Projet PCII : Documentation

Jessy FALLAVIER - Maxime NANDA OMBALA - Amadou POUYE

Introduction :

Nous voulions réaliser un jeu vidéo inspiré des jeux de stratégie en temps réel, avec vue du dessus. Avec les contraintes suivantes.

2 types unités se déplacent sur la carte : les scientifiques et les aliens, chacune ayant leur temporalité.

Les scientifiques doivent nourrir la population en leur fabriquant des rations à partir de fruits. Cependant, les aliens veulent empêcher ce processus d'aboutir.

Le joueur peut cliquer sur les unités pour interagir et voir leurs caractéristiques via un panneau de contrôle.

Le joueur peut déplacer les scientifiques, leur faire planter des fruits ou bien fabriquer des rations. Les aliens restent éloignés des scientifiques.

Fabriquer des rations rapporte des crédits, qui permettent d'acheter de nouvelles graines de fruits ou de former de nouveaux scientifiques. Il y a différents types de fruits rapportant chacun, un certain nombre de crédits.

Les aliens sont là pour nous gêner en mangeant les plantes, pendant un temps, avant de s'en lasser.

Le but du jeu est de récolter un certain nombre de crédit. Il y a un temps imparti pour atteindre ce but.

Analyse :

Analyse Globale :

Fonctionnalité	Difficulté	Priorité
Modélisation des fruits	Facile	1
Modélisation des unités	Facile	1
Affichage des fruits (liste)	Facile	2
Affichage des fruits (carte)	Facile	2
Affichage des unités	Facile	
Rayon d'action de l'unité sélectionnée	Facile	3
Gain	Moyen	3
Interactions Unité-Fruits	Moyen	2
Affichage map	Facile	1
Affichage panneau de contrôle	Normale	1
Interaction unité-grille-contrôleur	Difficile	1
Effet des boutons	Difficile	1
Déplacement d'une unité	Normal	1
Recrutement d'une unité	Normal	2
Fenêtre graphique	Facile	1
Aliens	Difficile	4
Affichage des Métriques	Normal	2
Affichage des informations au clic	Difficile	4
Ajout automatique de superfruit	Facile	3
Temps	Facile	2
Détection de fin de partie	Normal	3

Plus prioritaire = 1 ; Moins prioritaire = 4 ; **Voir l'analyse détaillée**

Analyse détaillée :

Certaines fonctionnalités ont des sous-fonctionnalités :

Fonctionnalité	Sous-fonctionnalité
Modélisation des fruits	Création d'un fruit
	Avancement de la vie d'un fruit
Affichage des fruits (liste)	Affichage des fruits dans une fenêtre de la zone d'action
	Redessinage de la fenêtre
Affichage des unités	Affichage des unités dans une fenêtre Redessin de la fenêtre
Gain	Modélisation d'une banque
	Perte de d'argent quand on plante
	Récolte d'argent quand on récolte
Interactions Unité-Fruits	Planter des fruits dans la zone d'action
	Récolter des fruits dans la zone d'action
	Impossibilité de planter si fruit pourri
Affichage	Fenêtre graphique Grid Layout
Effet des boutons	Grid Layout Déplacement S'arrêter sur place
Interaction unité-grille-contrôleur	MouseListener ActionListener Clic sur Unite active boutons Unite en déplacement active bouton stop Clic sur la map désactive les boutons
Déplacement d'une unité	MouseListener Thread Clic sur Unité la sélectionne Clic sur le bouton "se déplacer" autorise un déplacement Clic sur une zone de la map déplace l'unité
Recrutement d'une unité	ActionListener Clic sur le bouton "recruter une unite" Prix de recrutement
Aliens	Création d'un Alien Ajout automatique d'un alien

Plan de développement :

Temps de travail estimé pour chaque fonctionnalité :

Fonctionnalité	Temps estimé
Modélisation des fruits	45 min
Affichage des fruits	1h 30 min
Rayon d'action de l'unité sélectionnée	30 min
Interactions Unité-Fruits	1h 30 min
Gain	45 min
Modélisation des unités	15 min
Affichage de la fenêtre graphique	5 min
Affichage de la map	5 min
Affichage de la grille – panneau de contrôle	30 min
Interaction unité-contrôleur-grille	180 min
Déplacement d'une unité	80 min
Aliens	2h
Affichage Métriques	45 min
Affichage Information au Clic	45 min
Ajout automatique de superfruit	30 min
Temps	30 min
Détection de Fin de Partie	30 min

Autres tâches liées au projet :

Tâche	Temps estimé
Création d'un Git pour la fonctionnalité le projet	15 min

Diagramme de Gantt :

Fonctionnalité	S5	S6	S7	S8	S9	S10
Modélisation des fruits	Jessy					
Affichage des fruits (liste)	Jessy					
Modélisation des unités	Amadou					
Affichage des unités dans une fenêtre	Amadou					
Redessin de la fenêtre des unités Unité	Amadou					
Sélection d'une Unite	Amadou					
Déplacement Unité	Amadou					
Rayon d'action de l'unité sélectionnée		Jessy				
Interactions Unité-Fruits		Jessy				
Interactions Unité-Fruits (Fruit pourri)			Jessy			
Gain			Jessy			
Temps			Jessy			
Détection Fin De Partie			Jessy			
Grille des Boutons	Maxime					
Clic sur bouton "se déplacer" autorise déplacements		Maxime et Amadou				
Activation Dynamique des Boutons			Maxime			
Recrutement d'une unité				Amadou		
Aliens					Amadou et Jessy	
Info au Clic					Maxime et Jessy	
SuperFruit						Maxime et Jessy

Conception Générale :

Nous avons utilisé une conception avec le modèle MVC.

Une classe Main permet d’instancier toutes les classes nécessaires au fonctionnement du jeu.

Conception Détaillée :

Fenêtre graphique – Affichage global :

On utilise la classe JFrame afin d'afficher la fenêtre graphique de notre jeu.

On vient la spécifier avec un BorderLayout afin d'afficher à gauche (au "centre") la map du jeu où l'on peut voir les différents personnages ainsi que fruits s'afficher.

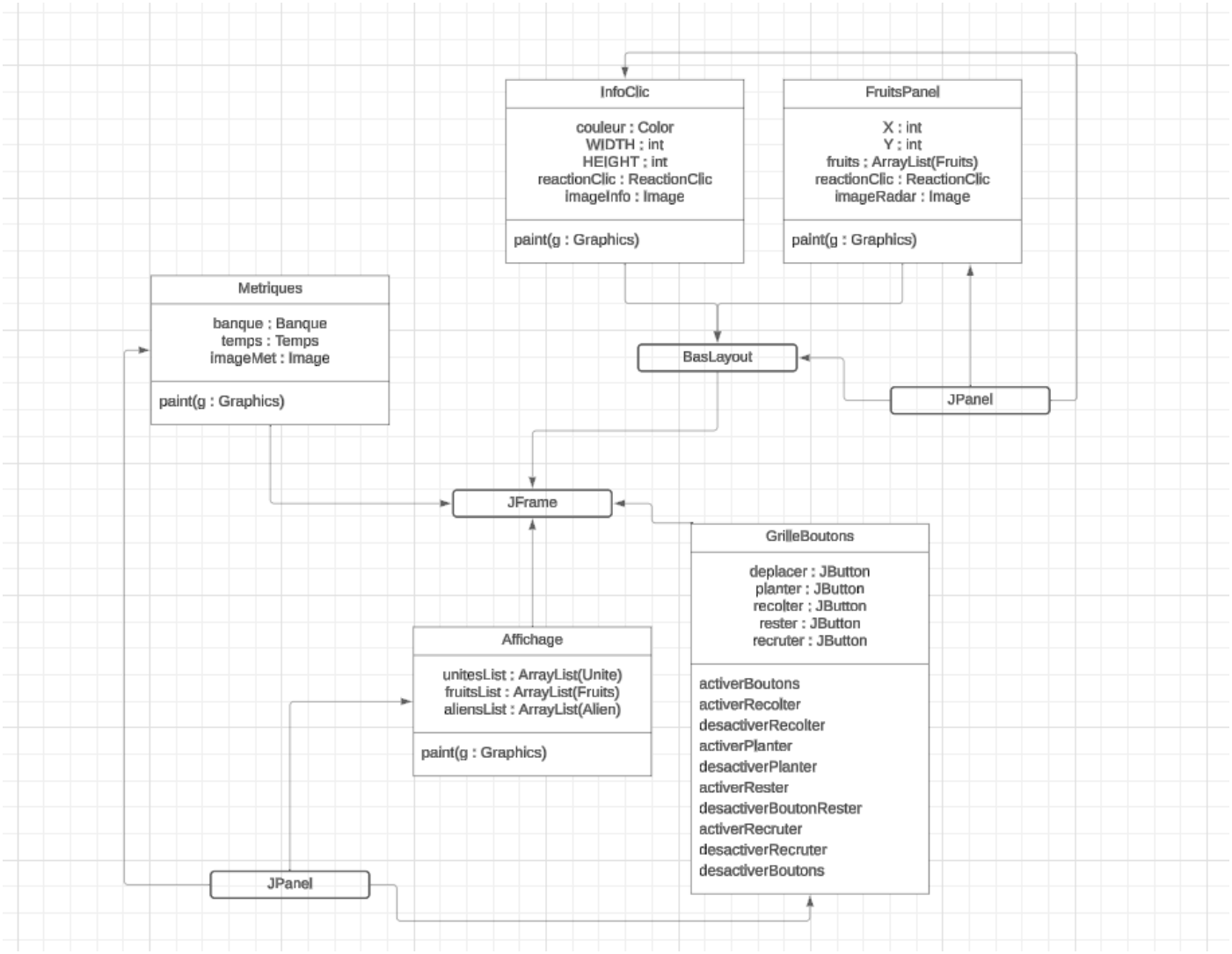
En haut le compteur de monnaie du joueur qui évolue au cours de la partie avec juste en dessous la barre de progression du temps qui diminue au cours du temps, changeant également de couleur pour être toujours plus impactante.

À droite, les différents boutons permettant d'interagir avec les unités et donc de jouer au jeu. Il y a plusieurs actions disponibles afin de progresser dans le jeu mais certaines sont disponibles sous conditions.

Le bas de la fenêtre graphique est lui-même divisé en deux via un autre BorderLayout.

En bas à gauche, la barre de progression des fruits se trouvant dans le rayon du scientifique cliqué en particulier. Cela veut dire que si les fruits sont dans le champ d'action d'un scientifique mais pas d'un autre leur barre de progression ne s'affichera que pour ce premier scientifique et non l'autre.

En bas à droite, les informations de l'unité ou du fruit cliqué en particulier.



Modélisation des fruits – Création d'un fruit :

On crée une classe Fruits représentant un fruit.

Un fruit possède une vie initiale aléatoire entre 5 et 15 générée à chaque création d'une nouvelle instance.

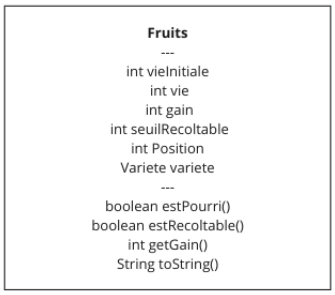
Un fruit possède une vie, celle-ci à la valeur de la vie initiale au début, plus elle diminue. Si la vie du fruit atteint 0, le fruit est pourri.

Un fruit possède un gain aléatoire entre 1 et 10 générée à chaque création d'une nouvelle instance. Ce gain est de 0 si le fruit est pourri.

Un fruit possède un seuil de récoltabilité correspondant à la moitié de sa vie initiale. Si ce seuil est atteint, le fruit est récoltable.

On surcharge la méthode toString pour pouvoir avoir un affichage console rudimentaire.

Un fruit possède une position et une variété

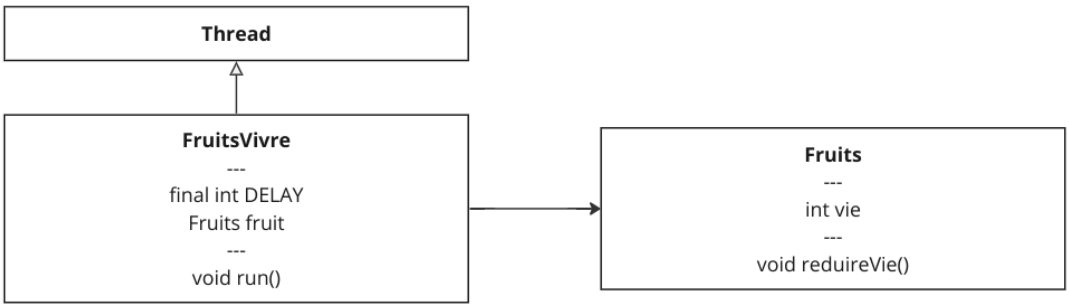


Modélisation des fruits – Avancement de la vie d'un fruit :

On crée une classe FruitVivre. Cette classe est un Thread

Cette classe possède un attribut fruit représentant le fruit dont la vie va diminuer.

Cette classe possède un attribut final DELAY. Tous les DELAY ms, cette classe va appeler la méthode reduireVie() de fruit pour décrémenter la vie du fruit de 1.

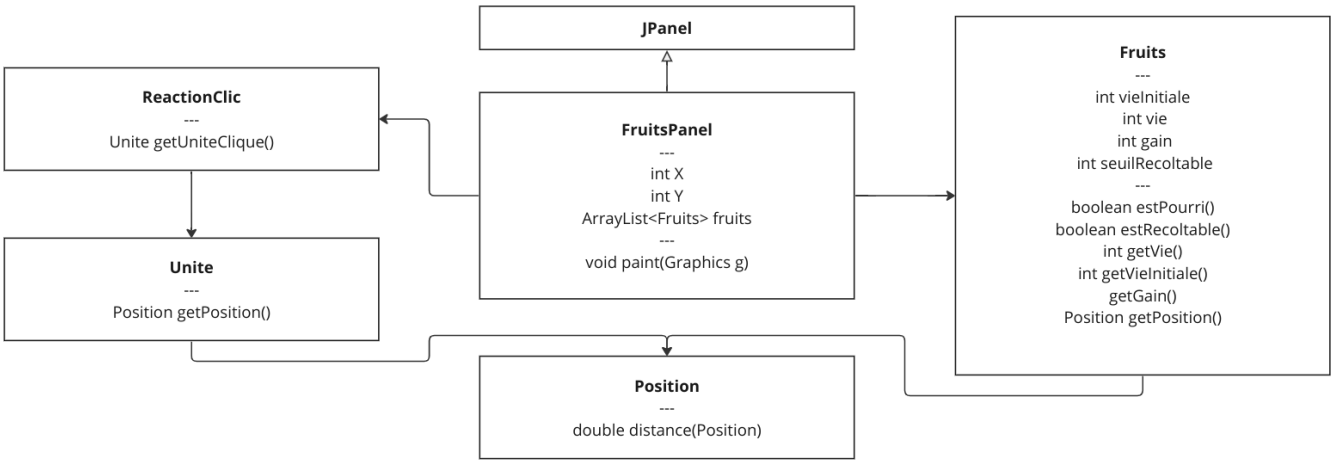


Affichage des fruits (liste) – Affichage des fruits dans une fenêtre :

On crée une classe FruitsPanel étendant JPanel de dimensions X et Y.

L'affiche des fruits suit un algorithme :

- Pour tous les fruits :
    - On vérifie que le fruit est dans la zone d'action de l'unité cliquée (getUniteClique de ReactionClic, distance de Position, < Unite.RAYONACTION), si oui :
        - Si le fruit est pourri, on met un fond rouge, sinon noir.
        - On affiche la vie restante, tel une barre de progression, en vert si c'est récoltable, en orange sinon.
        - On affiche un texte informatif avec le gain et le statut du fruit (pourri, en pousse, récoltable).

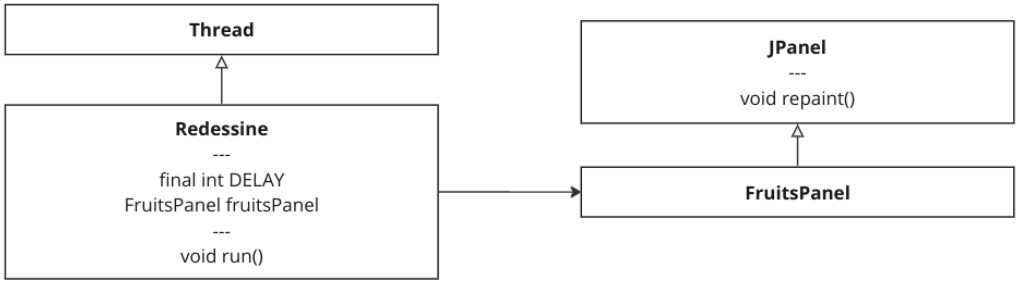


Affichage des fruits (liste) – Redessinage de la fenêtre

On crée une classe Redessinne. Cette classe est un Thread

Cette classe possède un attribut fruitsPanel représentant le JPanel à rafraichir.

Cette classe possède un attribut final DELAY. Tous les DELAY ms, cette classe va appeler la méthode repaint() de fruitsPanel pour le rafraichir.

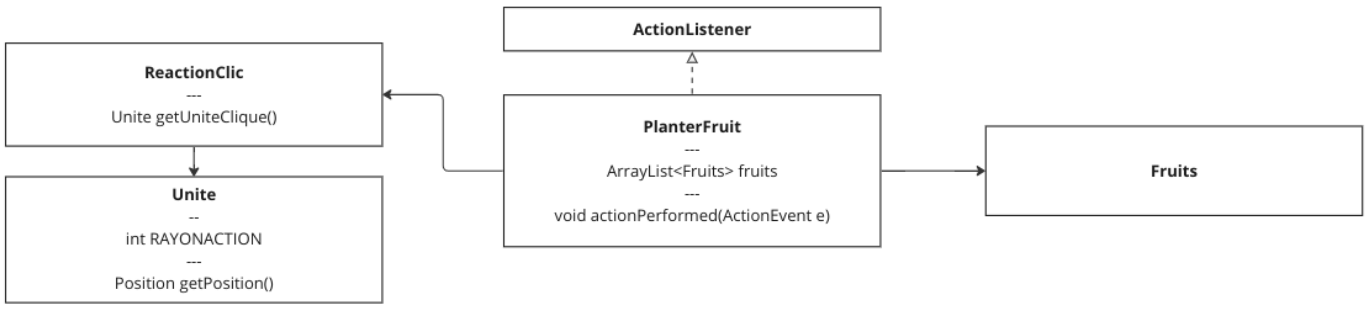


Interactions Unité-Fruits – Planter un nouveau fruit dans la zone d'action :

On crée une classe PlanterFruit qui implémente un ActionListener.

Sa méthode actionPerformed va ajouter une nouvelle instance de fruit dans à la liste des fruits.

On va planter le fruit dans la zone d'action de l'unité cliquée (reactionClic.getUniteClique()), créer donc une position aléatoire pour le fruit mais dans le rayon d'action (Unite.RAYONACTION)

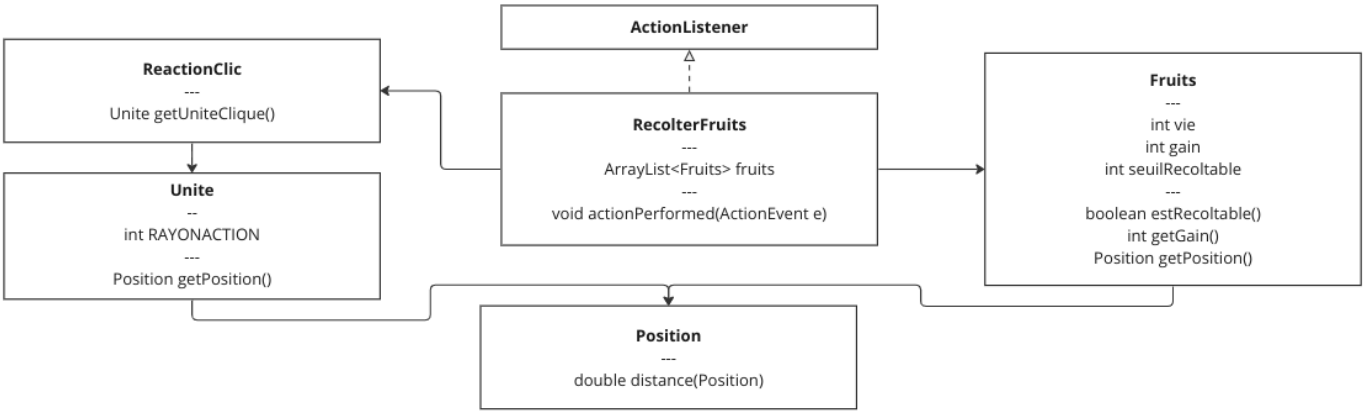


Interactions Unité-Fruits – Récolter les fruits dans la zone d'action :

On crée une classe RecolterFruits qui implémente un ActionListener.

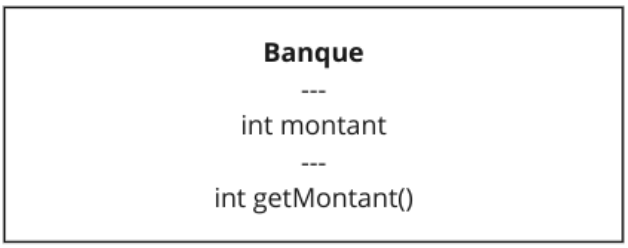
Sa méthode actionPerformed va itérer sur tous les fruits :

- Si le fruit est dans le rayon d'action de l'unité cliquée :
  - Si le fruit est récoltable
    - On l'enlève de la liste des fruits
    - On affiche son gain



Gain - Modélisation d'une banque

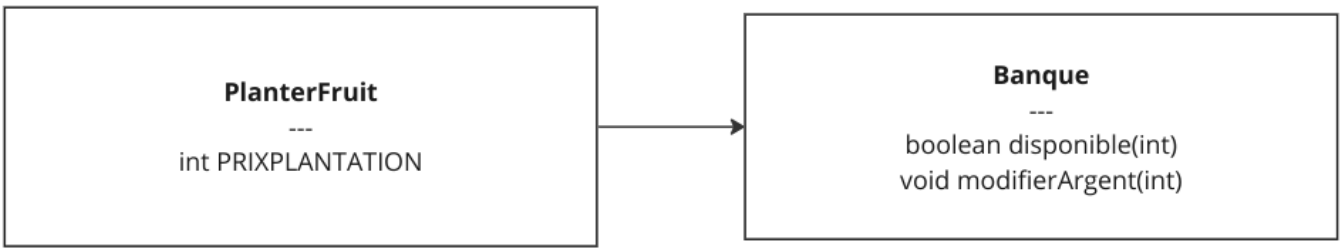
On crée une classe Banque qui contient un attribut argent initialisé à 10.



Gain - Perte de d'argent quand on plante

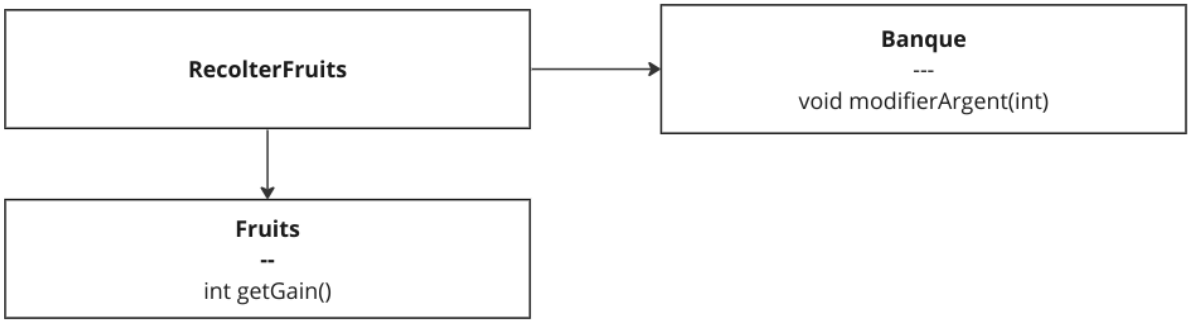
Quand on plante, on vérifie que l'argent est disponible grâce à la méthode disponible (int montant) de la banque.

Si c'est disponible, on utilise la méthode modifierArgent(int montant) avec comme argument la constante FruitPlanter. PRIXPLANTATION . La plante sera donc payée.



Gain - Récolte d'argent quand on récolte

Quand on récolte, on crédite la banque grâce à la méthode modifierArgent(int montant) de la banque, avec le montant de getGain du fruit. Si le fruit est pourri, il ne gagnera rien.



Modélisation des Unités

-On crée une classe Unité représentant les scientifiques qui hérite de Entite

-Un scientifique possède un identifiant unique, généré à partir de 1 à chaque création d'une nouvelle instance

-Un scientifique a une ancienne et une nouvelle position définie par une classe Position

-La classe Position défini une abscisse (x) et une ordonnée (y)

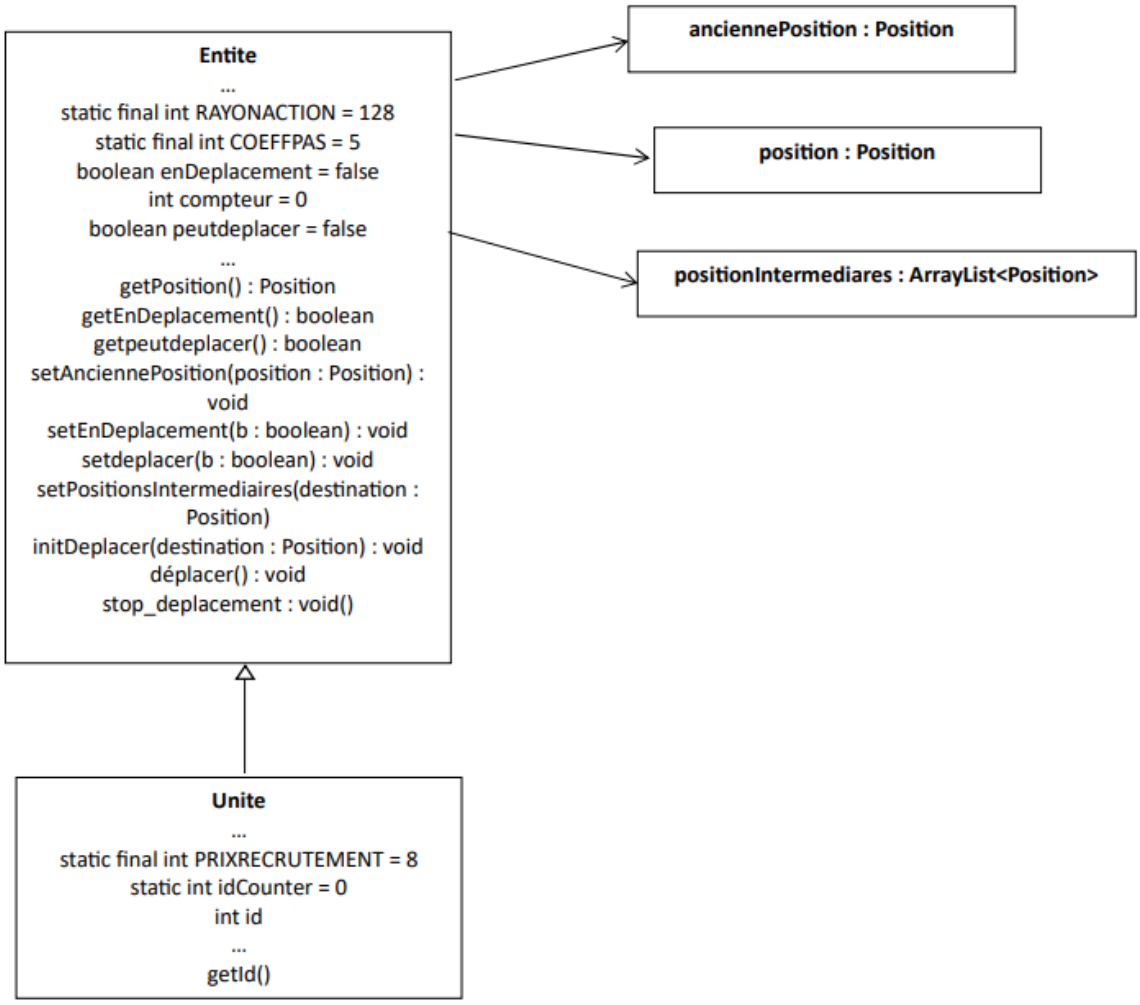
-Un scientifique peut être en déplacement ou non

-Un scientifique peut être déplaçable ou non

-Un scientifique a des positions intermédiaires atteignables via un compteur

-On détaillera certaines méthodes en temps voulu

-Diagramme de classe :



Affichage Unités – Affichage des unités dans une fenêtre

-On crée une classe Affichage étendant JPanel de dimensions longueur et largeur

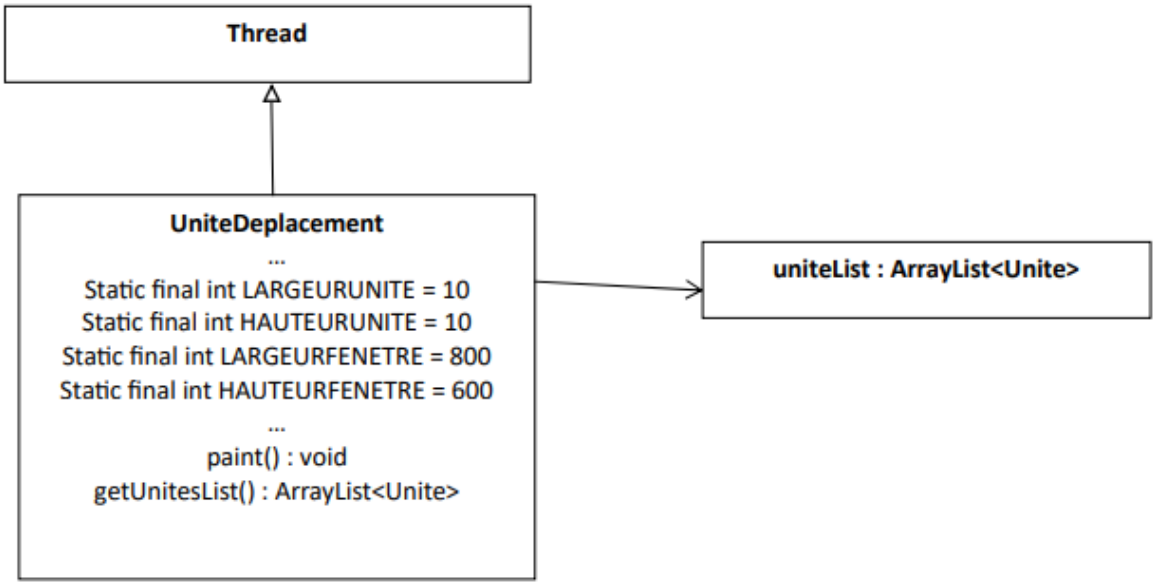
-Elle possède une liste d'unité

-L'affichage des unités suit un algorithme :

Pour chaque unité de la liste unité :

Afficher l'unité à la position qui lui correspond

-Diagramme de classe :



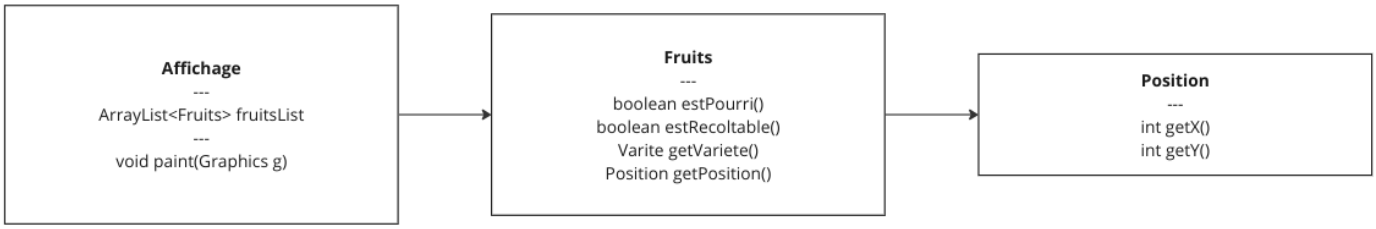
Rayon d'action de l'unité sélectionnée :

On utilise la méthode getUniteClique de reactionClic, lors de l'itération sur toutes les unités, dès que l'on arrive sur l'unité cliquée, on affiche un rayon de Unite.RAYONACTION autour de l'unité.

Affichage des fruits (carte) :

Pour chaque fruit de la liste des fruits, on affiche le fruit à ses coordonnées.

Si le fruit est en pousse, on affiche un sachet, si on affiche selon sa variété. Si le fruit est pourri, on l'affiche en noir et blanc.

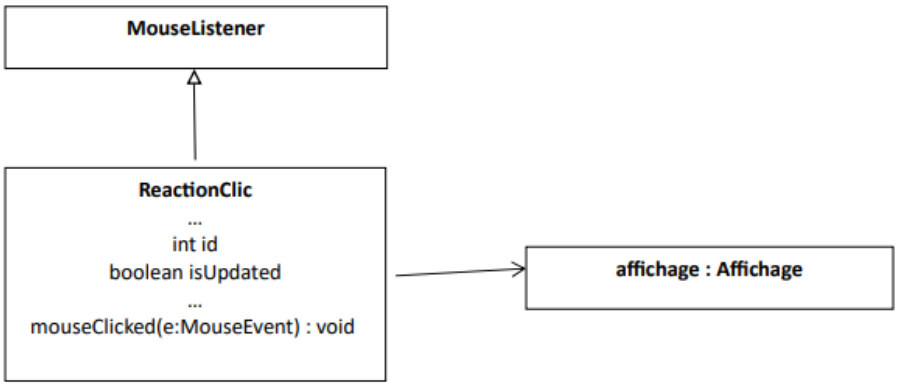


Sélection d'un scientifique

-On créer une classe ReactionClic qui implémente MouseListener

-On enregistre l'id du scientifique après un clic : Si la position du clic correspond à celle d'une unité dans un rayon de 10, on a cliqué sur l'unité

-Diagramme de classe :



Déplacement d'un scientifique

-On utilise à nouveau la classe MouseListener

-Si le scientifique peut se déplacer (cf avec bouton se déplacer) alors on clique sur l'endroit où on veut le déplacer : On enregistre une nouvelle position pour le scientifique (si le clic n'est pas sur un scientifique) et on met à jour les positions intermédiaires entre l'ancienne et la nouvelle

Algo de setPositionIntermediares :

On met a zéro la liste des position intermédiaires qui précèdent la case correspondante au compteur

On calcule la distance entre la position courante et l'ancienne

On définit un nombre de pas intermédiaire

Pour chaque i allant de zéro à nbPas

On calcule une position correspondante a l'avancement d'un pas

On l'ajoute dans le tableau

-On crée une classe UniteDeplacement qui étend Thread : Elle possède une unité et appelle la méthode déplacer de l'Unité si celle-ci est en déplacement. Chaque scientifique possède son propre thread

Algo de la méthode déplacer

Si le compteur n'a pas atteint la taille max

On atteint la position à la case correspondant au compteur

On incrémente le compteur

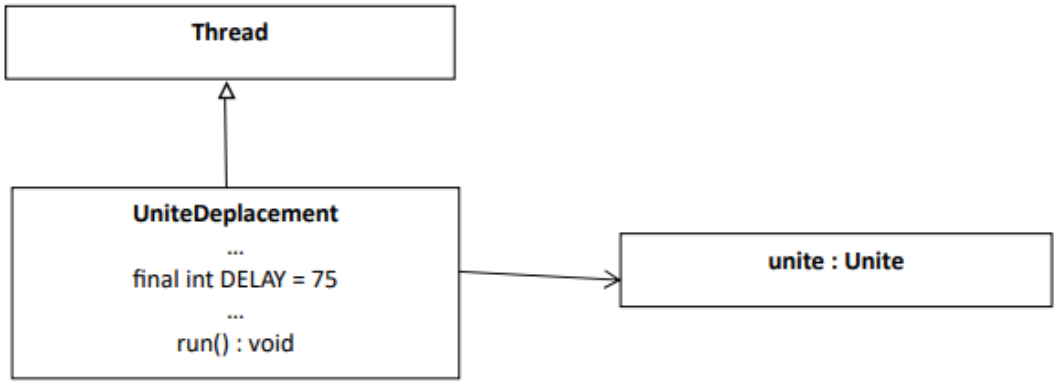
Sinon

L'unité n'est plus en déplacement

On remet le compteur à zéro

-Chaque scientifique aura son propre thread, directement initialisé dans le constructeur, ce qui fait qu'on pourra les déplacer simultanément

-Diagramme de classe :



Recrutement d’un scientifique

-On utilise à nouveau la classe MouseListener

-On créer une classe UniteRecrutement qui implémente ActionListener.

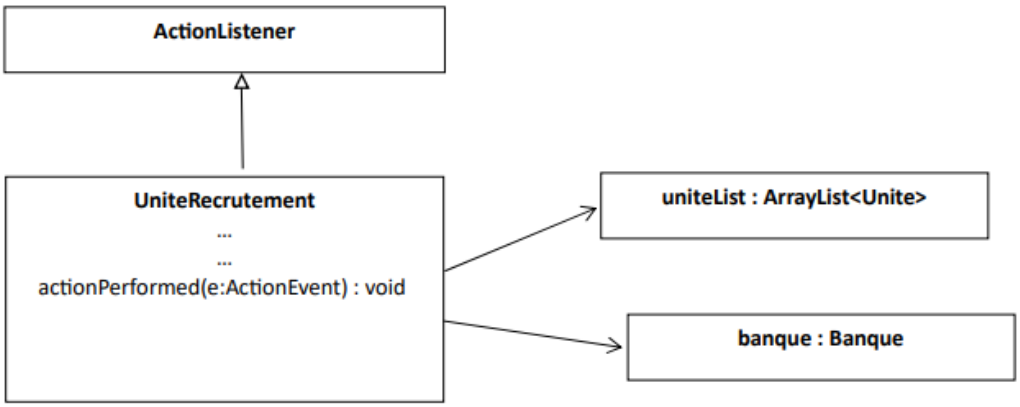
-Elle permet l’implémentation d’un bouton cliquable “recrutement d’une unite” : On la passe en paramètre de celui-ci dans la classe GrilleBoutons (cf partie boutons).

-Elle prend en paramètre une instance de la classe Banque pour consulter le solde.

-Recruter un scientifique coute 8 dollars, ce qui correspond à la constante Unite.RECRUTEMENT (donc de la classe Unite).

-Elle prend aussi la liste des unités en paramètre pour en générer une nouvelle si les conditions sont réunies (clic + solde)

-Diagramme de classe :



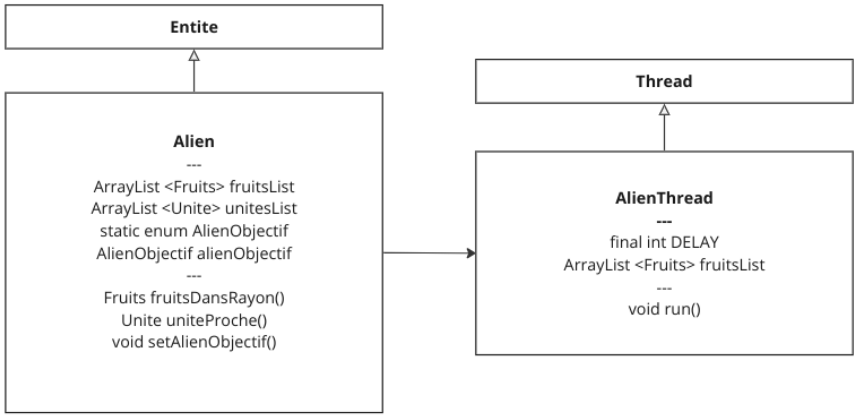
Alien - Création d'un Alien

On crée un Alien qui étend la classe Entité. Cette dernière contient des méthodes spécifiques au déplacement, comme dans Unite.

Un Alien a un objectif, il peut se balader, foncer sur un fruit ou être en fuite.

Grâce à la classe AlienThread qui étend Thread, on peut donner un comportement à l'Alien. Il suit un algorithme :

- Si l'alien est très proche d'un fruit, il peut l'empoisonner.
- Si l'alien ne se déplace pas et qu'il n'y a pas de fruit proche alors il va choisir une position aléatoire sur la carte et y aller.
- Si l'alien ne fonce pas sur un fruit mais qu'il y en a un proche, alors il va foncer sur lui.
- S'il y a une unité proche de lui, il va fuire 2x plus rapidement dans la direction opposée à lui.

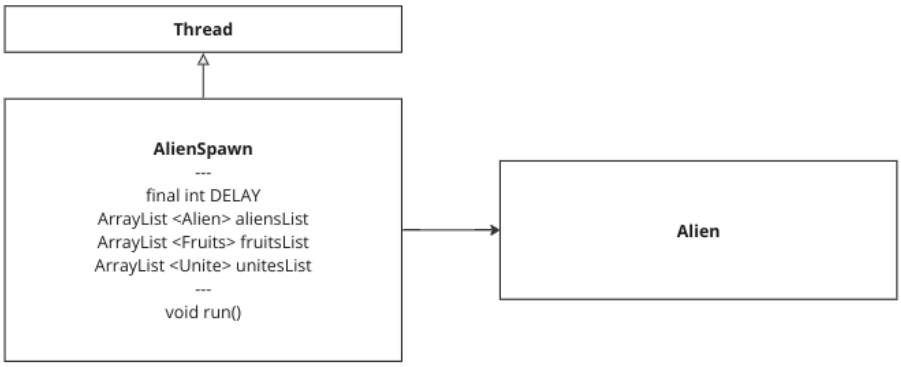


Alien – Ajout automatique d'un Alien

On crée une classe AlienSpawn qui étend un Thread.

Tous les DELAY ms, elle va ajouter un Alien a la liste des Aliens.

Ici, la liste des Unités et la liste des Fruits servent à pouvoir instancier un Alien.

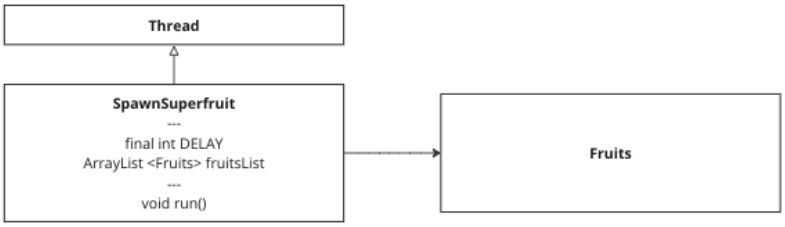


Ajout automatique d'un Superfruit

On va régulièrement ajouter un fruit de variété SuperFruit.

Ce fruit spécial donne 20 de gain et est immédiatement récoltable.

Ce fruit est ajouté à la liste des fruits

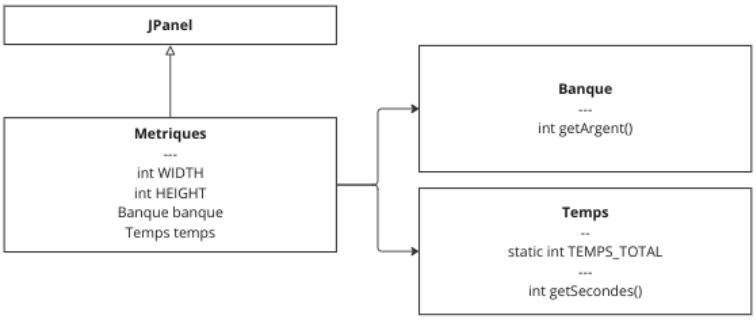


Affichage des Métriques

On crée une classe Métriques qui va étendre un JPanel.

Cette classe prends la Banque et le Temps en attribut.

Grâce aux méthode getArgent et getSecondes de ses classes, on va pouvoir obtenir l'argent et les secondes restantes.
Pour afficher le temps sous la forme d'une barre de progression, on utilise la constante Temps.TEMPS_TOTAL.



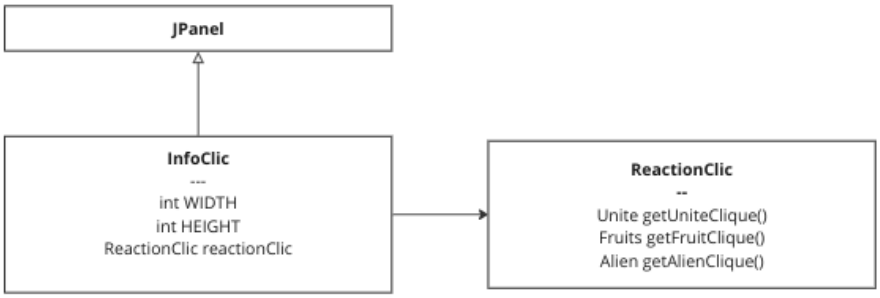
Affichage des informations au clic

On crée une classe InfoClic qui va étendre JPanel.

On utilise ReactionClic pour pouvoir obtenir l'unite ou le fruit ou l'alien cliqué.

Une fois obtenu, il nous suffira d'afficher les informations relatives à chaque classe.

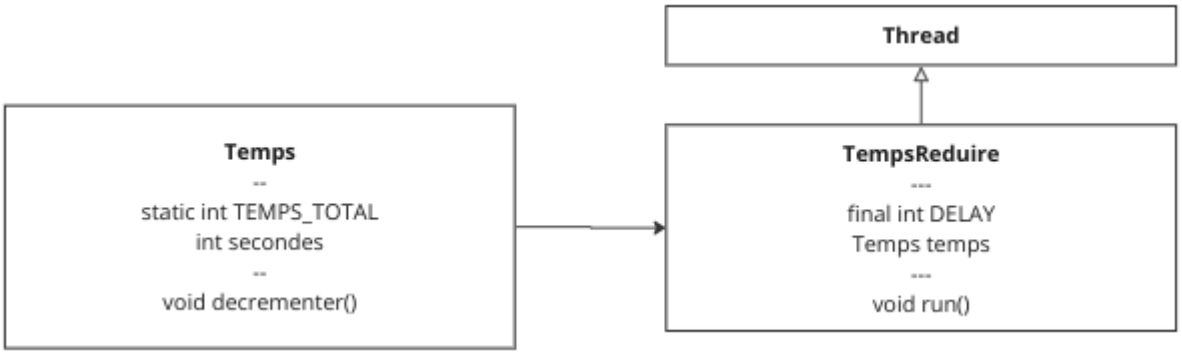
L'ordre de priorité lors d'un clic est fruit > alien > unité.



Temps - Création et Réduction

Pour le temps, on crée une classe Temps dans le modèle avec un attribut entier secondes.

Un Thread lui est associé, il permet de réduire le temps toutes les 1 secondes grâce à la méthode decrementer() de Temps.



Détection de fin de partie

On crée une classe DetectionFinPartie qui va étendre Thread.

Cette classe a comme paramètre la banque et le temps.

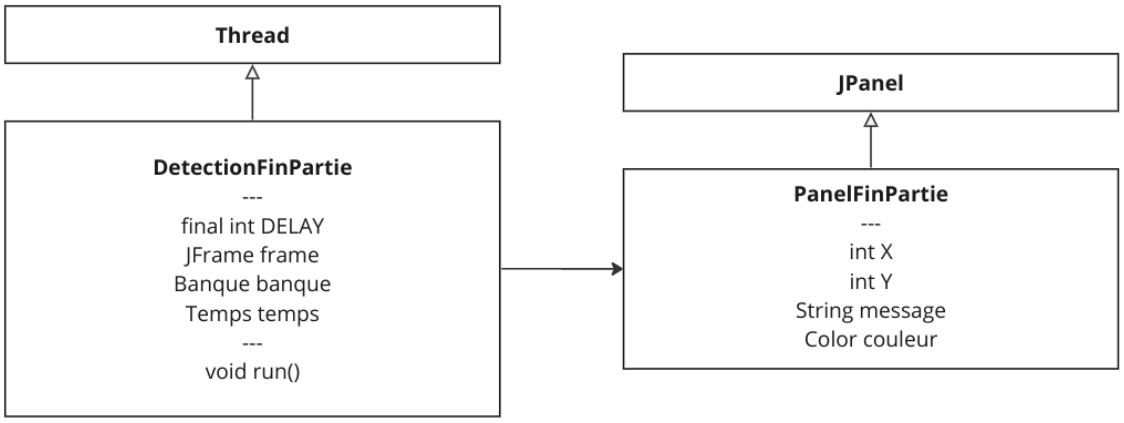
On va vérifier si la partie est finie, si oui, on instancie la classe PanelFinPartie avec un message et une couleur. On aura plus qu'à l'afficher dans la fenêtre.

Le message et la couleur changent :

Si le temps est écoulé on affiche le score avec un écran bleu

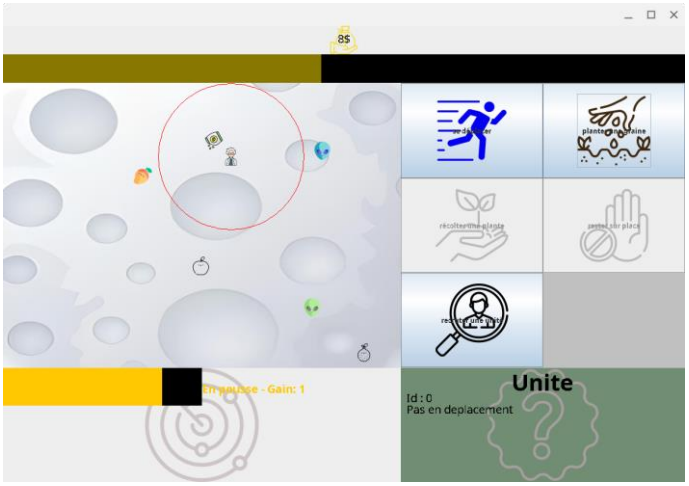
Si on fait faillite on affiche Banqueroute avec un écran noir

Si on atteint le seuil de richesse, on affiche richesse avec un écran vert.

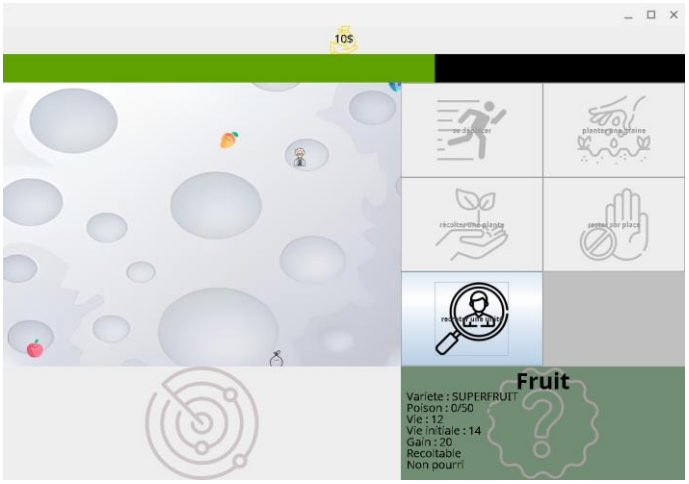


Résultats

Interface lors du lancement du jeu, un alien fonce et empoisonne un fruit



Un Alien est ici en fuite, un fruit est dans le rayon, deux aliens sont présents



On a ici des informations sur le superfruit dans la fenêtre d'information au clic.

Documentation utilisateur

Prérequis : Java avec IDE comme IntelliJ ou Visual Studio Code

Mode d'emploi : Importez le projet dans votre IDE, sélectionnez la classe Main à la racine du projet puis cliquez sur l'icône "Run".

- Observez la barre de temps en haut, ainsi que votre solde.
- Vous démarrez avec un scientifique, deux fruits poussent un alien commence à les attaquer.
- Cliquez sur un scientifique pour le sélectionner : les boutons d'actions s'activent sur la droite.

  -Se déplacer : cliquez sur le bouton "se déplacer" puis sur l'endroit voulu sur la carte.

  -Planter une graine : cliquez sur le bouton "planter une graine" et un fruit se mettra alors à pousser dans la zone d'action du scientifique. Attention, le bouton sera grisé si vous n'avez pas assez d'argent ou bien s'il y a déjà 3 fruits (en pousse ou non) dans la zone d'action. Observez votre solde diminuer.

  -Récolter une plante : cliquez sur le bouton "récolter une plante" si des fruits se trouvent dans votre zone d'action et que vous souhaitez les récolter. Chaque fruit vous rapportera plus ou moins d'argent, le saint-graal étant le super fruit. Observez votre solde augmenter. (Attention, si vous récoltez un fruit pourri, votre solde diminue : voir ci-après)

  -Rester sur place : cliquez sur le bouton "rester sur place" lorsque vous êtes en déplacement et que vous souhaitez interrompre le mouvement du scientifique. Ce dernier s'arrêtera donc à la position où il est au moment du clic.

  -Recruter une unité : cliquez sur le bouton "recruter une unité" et un scientifique sera généré à une position aléatoire sur la carte. Il disposera alors des mêmes capacités que l'originel. Attention, le bouton sera grisé si vous n'avez pas assez d'argent.
- Lorsqu'un fruit est dans votre zone d'action, vous pouvez observer son statut et son évolution en bas à gauche : orange s'il est en pousse, vert quand il est récoltable et rouge s'il est pourri.
- Cliquez sur un des éléments du jeu (fruit, alien ou scientifique) pour voir les informations correspondantes en bas à droite.
- Rapprochez-vous d'un alien et observez-le devenir bleu et fuir.
- Eloignez-vous en et il redeviendra vert.
- Eloignez-vous maintenant d'un fruit et observez le devenir rouge et venir attaquer : cliquez sur le fruit en question et constatez que la vie de celui-ci diminue.
- Gagnez la partie en atteignant un solde de 75 avant la fin du temps imparti.

Documentation développeur

La classe contenant la méthode main est la classe Main située dans le package main.

Nos principales constantes sont :

FruitPlanter. PRIXPLANTATION : Le prix pour planter un fruit

Entite.RAYONACTION : Le rayon d'action d'une entité

Entite. COEFFPAS : Plus on augmente ce coefficient, plus on ralenti l'entité

Fruits.seuilPoison : Seuil à partir duquel un fruit est considéré comme empoisonné

Temps. TEMPS_TOTAL : Le temps total disponible au lancement du jeu

Unite. PRIXRECRUTEMENT : Le prix pour recruter une unité

Affichage. TAILLEUNITE : La taille d'une unité dans l'affichage, s'applique aussi aux fruits et aux aliens

DetectionFinPartie. SEUILRICHESSE : La seuil à partir duquel on a gagné

Les DELAY dans les threads permettent de définir le delai d'attente entre chaque itération de la boucle du thread.

Les X/Y ou WIDTH / HEIGHT ou LARGEURFENETRE / HAUTEURFENETRE dans les JPanel permettent de définir la taille de ces panels. Ils suivent un ratio défini de Main (GLOBAL_RATIO_WIDTH, GLOBAL_RATIO_HEIGHT).

Conclusion et perspectives

Ce projet nous a permis de voir mettre en œuvre un jeu complet avec des Threads.

Ce projet nous a également permis de voir une nouvelle approche du travail de groupe.

Une des difficultés rencontrées était au début lors de l'intégration. Elle avait laissé des bouts de code inutiles. Nous avons donc créé un package poubelle avec nos anciennes classes pour pouvoir continuer à nettoyer et simplifier le code.

Nous pourrions améliorer le côté jeu en ajoutant plus d'imprévisibilité par exemple, ou augmenter l'immersion avec des musiques.

Nous avons déjà fait des premiers pas vers ce côté jeu en ajoutant une animation lors du changement d'argent et en ajoutant des aliens et des superfruits au fur et à mesures.