CHENG Jewin, RODRIGUEZ Paul-Alexis, CHIDIAC William, RIGOBERT Eliott

Cahier des charges : Projet PCII

1. Présentation :

Nous allons réaliser un jeu de stratégie en temps réelle, dans notre cas le jeu se déroule dans une jardinerie, l'objectif est de planter et récolter le plus de plantes possibles tout en les protégeant des lapins qui veulent les manger. Pour cela le joueur pourra contrôler différents jardiniers qui pourront planter et récolter les plantes et qui auront un certain rayon de protection contre les lapins. Le joueur gagne des points pour chaque plante qu'il a récolté ainsi que de l'argent pour pouvoir planter de nouvelles plantes. Le joueur doit réunir un certain nombre de points avant la fin du temps imparti pour gagner.

2. Contraintes :

Priorité 1 et difficulté 1 :

- Il faut créer une fenêtre de jeu contenant deux parties, la carte et les boutons.
- Il faut que les jardiniers soient visibles sur la carte sous la forme d'un carré coloré
- Il faut que les plantes soient visibles sur la carte sous la forme d'un rond coloré
- Il faut que les lapins soient visibles sur la carte sous la forme d'un triangle coloré
- Lorsqu'un lapin atteint une fleur, la fleur se fait manger par le lapin et elle disparaît

Priorité 1 et difficulté 2 :

- Il faut que le joueur puisse déplacer individuellement chaque jardinier dans l'environnement à l'aide de la souris en cliquant sur un jardinier, puis en cliquant sur une autre position de la carte.
- Lorsque l'on clique sur un jardinier, un menu doit s'ouvrir indiquant l'identité du jardinier, et les boutons offrant la possibilité d'arrêter le déplacement du jardinier et la possibilité de planter.
- Il faut que les plantes évoluent dans le temps (poussent) jusqu'à ce qu'elles soient prêtes à être cueillies, et cela de manière visible sur la carte.
- Il faut que des lapins apparaissent régulièrement aux extrémités du jardin.
- S'il n'y a pas de jardinier entre le lapin et la plante, alors le lapin peut se diriger vers cette plante.

Priorité 1 et difficulté 3 :

- Lorsque l'on a sélectionné un jardinier, qu'on clique ensuite sur une plante et que le joueur est dans le rayon d'interaction de la plante, alors la plante doit être cueillie.
- Les lapins ne restent pas constamment dans le jardin. Il faut qu'au bout d'un certain temps, le lapin sorte du jardin ou meure.

Priorité 2 et difficulté 1 :

- Lorsqu'on clique sur la plante sans qu'un jardinier soit sélectionné ou que la plante n'a pas fini de croître, alors un menu affichant ses informations apparaît à droite à la place du menu du jardinier.
- Le score doit être affiché et évoluer.

Priorité 2 et difficulté 2 :

- Lorsqu'un jardinier cueille une plante, l'argent et le score doivent augmenter de la quantité que cette fleur apporte.
- Le joueur gagne s'il atteint un certain nombre de points.
- Lorsqu'elles ne sont pas prêtes à être cueillies, les plantes ont une barre de progression. Quand elles sont prêtes à être cueillies, cette barre doit disparaître pour indiquer au joueur la possibilité de cueillir la fleur.

Priorité 2 et difficulté 3 :

- Les parties sont limitées dans le temps : il faut afficher le temps restant. Le joueur doit gagner avant la fin du temps imparti ; sinon, il a perdu.
- Planter doit coûter de l'argent, et celui-ci doit être prélevé de la bourse du joueur. S'il n'a pas assez d'argent, le joueur ne peut pas planter. Plus la plante est chère, plus elle rapporte d'argent et de points.

Priorité 3 et difficulté 1 :

- Un menu de défaite et de victoire doit s'afficher à la fin, avec un bouton pour relancer le jeu.
- Un menu au début pour lancer le jeu doit être affiché.
- Faire des variétés de plantes.
- Le temps de pousse peut varier en fonction de la fleur : plus la fleur rapporte d'argent et de points, plus elle met du temps à pousser.

Priorité 3 et difficulté 2 :

- En cliquant sur le bouton "Planter" du jardinier, un menu représentant la boutique doit apparaître avec les différentes variétés de plantes, un bouton "Acheter" et un bouton pour afficher les caractéristiques d'une plante. Le bouton "Acheter" n'est visible que si le joueur a assez d'argent.
- Générateur de plantes sur la carte de manière aléatoire

3. Fonctionnalités à implémenter :

Temps de travail en termes d'analyse, conception, développement et test pour chaque fonctionnalité :

- Apprentissage pour mettre les Layout et les différents composants d'un Panel : 30 mins
- Visibilité des jardiniers : 30 mins
- Déplacement des jardiniers : 45 mins
- Organisation des menus : 30mins
- Menu du jardinier : 15 mins
- Fonctionnalités du jardinier (arrêter le déplacement, planter) : 45 mins
- Visibilité des plantes : 30mins
- Évolution des plantes : 15 mins
- Cueillir des plantes + rayon de cueillette : 30 mins
- Effet de la cueillette des plantes sur le score et l'argent : 15 mins
- Menu de la plante + conditions d'interaction : 30 mins
- Apparition des lapins : 30 mins
- Déplacement des lapins vers les plantes + conditions : 45 mins
- Lapin qui mange une plante : 15 mins
- Disparition des lapins : 15 mins
- Système de boutique pour les plantes : 45mins
- Variété de fleurs : 30 mins
- Argent + visuel : 15 mins
- Défilement du temps + visuel : 15 mins
- Affichage du score + visuel : 15 mins
- Objectif de points + visuel : 15 mins
- Condition de défaite : 15 mins
- Condition de victoire : 15 mins
- Générateur de plantes : 30 mins

Diagramme de Gantt

	0h00 - 1h00	1h00 - 2h00	2h00-3h00	3h00-3h30
Jour 1 :				
Apprentissage des layouts				
Menu du jardinier				
Fonctionnalités du jardinier				
Fenêtre de jeu				
Rédaction du document de conception				
Visbilité des jardiniers				
Déplacement des jardiniers				
Rédaction du document de conception				
Visibilité des plantes				
Évolution des plantes				
Cueillir des plantes + rayon de cueillettes				
Rédaction du document de conception				
Déplacement Lapin				
Jour 2 :				
Intégration du code				
Intégration du code conversion en Jpanel				
Intégration du code sur la fenêtre de jeu principal				
Organisation des menus				
Système de boutique pour les plantes				
Jour 3 :				
Intégration du code				
Rédaction du document de conception				
Intégration du code				
Variétés de fleurs				
Déplacement des lapins vers les plantes + conditions				
Apparition des lapins				
Lapin qui mange une plante				
Jour 4 :				
Affichage du score + visuel				
Argent + visuel				
Rédaction du document de conception				
Menu de la plante + conditions d'interaction				
Rédaction du document de conception				
Simulation de boutique				
Intégration des lapins				
Disparition des lapins				

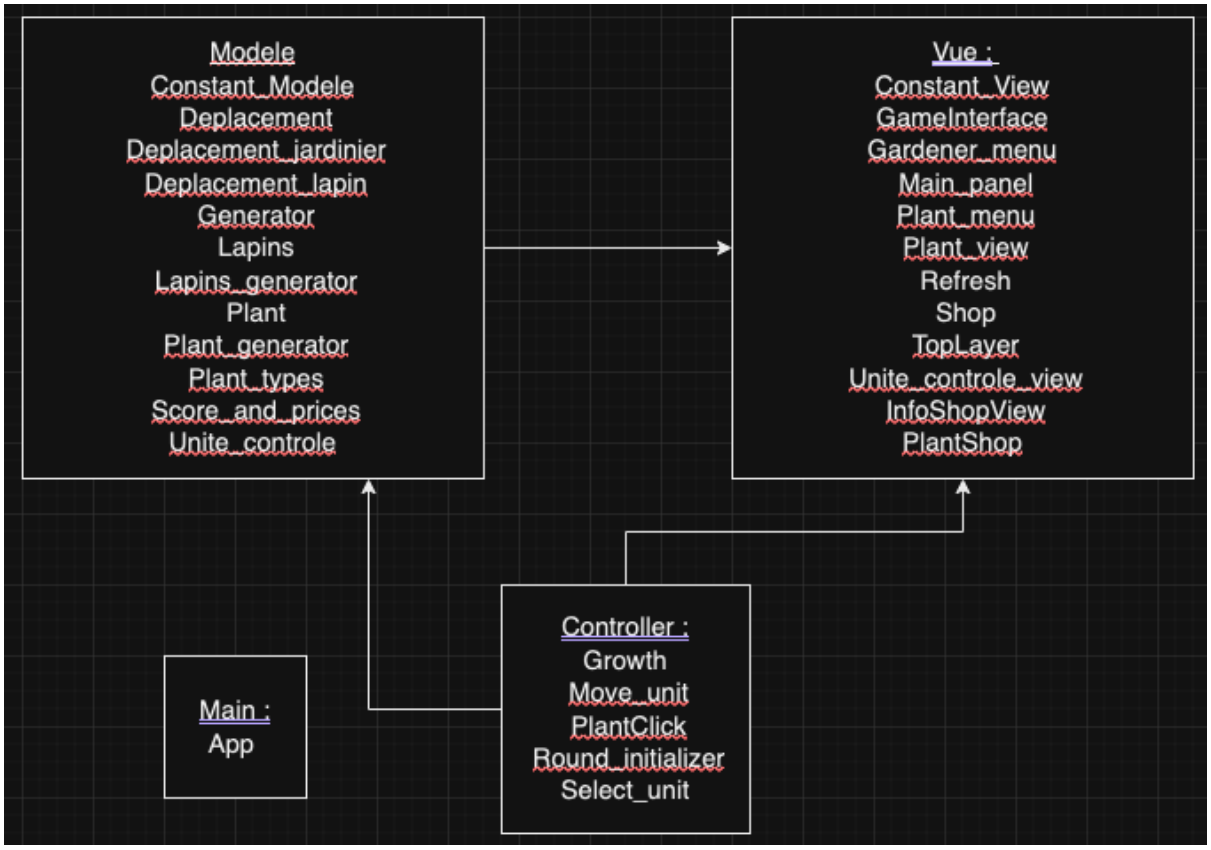
Jour 5 :				
Rédaction du document de conception				
Réorganisation des menus				
Intégration du code				
Système d'argent				
Rédaction du document de conception				
Menu de début/fin				
Rédaction du document de conception				
Effet de la cueillette des plantes sur le score et l'argent				
Jour 6 :				
Rédaction du document de conception				
Objectif de points + visuel				
Défilement du temps + visuel				
Condition de défaite				
Condition de victoire				
Rédaction du document de conception				
Intégration du code				
Intégration du code				
Générateur de plantes				
Couleur des contributeurs	Jewin	William	Paul-Alexis	Eliott

4. Conception générale :

Précision a améliorer pour l'ensemble

On a choisi de suivre le modèle MVC pour la conception de notre jeu pour séparer les différentes parties et faciliter le développement et l'ajout de nouvelles fonctionnalités.

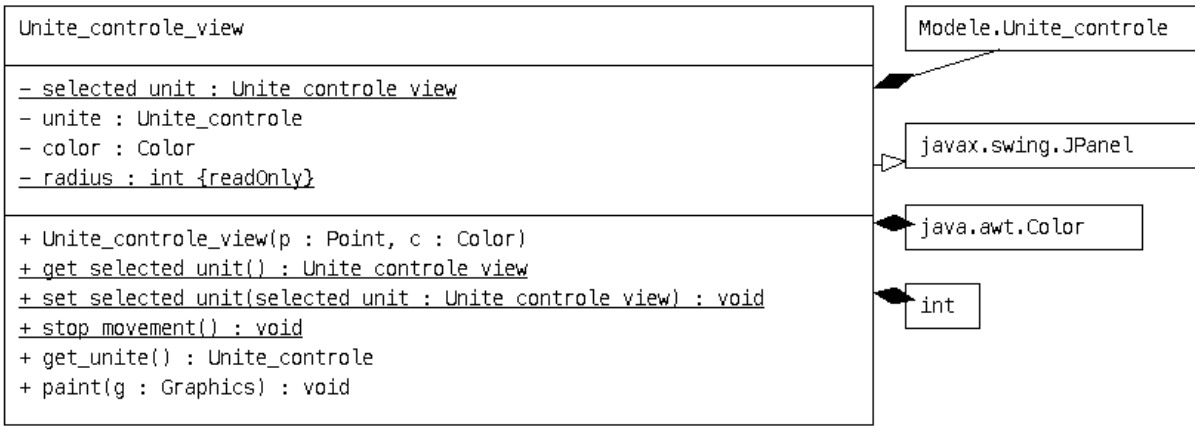
- Modèle : Constant_Modele, Deplacement, Deplacement_jardinier, Deplacement_lapin, Generator, Lapins, Lapins_generator, Plant, Plant_generator, Plant_types, Score_and_prices, Unite_controle
- Vue : Constant_View, GameInterface, Gardener_menu, Main_panel, Plant_menu, Plant_view, Refresh, Shop, TopLayer, Unite_controle_view, InfoShopView, PlantShop
- Controller : Growth, Move_unit, PlantClick, Round_initializer, Select_unit
- Main : App



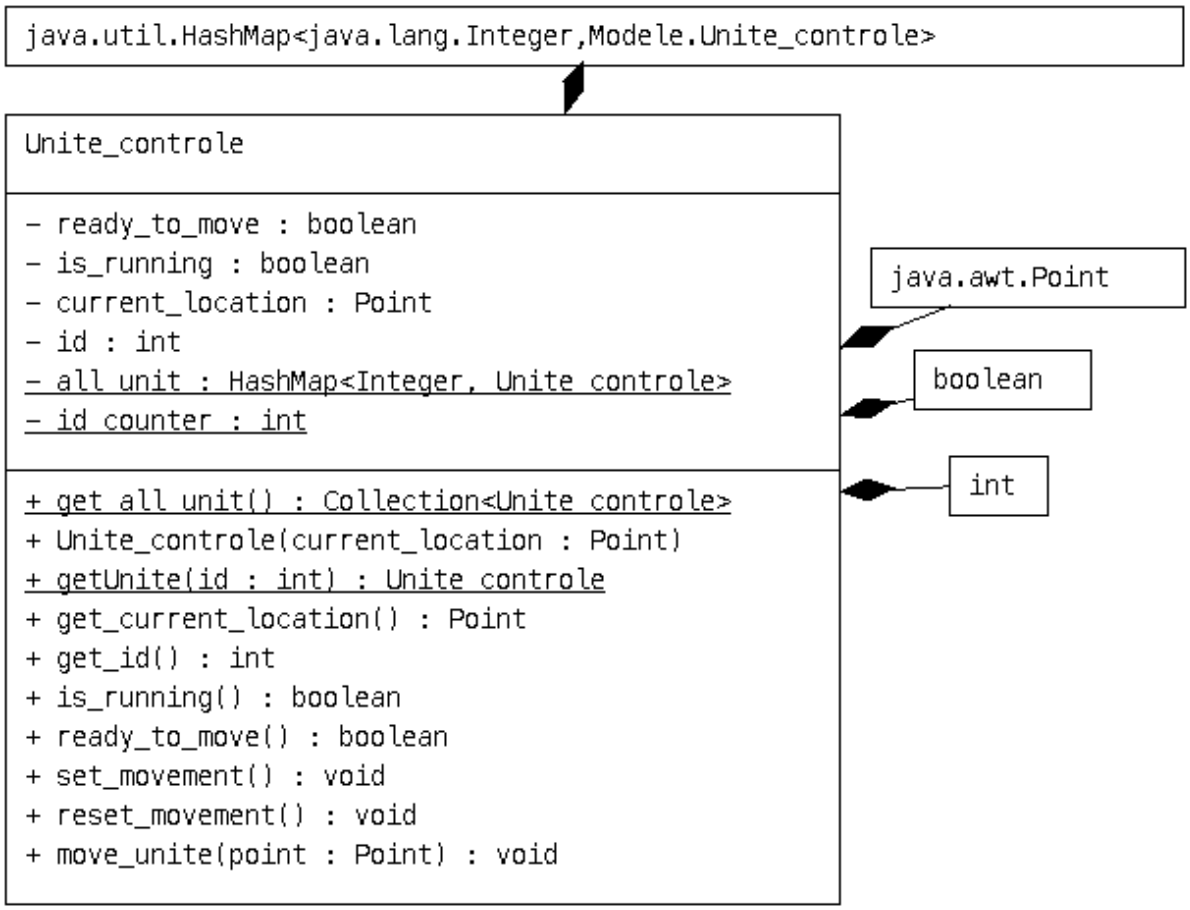
5. Conception détaillée :

- Visibilité des jardiniers :

Chaque jardinier est un JPanel donc la vue doit hériter de JPanel. On y stocke l'unité donc le jardinier avec une couleur et un rayon. On a aussi un attribut statique selected_unit qui stocke l'unité sélectionné. On a les getters pour selected_unit(), get_unite() et un setter set_selected_unit(). On a les méthodes : stop_mouvement() qui appelle reset_mouvement() de l'unité pour la fonctionnalité stay. On a aussi le dessin du jardinier sur la carte.



La classe pour le jardinier est composé d'un entier id, d'un id_counter, de coordonnées de type Point, une HashMap commune à tous les jardiniers qui les stocks et de booléen is_running et ready_to_move pour arrêter la course ou continuer. Les méthodes sont des getters et un setIs_set_movement() et reset_movements() pour stopper la course et d'un move_unit pour déplacer le jardinier.

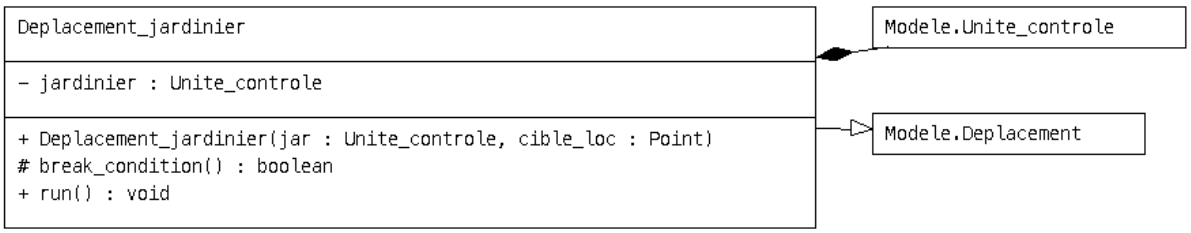


- Déplacement des jardiniers :

Ceci est le thread de déplacement du jardinier qui hérite de Deplacement.

Il a un attribut qui contient le jardinier. Le constructeur prend en argument une Unite_controle et une position, appelle le constructeur de la classe mère avec la position et initialise l'attribut.

Il y a break_condition() qui renvoie si la négation du fait que l'unité court et run() qui déplace le jardinier avec set_movement() et reset_movement() petit à petit.



- Fonctionnalités du jardinier (arrêter le déplacement, planter)

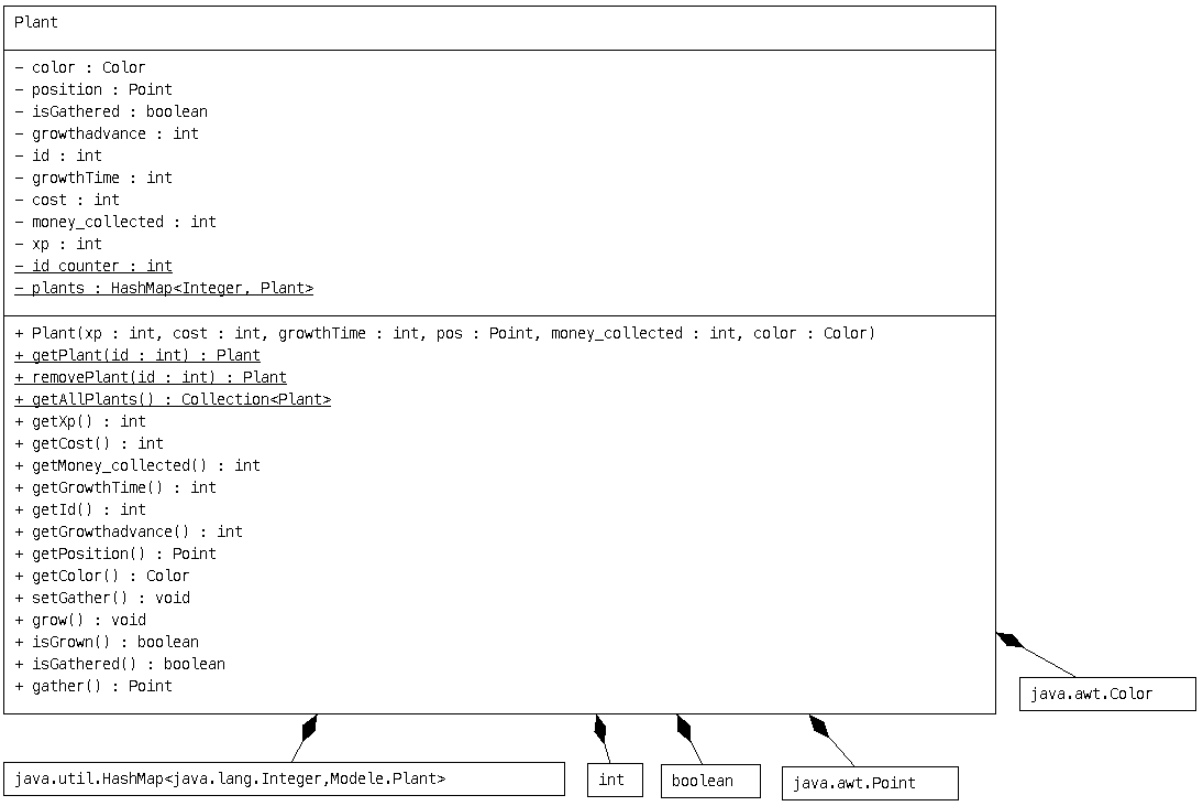
Les fonctionnalités du jardinier sont des boutons du menu du jardinier :

Stay appelle la méthode stay de main_panel qui change le menu au menu standard et appelle la procédure stop_mouvement de Unite_controle_view qui vide l'unité sélectionné et appelle stop_movement de l'unité qui fait à son tour l'arrêt des mouvements en changeant les booléens ready_to_move et is_running.

Shop qui appelle setEnabled de PlantShop et la méthode showShop s'il n'est pas affiché et le cache avec hideShop sinon. SetEnabled permet de rendre interactible les boutons acheter de chaque plante si le joueur a assez d'argent et

- Plantes :

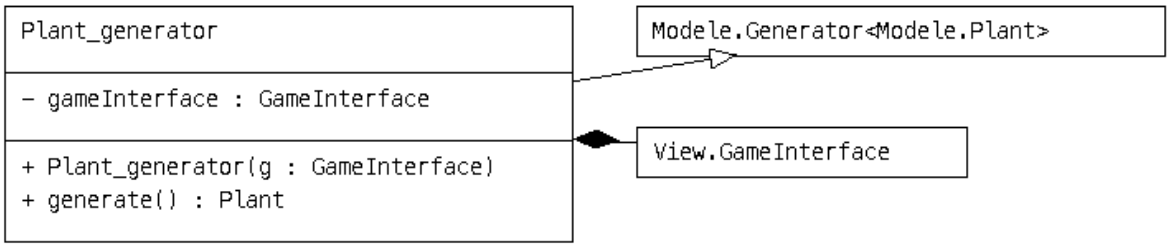
  Les plantes sont représentées par une classe qui contient tous les attributs et méthodes qui permettent de mettre en place les fonctionnalités décrites ci-dessous.



- Générateur de plantes :

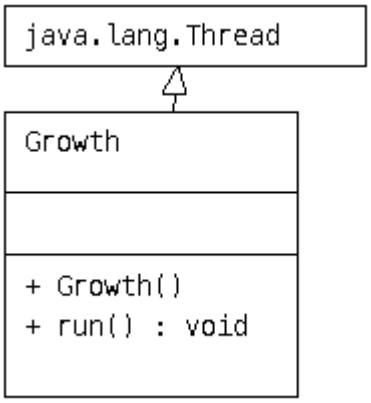
  Pour la génération des plantes, nous avons créé une classe abstraite Generator, avec une méthode generate(), la classe prend en paramètres la quantité d'objets à générer au départ et le temps entre la génération de deux objets. Ensuite une classe Plant_Generator hérite de cette classe et s'initialise avec les deux paramètres ci-dessus qui prennent les valeurs des constantes du modèle PLANT_GENERATION_DELAY et NB_PLANT_INIT. La classe prend aussi en paramètre une gameinterface afin de rendre visible les plantes à générer. Pour la méthode generate(), on crée des coordonnées qui sont dans la fenêtre de jeu de manière aléatoire, on choisit un type de fleur avec une probabilité inversement proportionnelle à la rareté (plus une plante est rare moins elle a de chances d'apparaitre). On génère ensuite une plante du type

obtenu au coordonnées obtenues, on rajoute cette plante sur le plateau de jeu et on la renvoie.



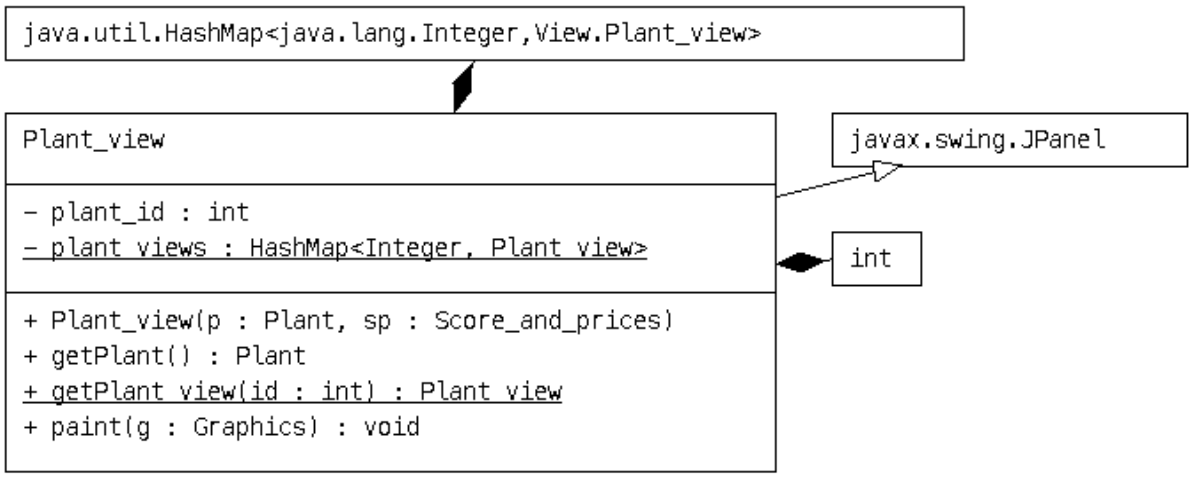
- Évolution des plantes (pousse) :

Pour l'évolution de la plante nous avons un attribut, qui est entier, dans la classe Plant qui indique le niveau de pousse. Pour limiter la croissance un autre entier indique la croissance maximale (durée de croissance), ensuite un thread fait évoluer le niveau de pousse jusqu'à la limite. L'ensemble des plantes est stocké dans une table de hachage dans la classe Plant qui est donc un attribut de classe et non d'instance. Le thread agit sur toutes les plantes et fait donc pousser toutes les plantes.



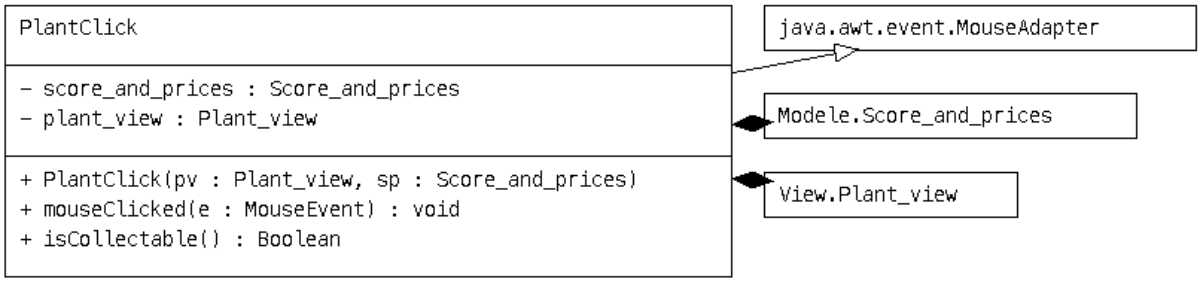
- Visibilité des plantes :

Pour l'affichage chaque plante est contenue dans un Jpanel ne contenant que cette plante, ensuite ce JPanel est ajouté au JPanel principal qui est lui-même contenu dans une JFrame. Dans le Jpanel de la plante on dessine un cercle puis on rajoute le JPanel dans le JPanel principal à la position de la plante, cette position est un attribut de la plante. Pour afficher l'évolution de la pousse de la plante, nous allons dessiner en plus du cercle représentant la plante, un rectangle de taille fixe. Pour ceci nous calculons le ratio entre la pousse actuelle et la pousse maximale, que nous multiplions par la taille de la barre pour obtenir la taille de la sous barre (colorée) qui va donner l'indication du niveau de pousse. Ensuite nous affichons le contour du rectangle externe et le rectangle coloré, une fois que la plante a fini de pousser la barre change de couleur (vert à rouge).



- Cueillir des plantes + rayon de cueillette :

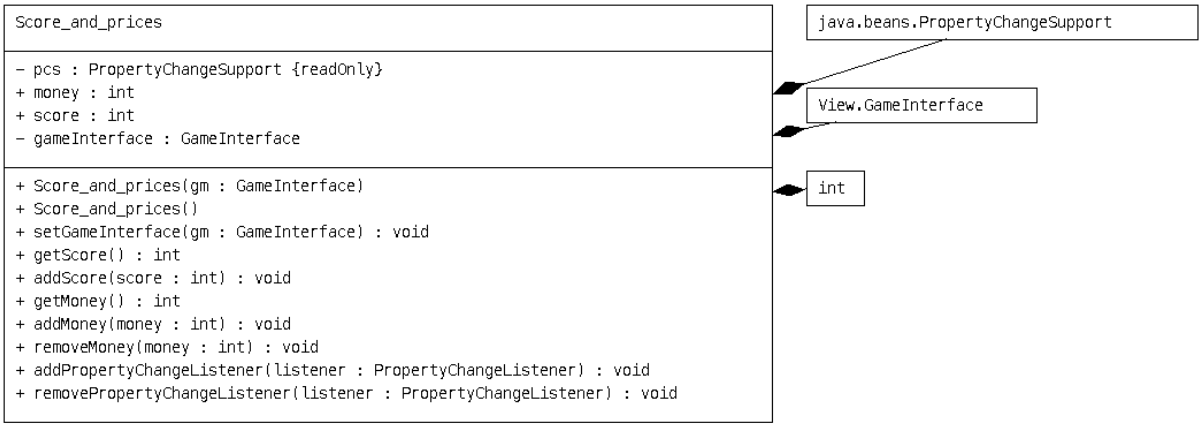
Pour la cueille des plantes nous avons un MouseAdapter qui attends à ce que l'on clique sur une plante, pour éviter que l'on puisse cueillir une plante en cliquant n'importe où dans la fenêtre nous vérifions que la position de la souris lors du click est dans l'aire du cercle de la plante, et ce pour toutes les plantes présentes. Nous vérifions aussi si au moins un jardinier est contenu dans un cercle de rayon modifiable (afin de rendre plus ou moins simple la cueille) autour de la plante, s'il n'y en a pas, on ne peut pas cueillir la plante. Si le click s'effectue effectivement dans l'aire du cercle d'une plante, on récupère l'argent et les points générés par celle-ci et on l'enlève du terrain.



- Système du score et d'argent :

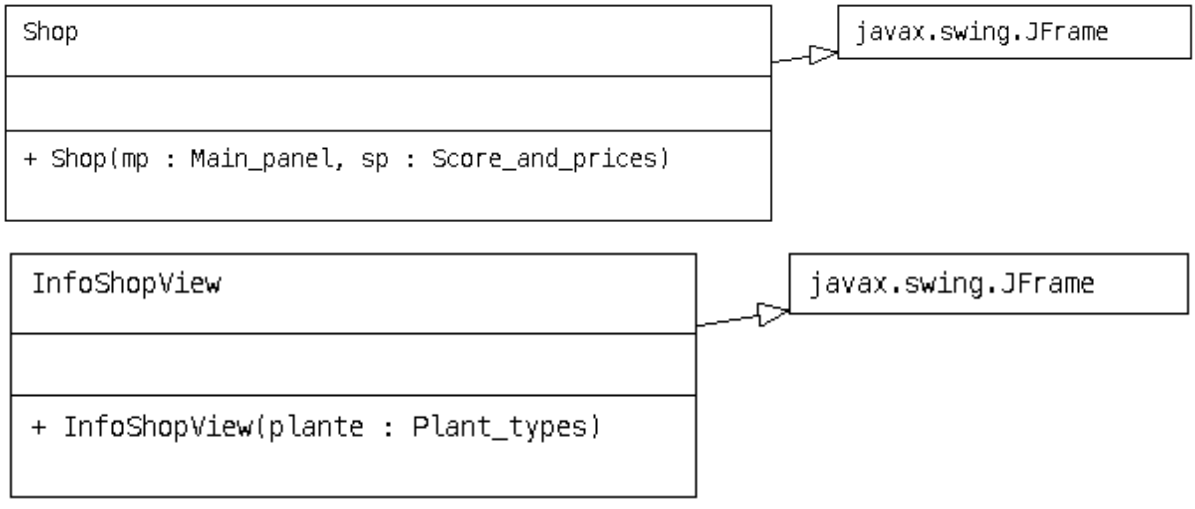
  Pour le système d'argent nous avons une classe principale score_and_prices, qui garde l'information du score actuel et de l'argent possédé, cette classe possède une instance de l'interface de jeu afin de pouvoir afficher l'argent possédé et le score actuel. Il y a deux constructeurs pour cette classe un qui prend l'interface du jeu en paramètre et qui construit une instance complète et un autre qui permet de construire une instance incomplète et de rajouter l'interface de jeu plus tard (à l'aide de la méthode setGameInterface). Lorsque l'on cueille les plantes on appelle la méthode addScore et addMoney de la classe qui vont s'occuper de récupérer le score et l'argent que rapporte la plante qui vient d'être cueilli, de mettre à jour la vue avec la nouvelle quantité d'argent et le nouveau score, à l'aide de la méthode updateMoney et updateScore de gameInterface. La méthode addScore vérifie également si on dépasse l'objectif de score, auquel cas on déclenche la méthode win() de gameInterface. La méthode addMoney déclenche aussi un property change pour notifier que l'argent a été modifié et envoie la nouvelle valeur aux objets qui doivent être notifiés. C'est

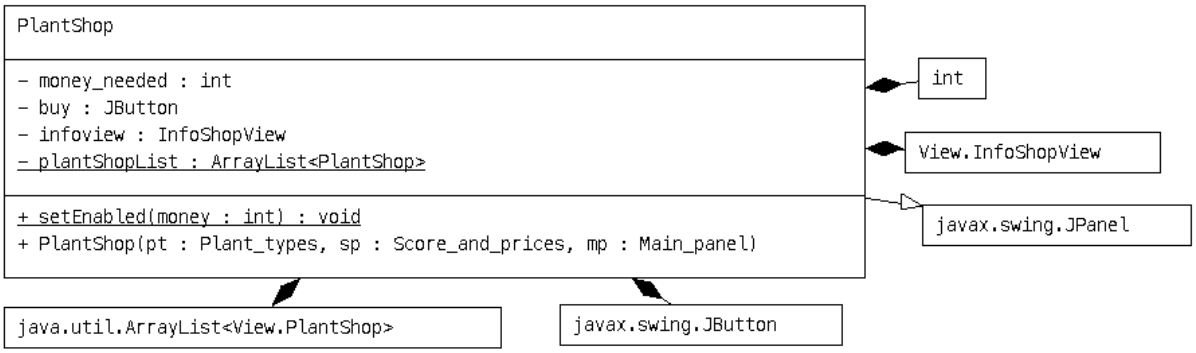
également le cas de la méthode removeMoney. Il y a également deux méthodes qui permettent à la classe d'avoir des PropertyListener (classe de java), ce qui nous permettra notamment de garder un contrôle lors de l'achat des plantes (d'où le fait que l'on a besoin de notifier certains objets du changement dans la quantité d'argent afin de savoir si on peut acheter ou non).



- Shop/Magasin :

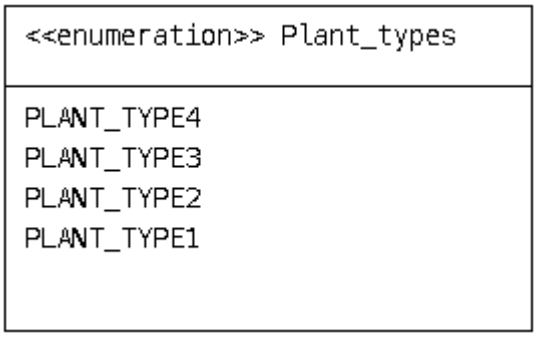
Nous avons un shop par plante qui permet d'acheter une plante spécifique, chacun des shops est un jpanel, contenant un bouton pour l'information (celui-ci fait apparaître un JPanel avec des informations concernant la plante) et un bouton d'achat. Pour éviter que l'on puisse acheter des plantes si on ne possède pas assez d'argent, on active ou désactive les boutons qui déclenchent l'achat selon le fait que l'on possède ou non suffisamment d'argent. On peut contrôler cela à l'aide du mécanisme présenté ci-dessus (PropertyListener) ou en gardant le prix dans la classe du magasin de chaque plante et en vérifiant que l'argent possédé est suffisant (nous avons opté pour la deuxième option). Le Background du shop est réglé selon la plante, c'est-à-dire que la couleur et le nom affichés dépendent de la plante. Il y a également une liste des différents shops afin que l'on puisse les réunir dans un JPanel plus grand qui est représenté par la classe Shop, cela nous permet aussi de désactiver les boutons en une seule fois. Dans la classe shop on se contente de créer et récupérer les shops de chacune des plantes et de les ajouter dans le JPanel du shop principal. Ce JPanel principal est celui qui apparaît lorsque l'on clique sur le button shop du menu des jardiniers





- Variété de fleurs :

Les différentes plantes se trouvent dans cette classe.



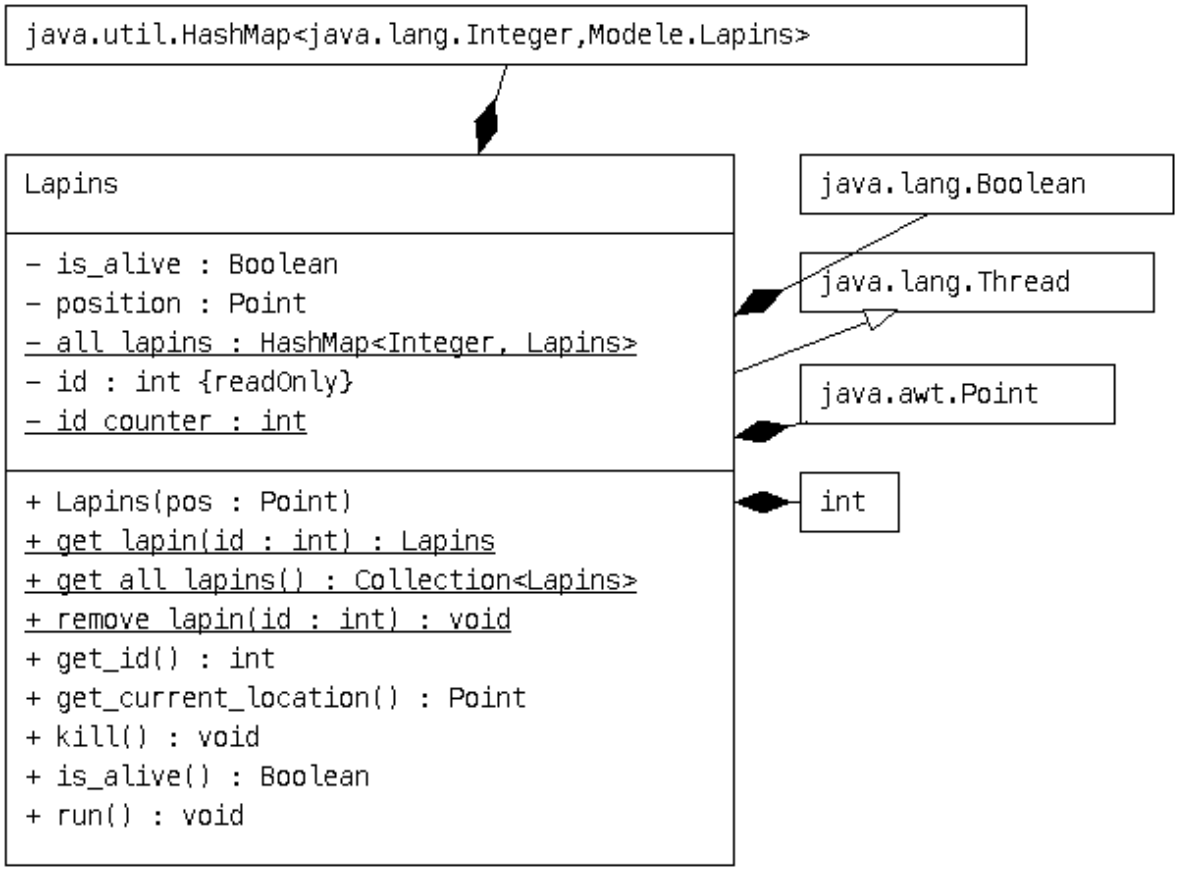
- Visibilité des lapins :

Les possèdent en attribut : boolean is_alive, Position position, int id qui est final et deux constantes HashMap<Integer, Lapins> all_lapins et id_counter.

Le constructeur initialise ces attributs, incrémente le nombre id_counter et ajoute le lapin crée dans la HashMap de lapins.

On dispose des méthodes :

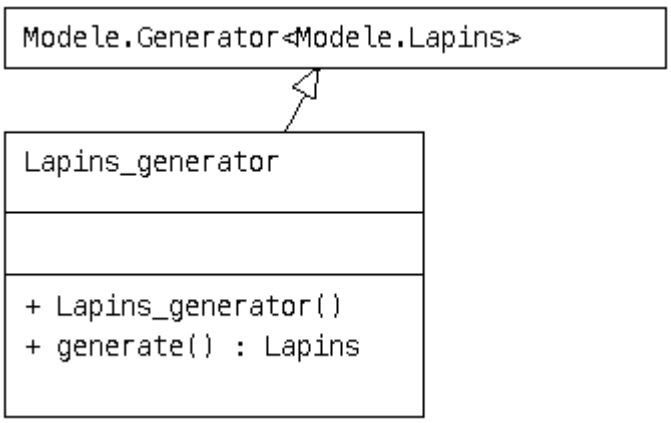
- Les getters qui renvoient ses attributs: get_all_lapins(), get_id(), get_current_location(),
- Get_lapin(id) qui renvoie le lapin d'id dans all_lapins
- Une méthode remove_lapin(id) qui enlève le lapin avec l'id de all_lapins
- Kill() qui change le boolean is_alive
- Run() qui représente le temps de survie du lapin et le tue après le sleep.



Les lapins sont affichés avec la méthode paint dans Main_panel qui itère sur les lapins et appelle paint_lapin(graphics g, Point p). Paint_lapin crée une ovale de couleur à la position donnée.

- Apparition des lapins/ Générateur de lapin :

Le générateur de lapins étend la class abstraite Générateur qui est un Thread. Elle implémente la méthode generate() qui crée un lapin avec sa position et son thread de déplacement et le renvoie. Le constructeur appelle le constructeur de la classe mère et définit le temps d'intervalle d'apparitions des lapins et le nombre d'objet initial. Ces nombres sont des constantes de la classe Class_modele. Les lapins générés sont ajoutés à leur instanciation dans all_lapins.



- Déplacement des lapins vers les plantes + conditions :

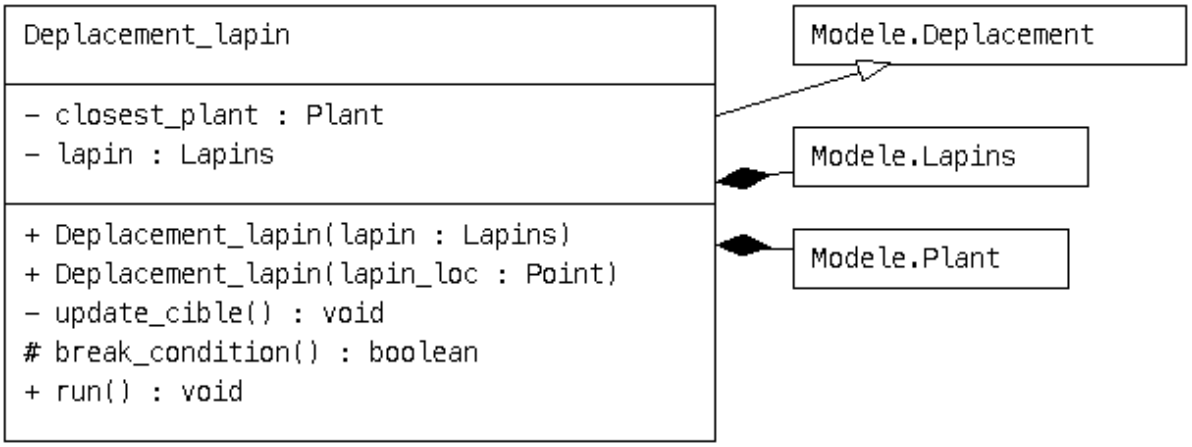
Cette classe étend la classe abstraite Déplacement qui est un thread. Elle a en argument un lapin et la plante la plus proche. Le constructeur prend un lapin en argument et appelle le constructeur de la classe mère avec la position du lapin puis initialise son propre attribut lapin. L'autre constructeur construit un lapin à partir d'une position.

On a les méthodes :

Update_cible() qui cherche la plante la plus proche et le jardinier le plus proche grâce au fait qu'on a une HashMap sur les jardiniers et les plantes accessibles par n'importe quelle instance de ces deux classes, change de cible pour la plante trouvée s'il y en a une et si le jardinier est entre la plante et le lapin alors la cible devient la moitié de la distance entre le lapin et la plante.

Break_condition() qui renvoie si la plante cible a été cueillie.

Et run () qui envoie le lapin vers son terrier s'il n'est pas vivant sinon il met à jour sa cible et mange la plante s'il l'atteint.



- Lapin qui mange une plante :

Le lapin mange une plante s'il l'atteint dans la classe Deplacement_lapin, alors à partir de l'id de la plante on l'enlève avec la procédure removePlante(id) et du Panel à partir de la plant_view obtenu grâce à la procédure getPlant_view(id).

- Disparition des lapins

Les lapins quand ils changent en mode mourir, doivent se déplacer vers leur terrier qui se situe en haut à gauche et un appel à Lapins.remove_lapin est fait pour l'enlever de la HashMap

- Interface du jeu (GameInterface) :

On utilise les APi's javax.swing.BorderFactory/JButton/JFrame/JPanel, java.awt.BorderLayout/CardLayout/Color/Dimension et les différentes classes de nos packages Constroller.Round_initializer, Modele.Plant, Modele.Score_and_prices.

Structure des données : un JPanel gameState et un CardLayout states pour changer de fenêtres (départ, jeu, fin), les différents menus (gardenerPanel, plantPanel), le score sc, le windowPanel

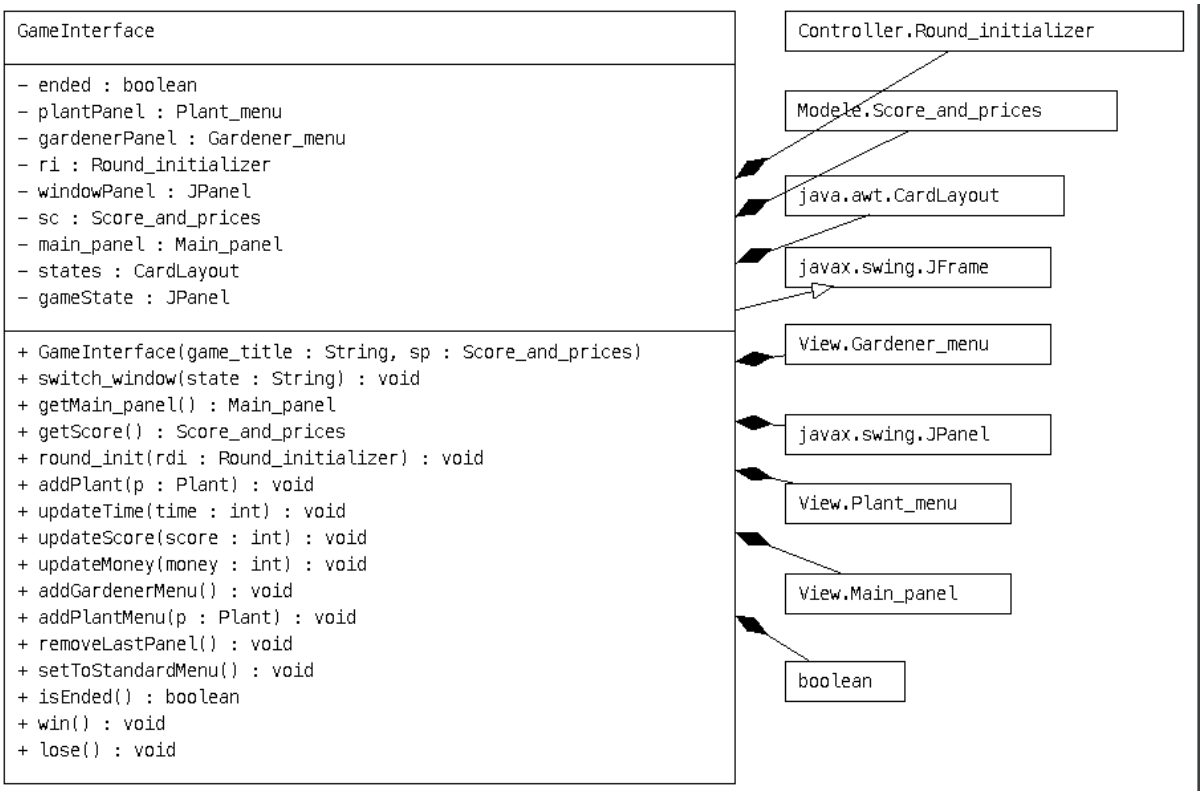
contient la carte main_Panel à gauche et le menu TopLayer à droite qui affiche aussi un autre menu selon l'entité sélectionner (lapin, jardinier ou plante), ri pour le Round_initializer qui initialise les entités et les générateurs du jeu et booléen ended qui indique quand le jeu est terminé. La classe hérite de JFrame, elle sera la fenêtre de jeu.

Dans le constructeur, on prend en argument un titre et le score (Score_and_prices), il faut initialiser la carte qui est un Main_panel et le score qui prend la classe en elle-même en argument, le menu du jardinier avec la carte et le score donné en argument et un JPanel windowPanel avec un BorderLayout puis donné en attribut qui sera le JPanel contenant la carte à gauche et le menu TopLayer à droite. On initialise ensuite les différents JPanel représentant les fenêtres de début et de fin victoire/défaite qu'on ajoute à gameState en plus du Panel du jeu. Ce panel et CardLayout servent à échanger les panels à afficher selon la situation. Il faudra rendre visible tous ces panels puis ajouter gameState à la classe elle même.

Il y a également un thread de rafraîchissement Refresh pour la classe GameInterface.

Il y a en plus les méthodes :

- Switch_window qui prend un état sous la forme d'un string et appelle switch de gameState avec cet état. Cela a pour effet d'échanger le panel avec un autre choisit selon l'état.
- Deux getters getMain_panel() et getScore()
- Round_init qui prend un Round_initializer et le donne à l'attribut ri
- updateScore qui prend un score et qui appelle la méthode update score du TopLayer (composant à l'index 1 du windowPanel)
- UpdateTime qui prend un entier time et mets à jour le temps affiché dans TopLayer
- UpdateMoney qui prend un entier money et mets à jour l'argent affiché dans TopLayer
- addGardenerMenu qui appelle removeLastPanel, ajoute le menu du jardinier au centre et change le titre de l'en-tête par l'id du jardinier sélectionné avec la fonction get_selected_unit de Unite_controle_view
- addPlantMenu qui appelle removeLastPanel, initialise l'attribut qui contient le menu de la plante , l'ajoute au toplayer au centre et change le titre de l'en-tête par l'id de la plante
- removeLastPanel qui enlève le panel récemment ajouté c'est à dire s'il y a plus d'un composant au TopLayer.
- setToStandardMenu qui appelle removeLastPanel et remet l'en-tête au nom Menu.
- IsEnded() qui renvoie l'attribut ended
- Win() qui appelle switch_window avec l'état win
- Loose() qui fait de même mais avec l'état loose



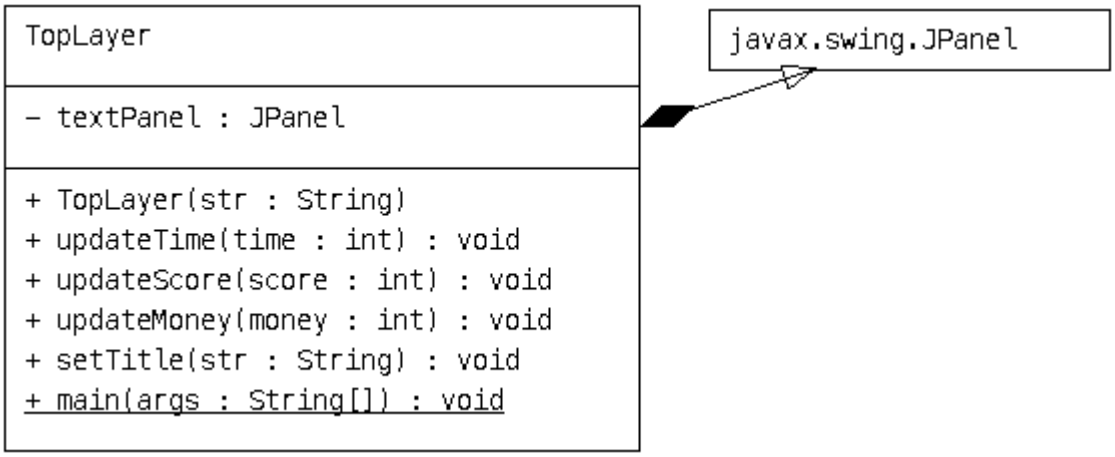
- Organisation des menus du jardinier et de la plante :

Le menu situé à droite de la fenêtre de jeu est composé de plusieurs JPanel. Le menu qui englobe tout est un TopLayer, au nord un autre JPanel qui utilise un GridLayout et qui possède deux JTextField, le premier pour le score et le second est un en-tête pour indiquer l'id de l'entité sélectionner. Au centre, on y ajoutera le menu du jardinier ou de la plante.

Les APIs utilisés sont : javax.swing.BorderFactory/JButton/JFrame/JPanel/JTextField et java.awt.BorderLayout/Color/Dimension/GridLayout.

La classe TopLayer, hérité de JPanel, utilise un borderLayout et possède en attribut le textPanel qui représente les informations du jeu et l'id du menu de l'entité en question en utilisant un GridLayout contenant les JTextField ajouté à lui-même. On a en plus des méthodes updateTime(time), updateScore(score), updateMoney(money) et setTitle(str) qui modifie respectivement le temps, le score, l'argent et le titre de l'en-tête du textPanel en attribut et

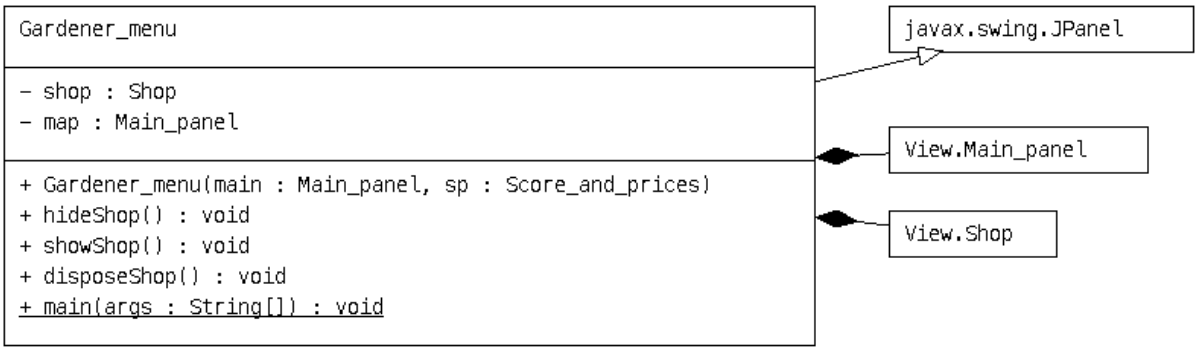
ce sont les composants à l'index 0, 1, 2 et 3.



- Menu du jardinier :

Les APIs utilisés sont : java.awt et javax.swing.BorderFactory/JButton/JFrame/JPanel.

Le menu du jardinier hérite de JPanel et est constitué d'un attribut Main_panel qui est la carte du jeu nécessaire à l'ajout des plantes par la boutique et la boutique. Le constructeur initialise l'attribut en créant un Shop et le cache avec hideShop() puis ajoute les boutons du jardinier qui sont plant et stay. Plant a pour effet de rendre le Shop visible avec showShop() et stay appelle la méthode stay du Main_panel. Le constructeur a en argument main qui représente la carte et sp le score qui sera utilisé pour la boutique.

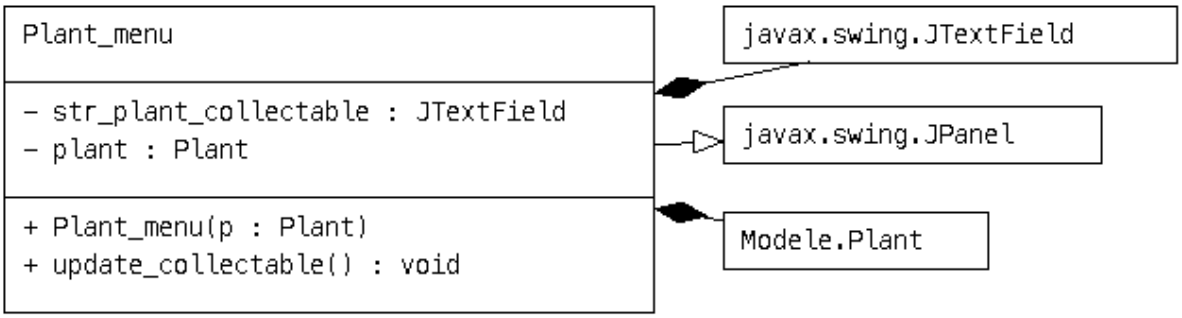


- Menu de la plante + conditions d'interaction :

Les APIs utilisés sont : javax.swing, java.awt et la classe Plant du package Modele de notre implémentation.

Le menu de la plante hérite de JPanel et possède en attribut une Plant et un JTextField qui représente si la plante peut être ramassé. Le constructeur initialise le menu avec un GridLayout et ses différents JTextField qui affiche les informations de la plante qu'on ajoute au menu lui-même. Il y a une méthode update_collectable() pour changer l'attribut.

Le menu de la plante n'apparaît que si on ne peut pas la collecter lorsqu'on clique dessus, cette condition est mise en œuvre dans le Controller de la plante PlantClick.



- Objectif de points + visuel :

La constante qui indique le nombre de points à atteindre se situe dans la classe Constant_modele. L'objectif de point est affiché par un JTextField dans TopLayer.

- Défilement du temps + visuel :

La constante qui indique la durée de la partie se trouve dans Constant_view. Le temps diminue avec le Thread Round_intializer.

- Condition de défaite :

La condition de défaite arrive lorsque le temps s'est écoulé on appelle alors la méthode lose() de GameInterface.

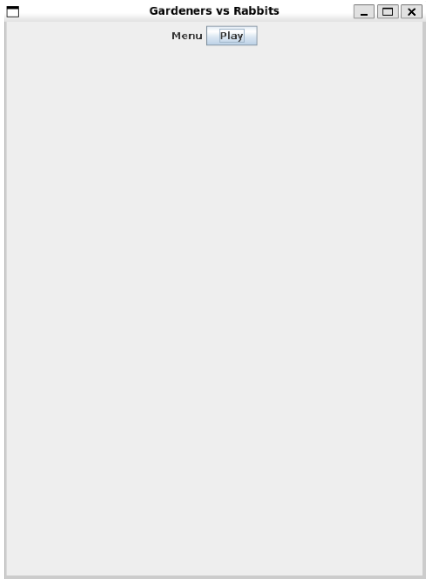
- Condition de victoire :

La condition pour gagner est vérifié au moment de l'appel de addScore() de Score_and_prices dans PlantClick, c'est à dire à chaque fois qu'une plante est ramassé. Si la condition de victoire est remplie, alors la classe Score_and_prices appelle la méthode win() de la classe GameInterface.

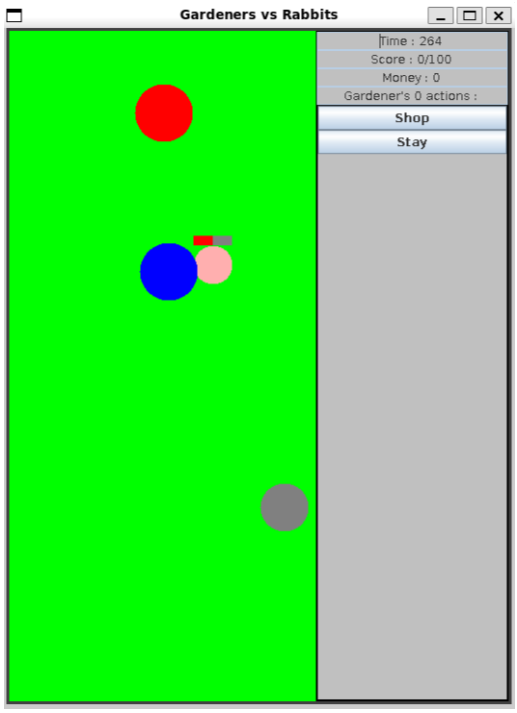
L'autre cas est si le temps est finit, alors on vérifie que l'objectif de score est atteint dans Round_initializer et on appelle la méthode win de GameInterface.

6. Résultat :

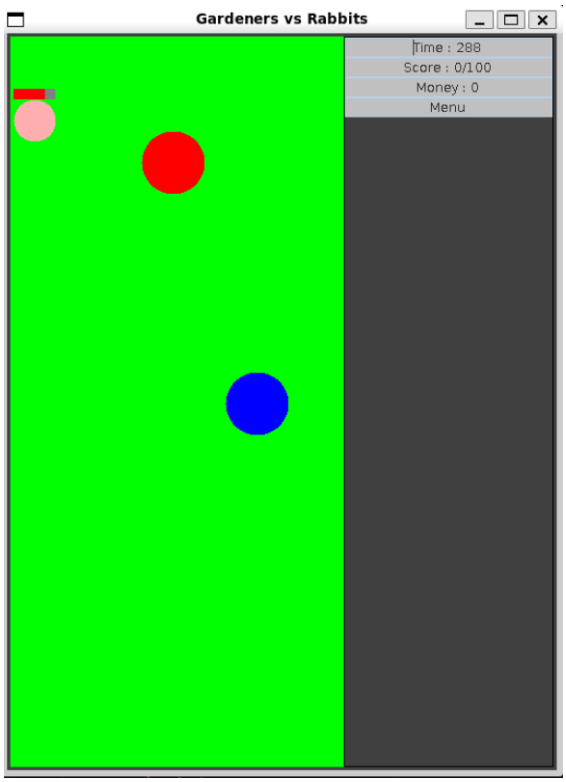
Nous avons créé un menu d'entrée, pour accéder au jeu il suffit d'appuer sur Play:



Ensuite nous arrivons sur l'interface du jeu sur l'image ci-dessous les jardiniers sont les points rouge et bleu, les lapins en gris et une plante en rose (la couleur de la plante dépend de son espèce), nous avons l'affichage du temps restant, du score avec l'objectif de score ainsi que l'argent possédé. Nous pouvons aussi voir le menu des jardiniers (Pour faire apparaître le menu du jardinier il suffit d'appuyer sur le jardinier en question) dans lequel on peut accéder au shop ou arrêter le mouvement du jardinier, on peut également voir la barre d'évolution de la pousse au-dessus de la plante.

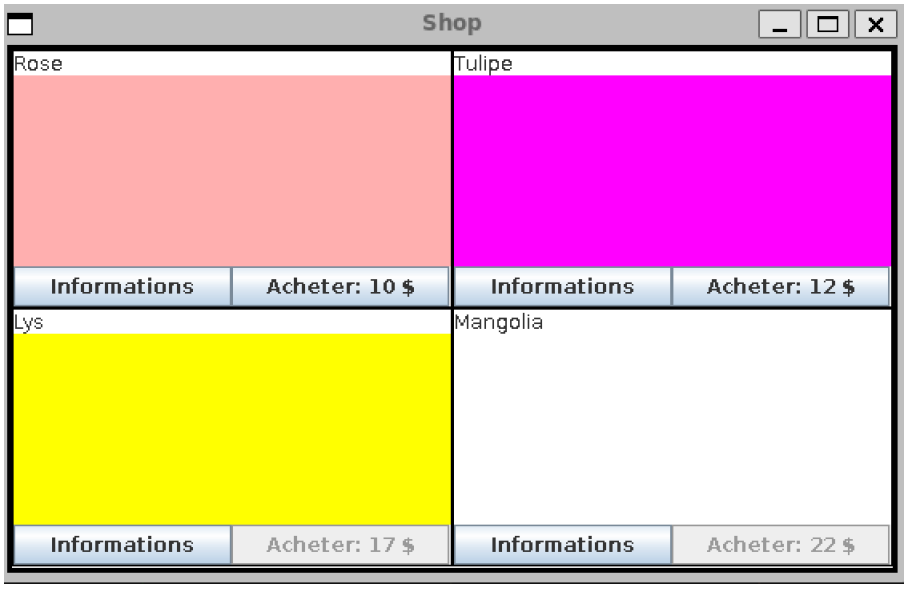


Lorsque le menu du jardinier n'est pas activé il n'apparait plus dans l'interface, pour le désactiver il suffit d'appuyer sur stay :

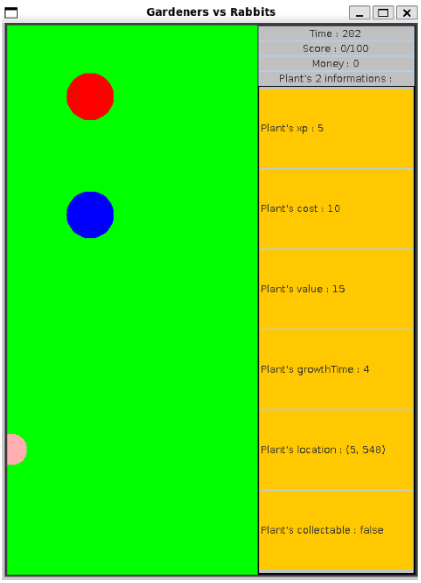


Quand on ouvre le shop une fenêtre que l'on peut voir ci-dessous s'ouvre, ici la partie vient de commencer nous n'avons donc pas assez d'argent pour acheter des plantes et donc les boutons sont grisés indiquant que l'on ne peut acheter (les boutons sont désactivés). On peut accéder à des informations sur les plantes en appuyant sur le bouton informations.

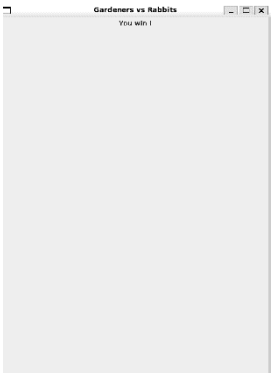
Lorsque l'on a assez d'argent les boutons sont réactivés et dégrisés. Quand on clique sur le bouton acheter et que l'on a assez d'argent, une plante de l'espèce choisie apparait a la position du jardinier sélectionné :



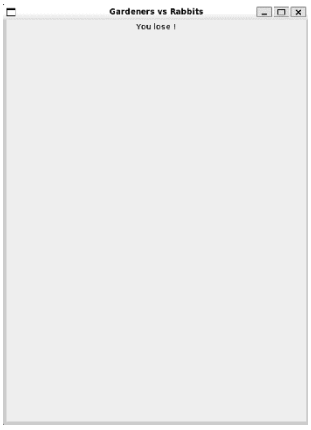
Chaque plante a un menu aquel on peut accéder en cliquant sur celle-ci, on y trouve le score rapporté par la plante, son cout, l'argent qu'elle rapporte, son temps de croissance, sa localisation et si elle peut-être cueillie:



Lorsque l'on gagne une nouvelle fenêtre s'ouvre indiquant que l'on a gagné :



Lorsque le temps imparti est fini et que l'on n'a pas compléter l'objectif de score une fenêtre indiquant que l'on a perdu apparaît :



7. Documentation utilisateur :

Le jeu nécessite un IDE pour lancer la class App qui contient la fonction main qui démarre le jeu.

9. Conclusion et perspectives :

On obtient un jeu fonctionnel avec un menu de départ et de fin, des conditions de victoire et de défaite, la possibilité de planter différentes variétés de fleurs et de les ramasser avec des jardiniers. Des lapins apparaissent au bord de la carte et des plantes apparaissent de manière aléatoire sur la carte.

On a rencontré des difficultés pour intégrer le code de chacun au début, chacun codait d'une manière différente avec une implémentation qui était souvent incompatible. On a pu surmonter cela en prenant le temps de l'adapter et de se fixer une manière d'implémenter le jeu. On a par la suite rencontré des problèmes d'organisations, certaines fonctionnalités dépendaient d'autres et n'avaient pas encore été implémenté. Il a fallu réallouer le travail de chacun.

On peut améliorer le rendu visuel avec des images, implémenter l'ajout de jardinier, d'autres niveaux, des caractéristiques pour les lapins ou encore des temps de cueillettes et pour planter.