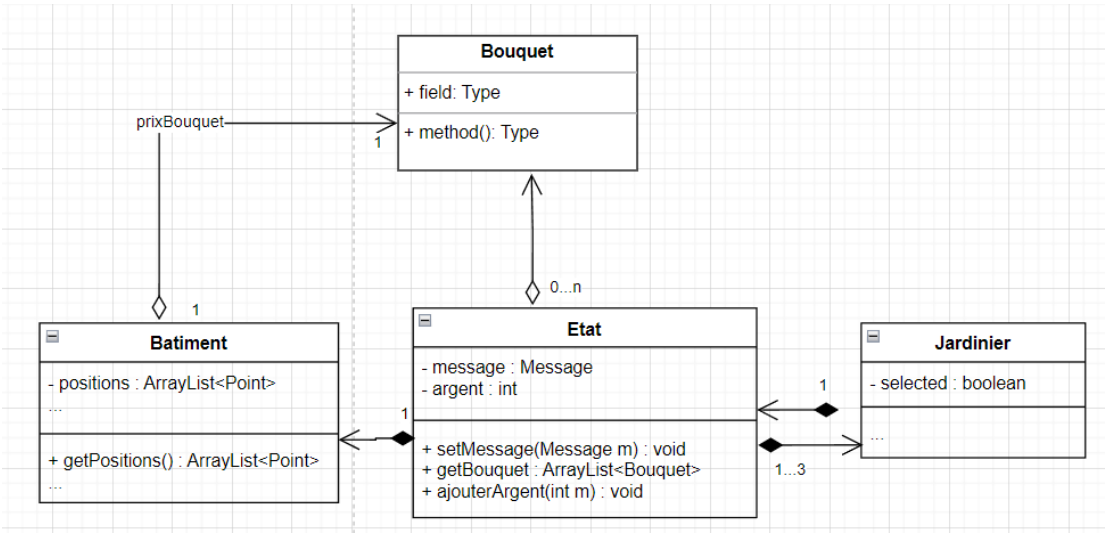


Figure 11 – Des classes concernant pour vendre des bouquets

10. Produire des bouquet

Cette fonction prend en paramètre un type de fleur “Fleur.Type” et vérifie que le jardinier est au batiment.Et elle fait un appel a la méthode “produireBouquet()” de la classe <Batiment> ou affiche un message d’erreur avec “setMessage” selon ces condition.

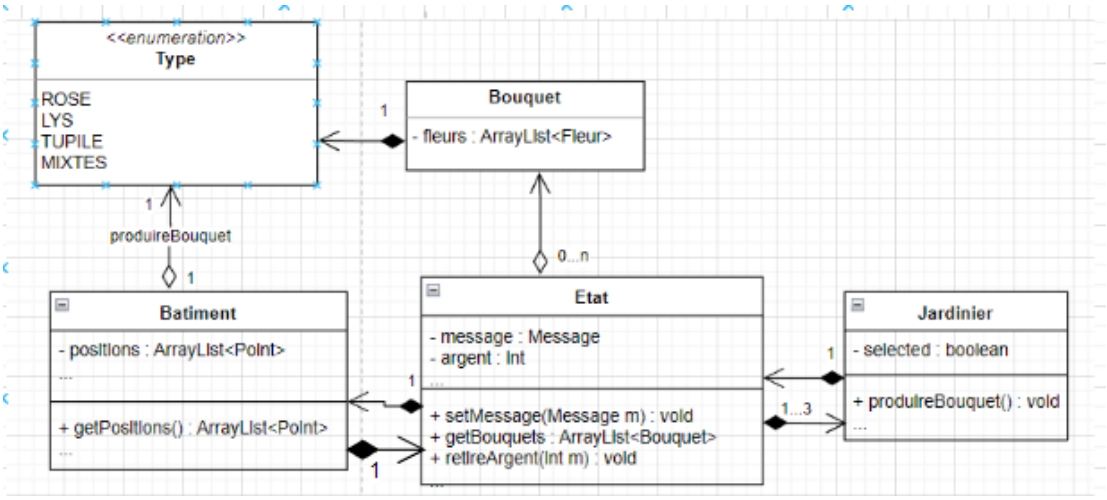


Figure 12 – Des classes concernant pour produire des bouquets

11. Recruter des jardinier

Le jardinier peut recruter deux autres jardiniers maximum (attribut “nbJardinier” fixé dans <Etat>) lorsqu'il arrive au bâtiment avec une somme d’argent d’au moins 200. Il vérifie ces conditions grâce au fonction “getJardinier()”, “getArgent()” de la class <Etat>. Si toutes les conditions sont réunies, un jardinier sera positionné aléatoirement dans le jardin.

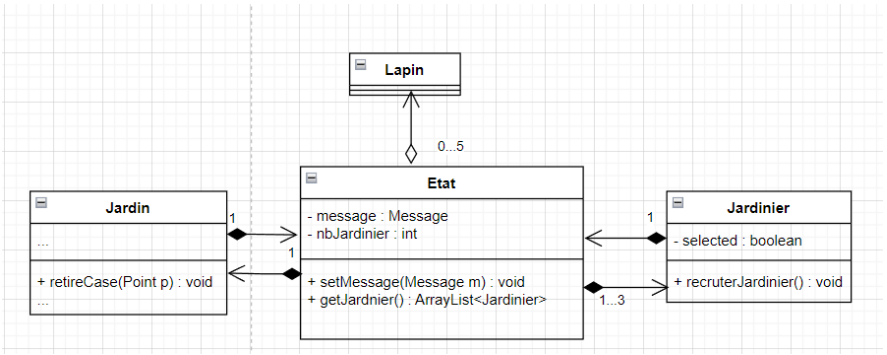


Figure 13 – Des classes concernant pour recruter des jardiniers

12. Fonctions de clic pour les boutons du panneau de contrôle

Les panneaux VueBatiment et VueJardinier contiennent des boutons pour sélectionner le type de fleur, de graine ou de bouquet à produire ou à vendre. Cette fonctionnalité de clic est mise en œuvre à l'aide des classes ChoisirRose, ChoisirLys, ChoisirTulipe et ChoisirMixte, et utilise des événements de souris. Lorsque l'utilisateur clique sur l'un des boutons pour sélectionner un type, il peut effectuer des opérations spécifiques pour ce type, telles que l'achat de graines ou la production de bouquets. Chaque bouton a sa propre classe de contrôle, comme illustré ci-dessous:

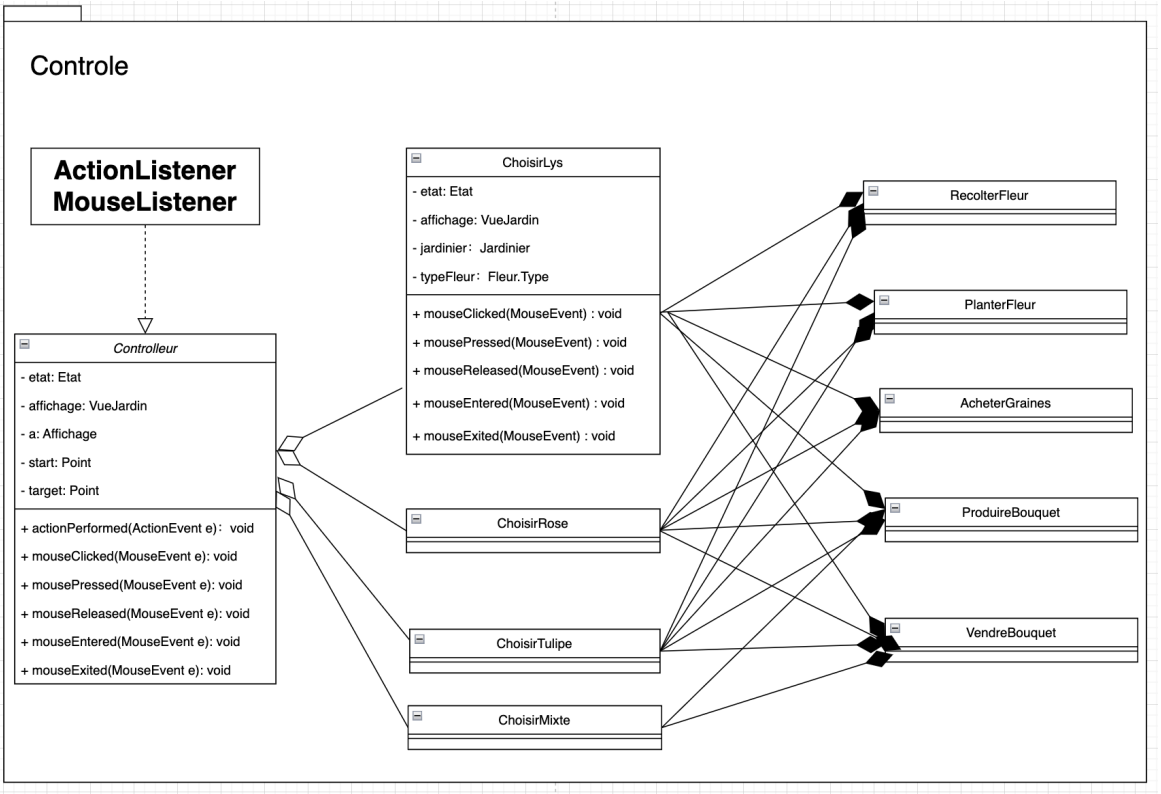


Figure 14 – Classe Controller

Après chaque sélection de type en cliquant sur le bouton correspondant, les boutons d'opération correspondants sont ajoutés à cette classe en appelant les méthodes addPlanterFleurListener() addProduireBouquetListener(), etc. Les boutons précédemment sélectionnés peuvent être supprimés en appelant les méthodes removeAllListenersBat() et removeAllFleurListeners().

13. Algo A*

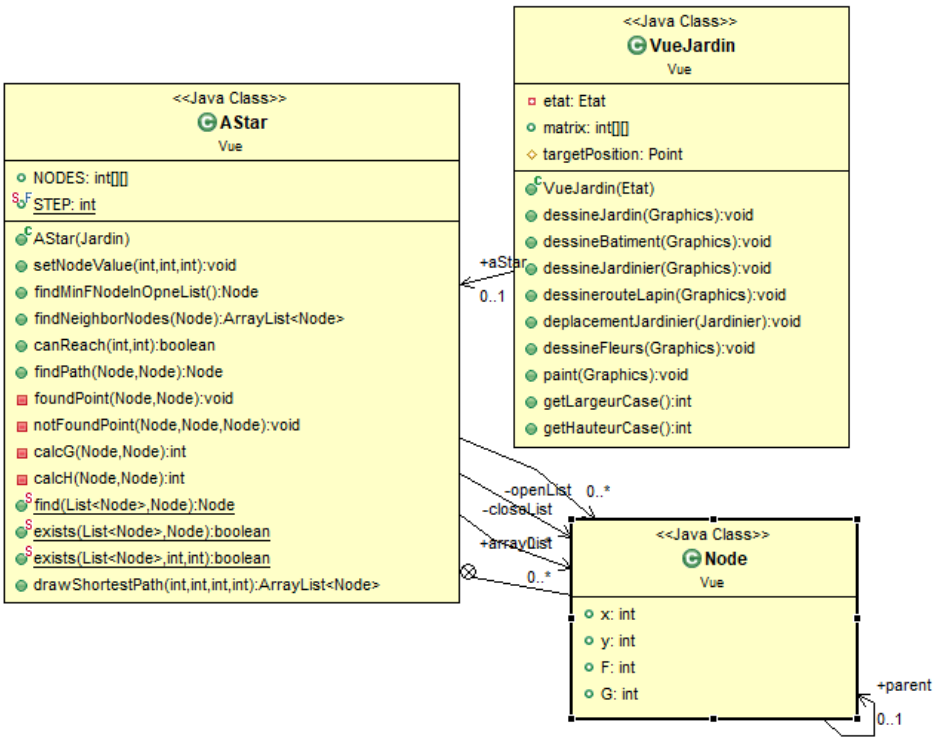


Figure 15 – Algorithme A star

L'algorithme A* est très utile pour les applications de navigation, comme les GPS, les jeux vidéo ou encore les robots autonomes. Il est capable de trouver rapidement le chemin le plus court entre deux points dans un environnement complexe, même en présence d'obstacles ou de contraintes de mouvement.

Dans l'implémentation de l'algorithme A* dans cette classe, chaque nœud du graphe est représenté par une instance de la classe Node, qui contient les informations suivantes :

1. Les coordonnées x et y du nœud dans la grille
2. Le coût du chemin parcouru depuis le nœud de départ jusqu'à ce nœud (G)
3. Une estimation du coût restant pour atteindre le nœud de destination depuis ce nœud (H)
4. Un pointeur vers le nœud parent, c'est-à-dire le nœud qui a conduit à ce nœud dans le chemin le plus court trouvé jusqu'à présent

L'implémentation de l'algorithme A* utilise une structure de données appelée "tas binaire" pour stocker les nœuds de la liste ouverte. Un tas binaire est un arbre binaire où chaque nœud a une valeur supérieure ou égale à ses enfants, ce qui permet d'extraire rapidement le nœud avec la valeur minimale.

La méthode drawShortestPath utilise la liste des nœuds parents pour retracer le chemin le plus court depuis la destination jusqu'au départ. Elle commence par suivre le pointeur vers le

nœud parent de la destination, puis le pointeur du nœud parent de ce nœud, et ainsi de suite, jusqu'à atteindre le nœud de départ. Cette méthode dessine ensuite ce chemin sur la grille en utilisant une couleur spécifique pour le rendre visible.

En somme, l'algorithme A* est un algorithme de recherche de chemin efficace et populaire, qui permet de trouver rapidement le chemin le plus court entre deux points dans un environnement complexe. Sa mise en œuvre nécessite une structure de données adaptée et une estimation appropriée du coût restant pour atteindre la destination.

14. Réalisation du déplacement du jardinier

Nous utilisons l'algorithme A* pour implémenter le déplacement du jardinier. Le jardin est représenté sous forme d'une grille en deux dimensions, et lorsque l'utilisateur clique sur un emplacement, cette information est transmise à une instance de la classe AStar. L'algorithme A* est ensuite utilisé pour générer le chemin le plus court vers la position cliquée. Une fois que le chemin est généré, la position du jardinier est mise à jour progressivement sur la carte en suivant le chemin calculé.

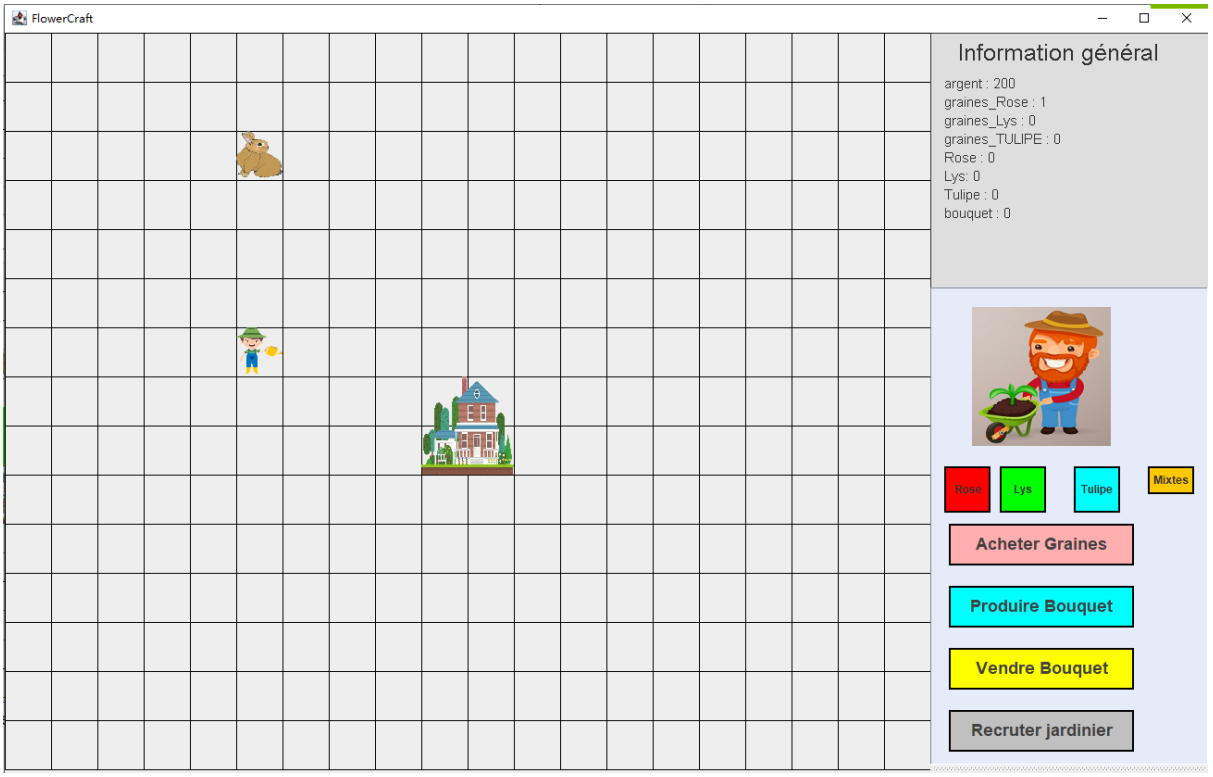
Le déplacement des lapins est également basé sur l'algorithme A*. Contrairement au jardinier, les lapins sont générés automatiquement sur la carte toutes les quinze secondes et jusqu'à cinq lapins peuvent être présents simultanément, comme le montre le diagramme UML précédent.

15. Fonction de lapin détruit les fleurs

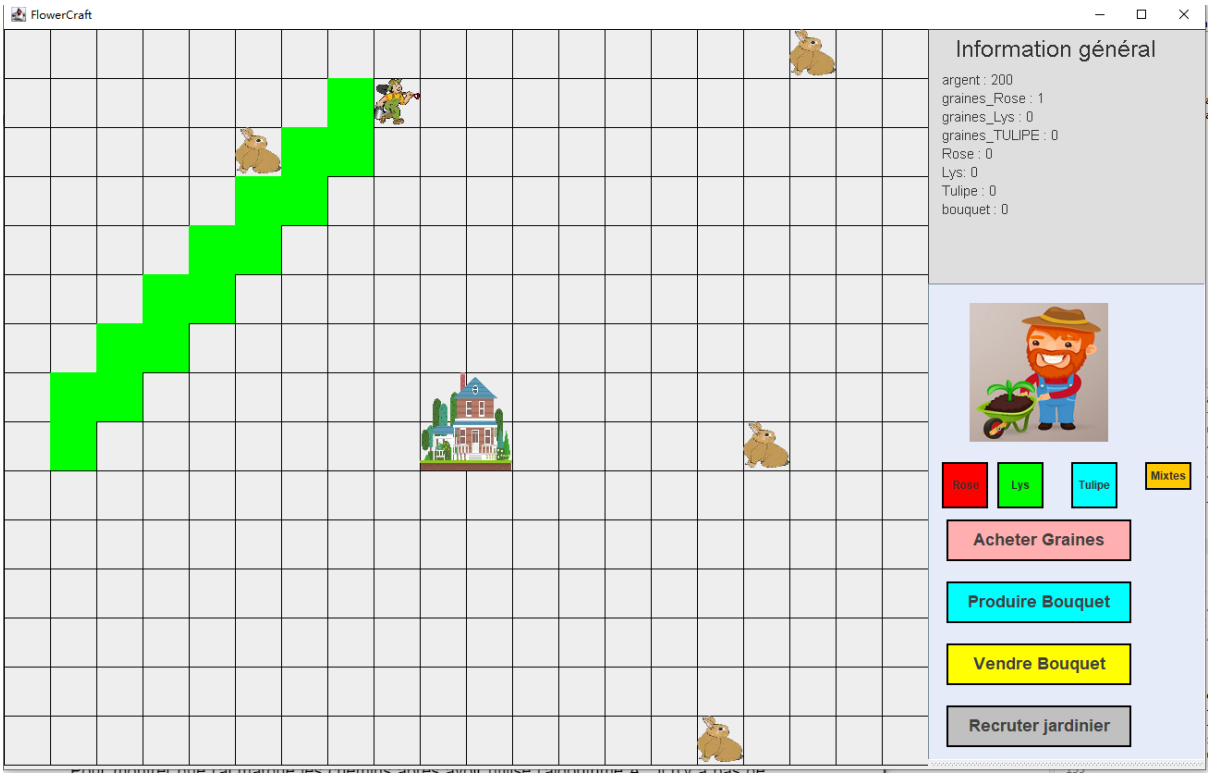
La fonction de déplacement du lapin est utilisée dans la fonction "dessinerouteLapin()" de la classe "VueJardin". Chaque fois qu'un nouveau lapin est généré, il cherche la fleur la plus proche sur la carte et utilise l'algorithme A* pour s'y déplacer. Si aucun jardinier ne chasse le lapin dans les 10 secondes, il détruira la fleur. Lorsque tous les lapins sont à côté de toutes les fleurs sur la carte, ils ne bougeront plus.

VI. RÉSULTAT

1. Début du jeu

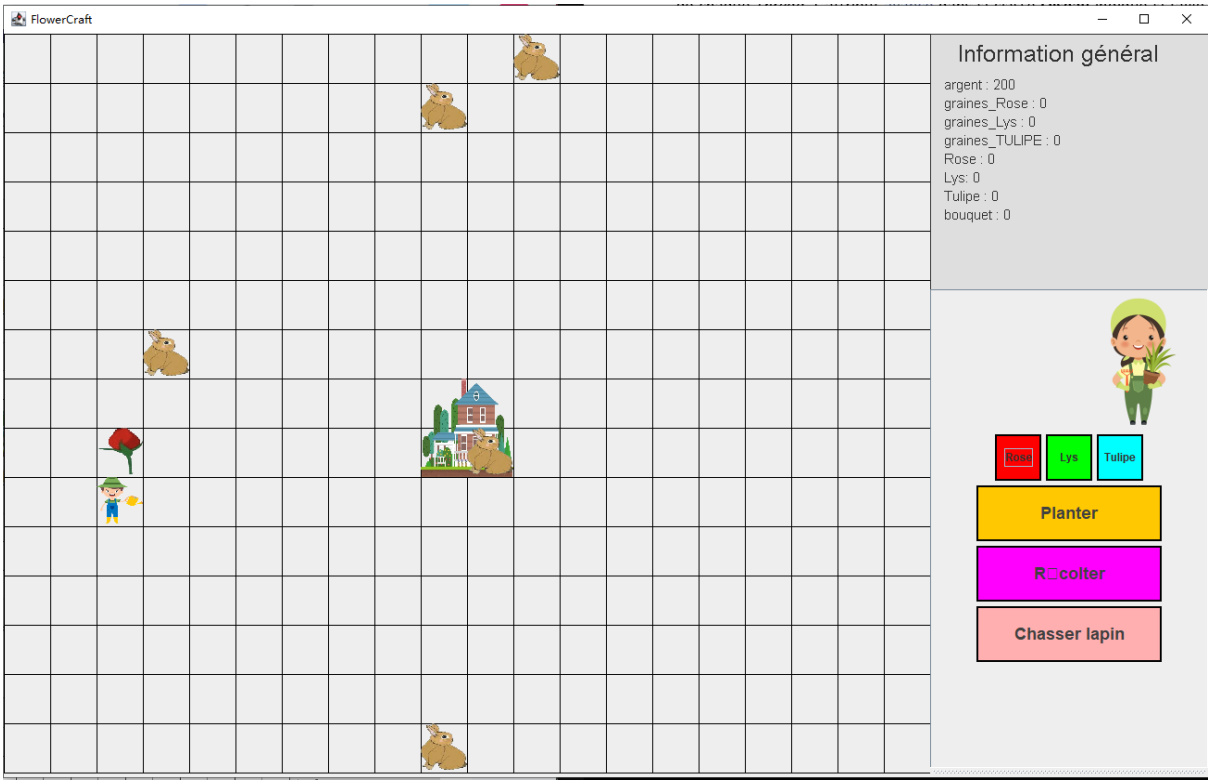


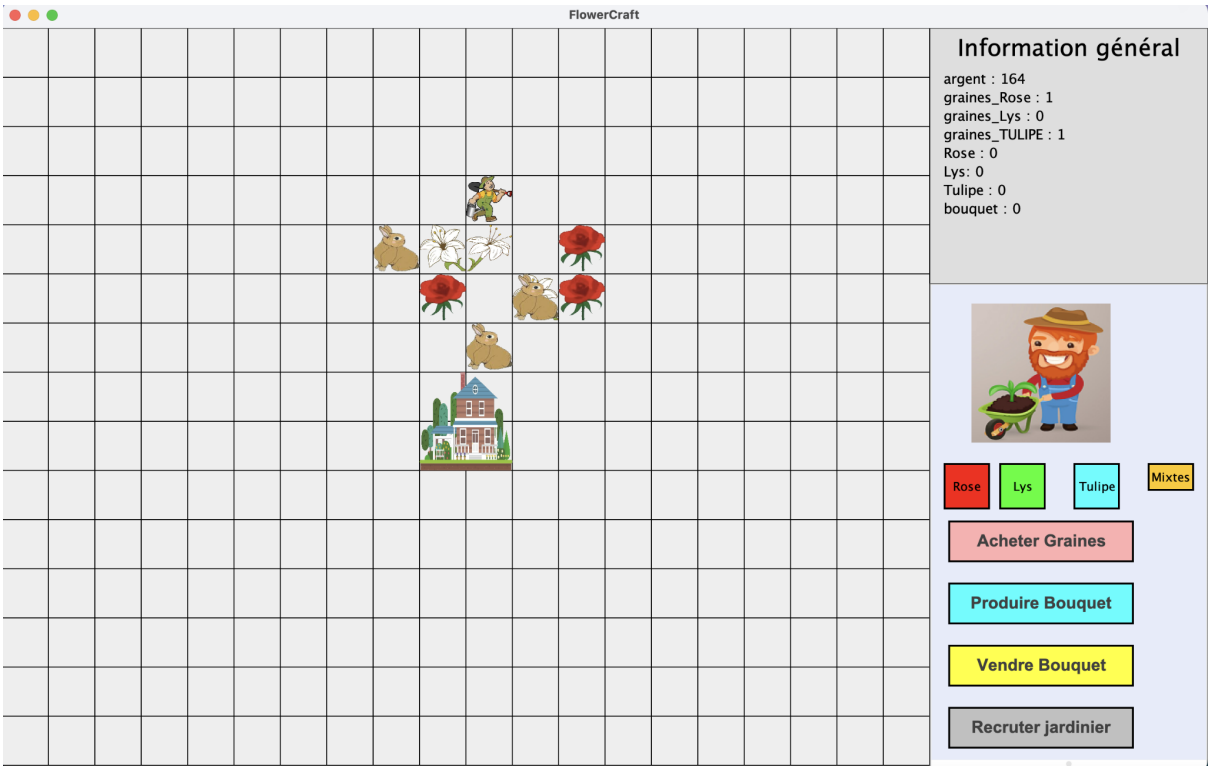
2. Déplacement du jardinier



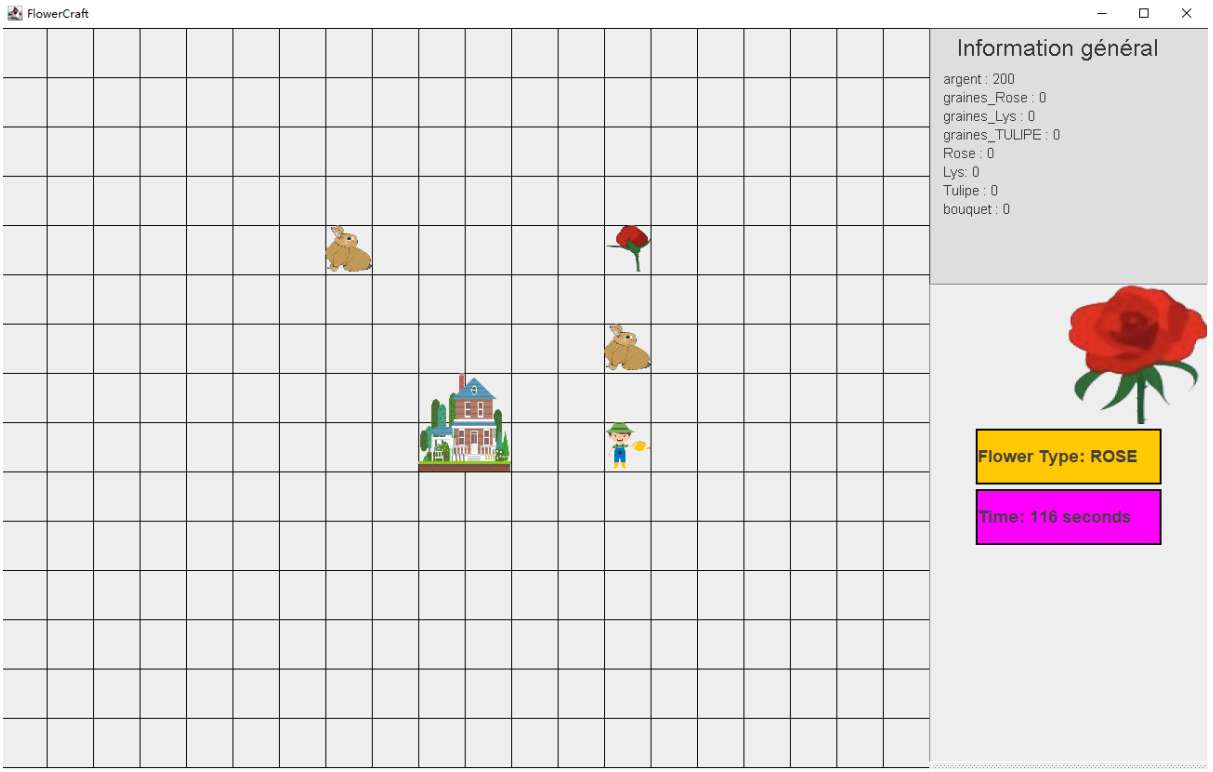
Pour montrer que j'ai marqué les chemins après avoir utilisé l'algorithme A*, il n'y a pas de chemin vert dans le jeu final.

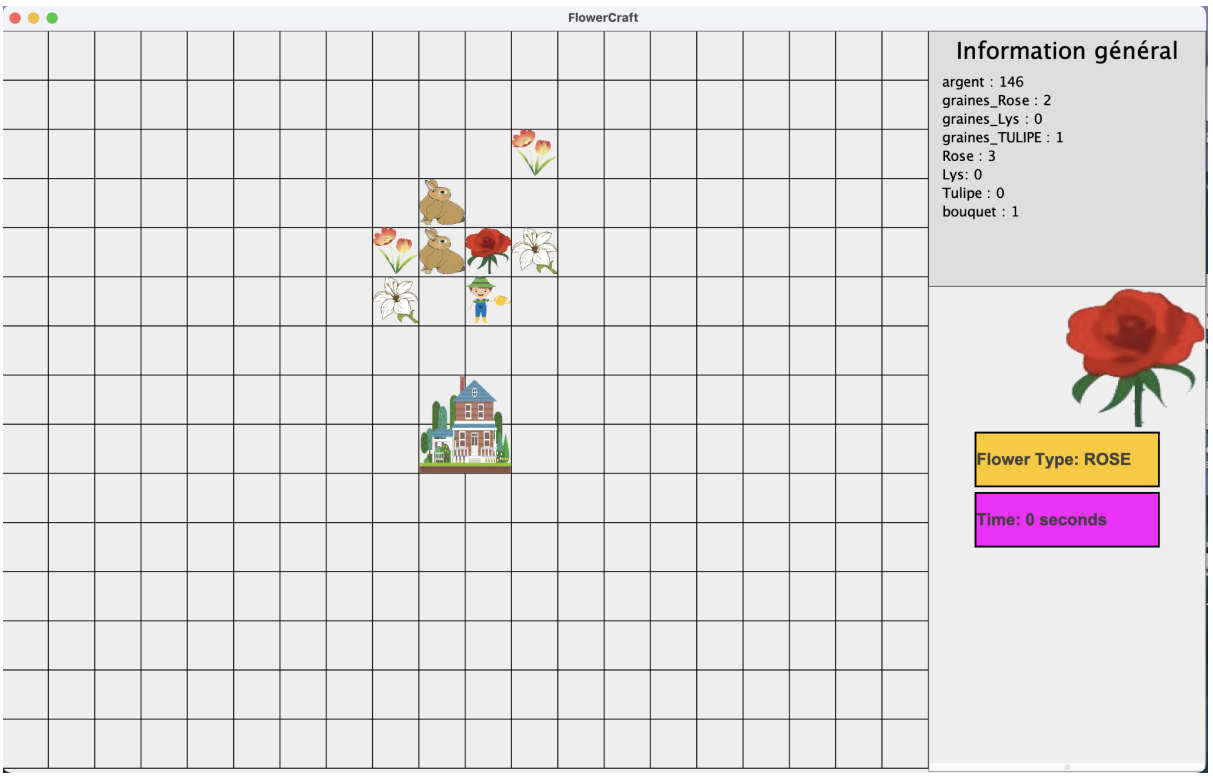
3. Planter


FlowerCraft
Information général
argent : 200
graines_Rose : 0
graines_Lys : 0
graines_TULIPE : 0
Rose : 0
Lys: 0
Tulipe : 0
bouquet : 0
Rose
Lys
Tulipe
Planter
Récolter
Chasser lapin

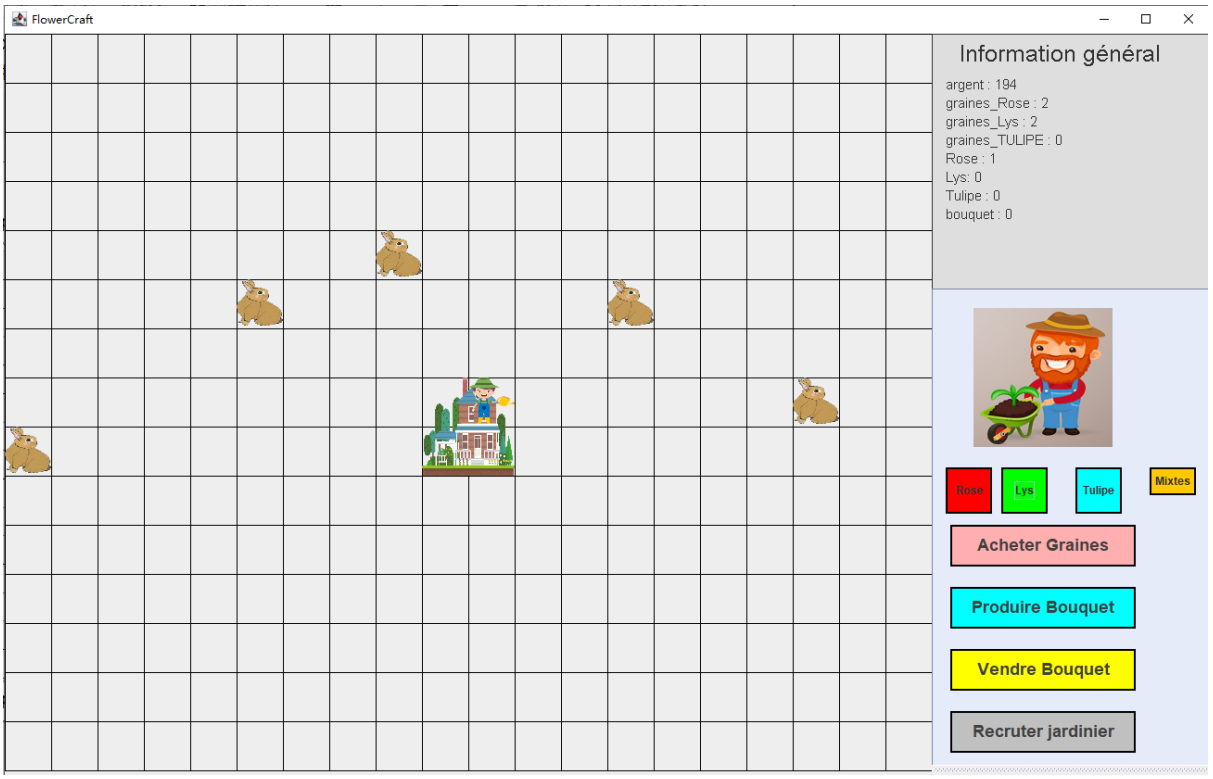

FlowerCraft
Information général
argent : 164
graines_Rose : 1
graines_Lys : 0
graines_TULIPE : 1
Rose : 0
Lys: 0
Tulipe : 0
bouquet : 0
Rose
Lys
Tulipe
Mixtes
Acheter Graines
Produire Bouquet
Vendre Bouquet
Recruter jardinier

4. Vérifier le fleur

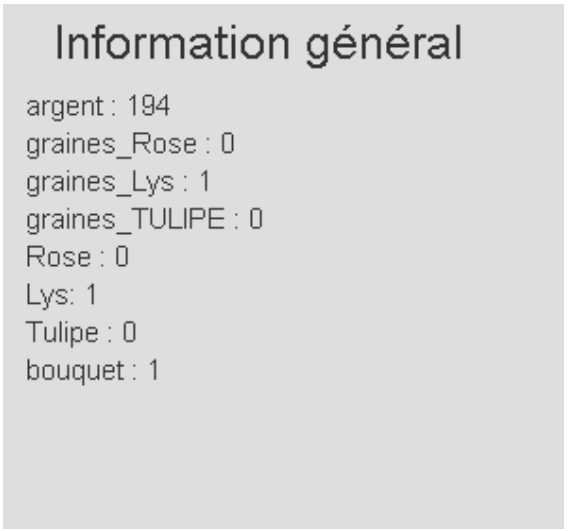


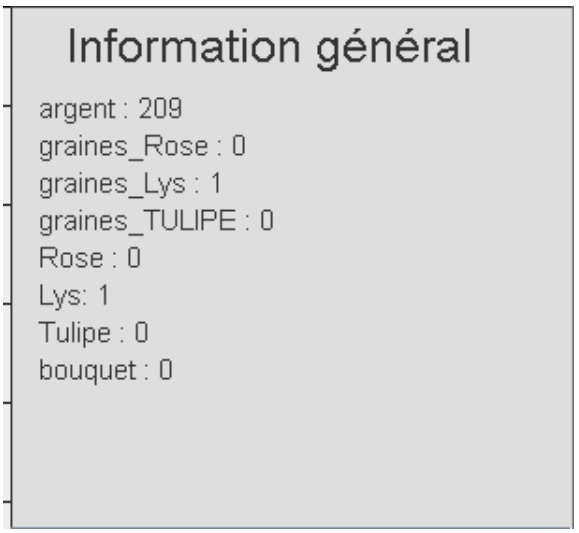


5. Après avoir acheter des graines et récolter



6.après produire le bouquet et le vendre





VII. DOCUMENTATION UTILISATEUR

- Prérequis : JAVA avec un IDE
- Mode d'emploi : Importez le projet dans votre IDE, sélectionnez la classe Main à la racine du projet puis《Run as Java Application》.
- Instructions du jeu : Cliquer sur l'icône du jardinier signifie que le jardinier est sélectionné. Cliquez sur la grille pour que le jardinier se déplace à la position spécifiée. Sur le côté droit se trouvent le solde du joueur, le nombre de graines, etc. Les joueurs peuvent cliquer sur les boutons pour effectuer une série d'actions.

VIII. DOCUMENTATION DÉVELOPPEUR

Classe Main
Dans cette classe, on peut trouver la méthode《main》en dehors des packages.

Les packages

***Control*:** Ce package comprend toutes les classes qui contrôlent le jeu. Par exemple, l'implémentation des différents boutons et les méthodes de la classe Controlleur pour les clics de souris indiquant au jardinier de se déplacer.

***Modele et Modele.Unite*:** Ces deux packages contiennent tous les éléments du jeu, tels que le jardin, les fleurs, le jardinier, etc. Vous pouvez modifier les méthodes de ce composant dans l'une ou l'autre classe, ou changer les propriétés ou la position initiale du composant dans la classe Etat, etc.

***Vue*:** Ce package contient les classes qui sont responsables de l'affichage graphique du jardin et des unités qui y sont placées. Les classes dans ce package implémentent des interfaces graphiques utilisateur pour permettre à l'utilisateur d'interagir avec le jeu. Il contient également la classe AStar, un algorithme de recherche de chemin qui implémente le déplacement du jardinier et du lapin, et vous pouvez toujours utiliser cette classe pour faire bouger n'importe quel objet dans le jardin.

IX. CONCLUSION ET PERSPECTIVES

En conclusion, ce projet nous a permis de mettre en pratique nos connaissances en Java et en développement d'applications orientées objet. Nous avons conçu et implémenté un jeu de simulation de jardinage qui permet aux utilisateurs de planifier, de cultiver et de récolter leurs fleurs dans un environnement virtuel.

Notre projet est un succès grâce à plusieurs aspects clés. Tout d'abord, nous avons acquis une maîtrise plus approfondie de Java, en particulier en ce qui concerne l'utilisation de Graphics2D pour implémenter le jeu. De plus, nous avons appris à implémenter l'algorithme A* pour trouver le chemin le plus court entre deux points dans un graphe pondéré.

En outre, nous avons beaucoup bénéficié de l'utilisation de GitLab pour la collaboration en groupe, ce qui a grandement amélioré notre efficacité. Nous avons également réussi à mettre en œuvre avec succès les règles du jeu que nous avons établies initialement, et nous sommes très satisfaits du résultat final.

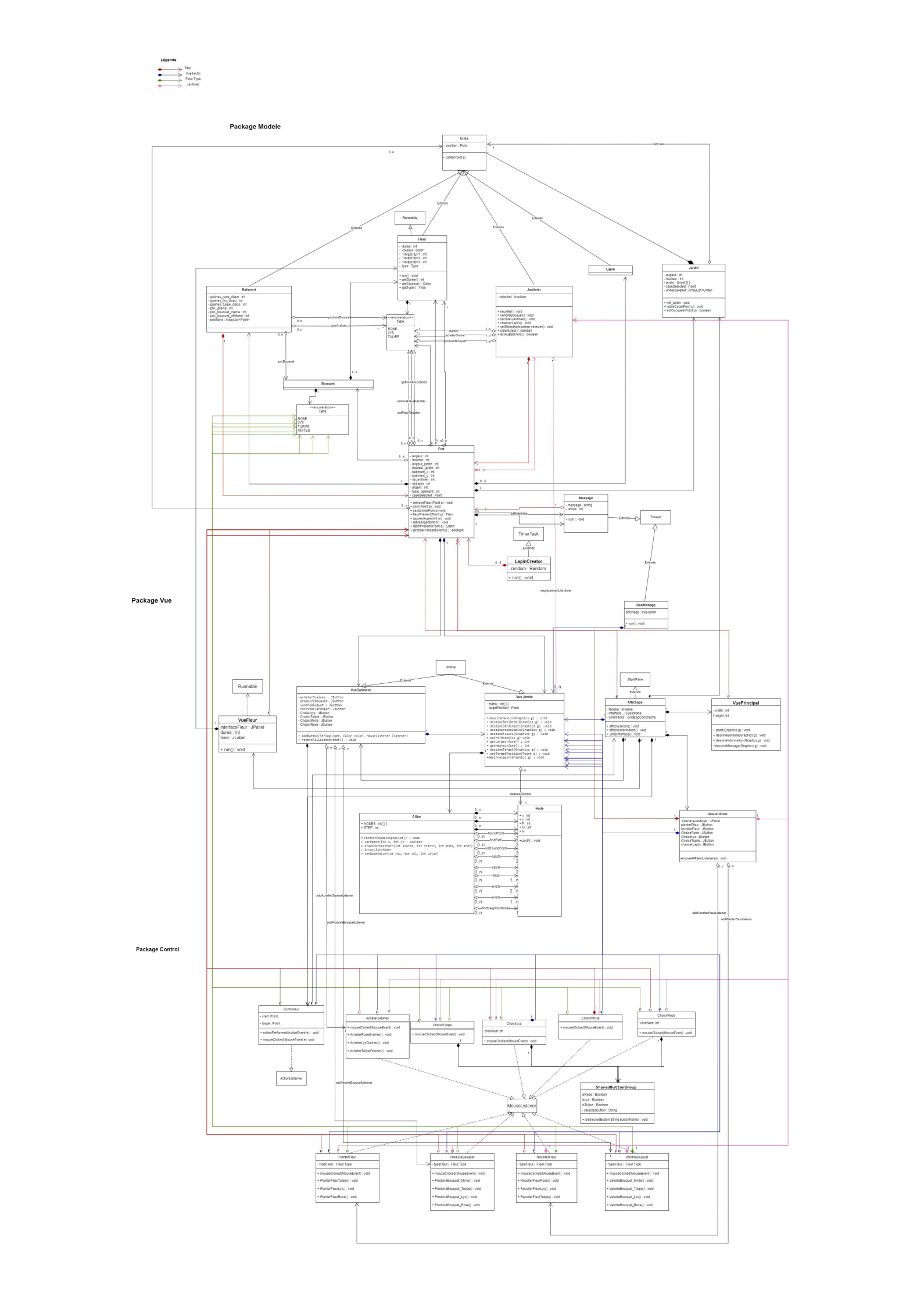
Les perspectives pour ce jeu sont nombreuses. Nous avons déjà évoqué la possibilité d'ajouter de nouvelles fonctionnalités telles que l'ajout de plus de graines différentes à planter ou encore l'inclusion d'un système météorologique. Cela permettrait de rendre le jeu plus complexe et plus réaliste.

De plus, nous envisageons d'optimiser la fluidité du jeu en modifiant certaines statistiques comme la vitesse de déplacement du jardinier ou la fréquence d'apparition des lapins. Ces ajustements permettront d'offrir une meilleure expérience de jeu aux utilisateurs.

Nous avons prévu d'ajouter de nouveaux personnages de jardiniers dotés de compétences différentes pour offrir aux utilisateurs une expérience de jeu plus variée. Nous avons déjà commencé à concevoir ces nouveaux personnages, mais pour l'instant, seule l'option de sélectionner un jardinier est disponible et les autres jardiniers à recruter ne sont pas encore fonctionnels. En outre, nous avons l'ambition d'étendre le jeu aux tablettes et aux téléphones portables afin de toucher un public plus large. Cela nécessitera une adaptation de l'interface utilisateur et de la jouabilité du jeu pour une utilisation optimale sur des appareils mobiles.

En somme, nous avons de nombreuses idées pour améliorer et développer le jeu dans le futur. Nous sommes impatients de travailler sur ces perspectives et de continuer à offrir aux utilisateurs une expérience de jeu agréable et stimulante.

Dans l'ensemble, ce projet nous a permis de développer nos compétences en programmation et en travail d'équipe, tout en nous offrant une expérience de jeu agréable et stimulante. Nous sommes impatients de continuer à améliorer et à développer notre jeu dans le futur.


Légende
Etat
VueJardin
FleurType
Jardinier
Package Modele
Unite
- position : Point
+ Unite(Point p)
Runnable
Fleur
- duree : int
- couleur : Color
- TIMESTEP1 : int
- TIMESTEP2 : int
- TIMESTEP3 : int
- type : Type
+ run() : void
+ getDuree() : int
+ getCouleur() : Color
+ getType() : Type
Lapin
Jardin
- largeur : int
- hauteur : int
- jardin : Unite[ ][ ]
- caseSelected : Point
- uniteSelected : ArrayList<Unite>
+ init_jardin : void
+ retireCase(Point p) : void
+ estOccupee(Point p) : boolean
Batiment
- graines_rose_dispo : int
- graines_lys_dispo : int
- graines_tulipe_dispo : int
- prix_graine : int
- prix_bouquet_meme : int
- prix_bouquet_different : int
- positions : ArrayList<Point>
Jardinier
- selected : boolean
+ recolter() : void
+ vendreBouquet() : void
+ recruterJardinier() : void
+ chasserLapin() : void
+ setSelected(boolean selected) : void
+ isSelected() : boolean
+ estAuBatiment() : boolean
<<enumeration>>
Type
ROSE
LYS
TULIPE
Bouquet
<<enumeration>>
Type
ROSE
LYS
TULIPE
MIXTES
Etat
- largeur : int
- hauteur : int
- largeur_jardin : int
- hauteur_jardin : int
- batiment_x : int
- batiment_y : int
- nbJardinier : int
- nbLapin : int
- argent : int
- taille_batiment : int
- caseSelected : Point
+ removeFleur(Point p) : void
+ click(Point p) : void
+ sacsInitiesPoint p):void
+ fleurPresent(Point p) : Fleur
+ ajouterArgent(int m) : void
+ retireArgent(int m) : void
+ lapinPresent(Point p) : Lapin
+ jardinierPresent(Point p) : boolean
Message
- message : String
- temps : int
+ run() : void
Thread
TimerTask
LapinCreator
- random : Random
+ run() : void
ReAffichage
affichage : VueJardin
+ run() : void
Package Vue
JPanel
Runnable
VueBatiment
- acheterGraines : JButton
- produireBouquet : JButton
- vendreBouquet : JButton
- recruterJardinier : JButton
- ChoisirLys : JButton
- ChoisirTulipe : JButton
- ChoisirMode : JButton
- ChoisirRose : JButton
+ addButton(String name, Color color, MouseListener listener)
+ removeAllListeners(data[ ]) : void
VueJardin
- matrix : int[ ][ ]
- targetPosition : Point
+ dessineJardin(Graphics g) : void
+ dessineBatiment(Graphics g) : void
+ dessineJardinier(Graphics g) :void
+ dessineRecolteLapin(Graphics g) : void
+ dessineFleurs(Graphics g) : void
+ paint(Graphics g): void
+ getLargeurCase( ) : int
+ getHauteurCase( ) : int
+ dessineTarget(Graphics g) : void
+ setTargetPosition(Point p) : void
+dessineLapin(Graphics g) : void
JSplitPane
Affichage
- fenetre : JFrame
- interface_ : JSplitPane
- constraints : GridBagConstraints
+ afficheJardin() : void
+ afficheInformation() : void
+ uniteInterface() : void
VuePrincipal
- width : int
- height :int
+ paint(Graphics g) : void
+ dessineBordure(Graphics g) : void
+ dessineInformation(Graphics g) : void
+dessineMessage(Graphics g) : void
VueFleur
interfaceFleur : JPanel
duree : int
time : JLabel
+ run() : void
AStar
- NODES : int[ ][ ]
- STEP : int
+ findRdFrNodeInOpenList() : Node
+ canReach(int x, int y) : boolean
+ drawShortestPath(int startX, int startY, int endX, int endY)
+ ArrayList<Node>
+ setNodeValue(int row, int col, int value)
Node
+ x : int
+ y : int
+ F : int
+ G : int
+ H
+ calcF() : void
VueJardinier
- interfaceJardinier : JPanel
- planterFleur : JButton
- recolterFleur : JButton
- ChoisirRose : JButton
- ChoisirLys : JButton
- ChoisirTulipe : JButton
- chasserLapin : JButton
removeAllFleurListeners() : void
Package Control
Controleur
- start :Point
- target :Point
+ actionPerformed(ActionEvent e) : void
+ mouseClicked(MouseEvent e) :void
ActionListener
AcheterGraines
+ mouseClicked(MouseEvent) : void
+ AcheterRoseGraines() : void
+ AcheterLysGraines() : void
+ AcheterTulipeGraines() : void
ChoisirTulipe
+ mouseClicked(MouseEvent) : void
ChoisirLys
- clickNum :int
+ mouseClicked(MouseEvent) : void
ChoisirMode
+ mouseClicked(MouseEvent) : void
ChoisirRose
- clickNum :int
+ mouseClicked(MouseEvent) : void
SharedButtonGroup
isRose : Boolean
isLys : Boolean
isTulipe : Boolean
- selectedButton : String
+ isSelectedButton(String buttonName) : void
MouseListener
PlanterFleur
- typeFleur : FleurType
+ mouseClicked(MouseEvent) : void
+ PlanterFleurTulipe() : void
+ PlanterFleurLys() : void
+ PlanterFleurRose() :void
ProduireBouquet
- typeFleur : FleurType
+ mouseClicked(MouseEvent) : void
+ ProduireBouquet_Mixte() : void
+ ProduireBouquet_Tulipe() : void
+ ProduireBouquet_Lys() : void
+ ProduireBouquet_Rose() : void
RecolterFleur
- typeFleur : FleurType
+ mouseClicked(MouseEvent) : void
+ RecolterFleurRose() : void
+ RecolterFleurLys() : void
+ RecolterFleurTulipe() : void
VendreBouquet
- typeFleur : FleurType
+ mouseClicked(MouseEvent) : void
+ VendreBouquet_Mixte() : void
+ VendreBouquet_Tulipe() : void
+ VendreBouquet_Lys() : void
+ VendreBouquet_Rose() : void