Licence Informatique – 2ème Année

Rapport de projet

PROJET de la simulation de l’abeille

Réaliser par :

Amine KECHIR

Rayane YEFSAH

Encadré par :

Jean-Luc Guérin

Année universitaire 2023-2024 je

SOMMAIRE

Introduction

Dans le cadre de notre seconde année en Informatique à l'UPJV, il nous est proposé un projet qui nous permettant de mettre en pratique nos connaissances et nos compétences au travers un projet orienté objet afin de réaliser un programme pour la simulation de la foule en JAVA. L'objectif de ce projet est de réaliser une application permettant de simuler le comportement de recherche de nourriture des abeilles mellifères. Les abeilles se répartissent en trois groupes: les abeilles employées, les observatrices et les éclaireuses : Pour communiquer, une abeille retourne à la ruche et effectue une danse indiquant les informations concernant une source.

Problématique

Comment modéliser les interactions entre les abeilles et leur environnement, y compris les sources de nourriture, la ruche et d'autres abeilles ?

Mission

Notre mission dans un premier temps est de créé ces classes pour modéliser différents types d'abeilles avec des comportements spécifiques :

1. La classe Ruche modélise une ruche dans un environnement simulé. Dans une simulation d'abeilles, la ruche est l'endroit où les abeilles stockent les meilleurs sources de nourriture qu'elles ont découvertes après avoir été contrôlées .
2. La classe Source représente les sources de nourriture dans l'environnement de simulation.
3. La classe Abeille est une classe abstraite qui sert de modèle de base pour les différents types d'abeilles de notre simulation. Ensuite nous avons les classes qui héritent de la classe abeille :

- La classe Éclaireuse représente un type spécifique d'abeille chargée de détecter les sources de nourriture.

- La classe Observatrice représente un autre type d'abeille chargé d'observer et de comparer les différentes sources de nourriture détectées.
- La classe Employée représente un type d'abeille qui récolte la nourriture.

Puis dans un deuxième temps , nous allons créer la classe de simulation et la fenêtre d’affichage des éléments :

- La classe Panneau permet la gestion de la simulation et de l'affichage des différents éléments du jeu. Elle contrôle également les interactions entre les abeilles et les sources de nourriture, ainsi que la fin du jeu lorsque toutes les sources sont collectées.
- La classe Fenetre qui représente la fenêtre principale de l'application. Elle est utilisée pour créer et afficher l'interface graphique du jeu d'abeilles.

Outils

1. Langage JAVA :

JAVA est un langage de programmation orienté objet créer par James Gosling et Patrick Naughton, employés de Sun Microsystems, avec le soutien de Bil Joy (cofondateur de Microsystems en 1982), présenté officiellement le 23 mai 1995 au SunWorld.

Développement technique et Méthodes

1. La stratégie prévisionnelle :

Ce projet étant personnel, notre première mission fut de définir nous-mêmes une stratégie prévisionnelle ainsi que les objectifs à atteindre. Bien entendu, cette stratégie a évoluée au cours du temps afin de satisfaire nos exigences, mais aussi les contraintes auxquelles nous avons fait face.

Tout d'abord,nous avons construit la classe **Abeille** qui une classe abstraite,
Elle contient les attributs communs à toutes les abeilles : les coordonnées x et y, le rayon, la couleur, et une référence à une Source.
Elle possède des méthodes abstraites **getposition**() et **dessinerAbeille**(Graphics g, int qualiteRec) qui doivent être implémentées par les classes filles.
Les méthodes **exploreSource**(int sourceX, int sourceY) et deplacerAbeille(int sourceX, int sourceY) permettent de déplacer l'abeille vers une source donnée.
Ensuite, on construit les classes Éclaireuse, Employée et Observatrice qui étendent de la classe Abeille.
D'abord, la classe **Éclaireuse**,

Elle a un attribut supplémentaire **aDetecteSource** qui indique si l'éclaireuse a détecté une source.
La méthode **marquerDetectionSource**() marque l'éclaireuse comme ayant détecté une source.
La méthode **dessinerAbeille**(Graphics g, int qualiteRec) dessine une éclaireuse avec des caractéristiques spécifiques (couleur, forme des ailes, tête, etc.).
La méthode **detecterCollision**(Source source) vérifie si l'éclaireuse a détecté une collision avec une source.

Observatrice :
Elle a un attribut **stockSourceObs** pour stocker les sources détectées.
La méthode **dessinerAbeille**(Graphics g, int qualiteRec) dessine une observatrice avec des caractéristiques spécifiques.
La méthode **detecterCollision**(Source source) doit être implémentée pour détecter les collisions avec une source.

La méthode **comparerSource**(ArrayList<Source> sourcesExistantes, Source nouvelleSource) compare la qualité d'une nouvelle source avec celles déjà enregistrées.

Employee :
Elle a un attribut qualiteEm pour stocker sa qualité.
La méthode **dessinerAbeille**(Graphics g, int qualiteRec) dessine un employee avec des caractéristiques spécifiques.
La méthode **detecterCollision**(Source source) vérifie s'il y a une collision avec une source.
La méthode **danser**() fait changer la couleur de l'employee de façon aléatoire.
Source :
Cette classe représente une source de nectar.
Elle a des attributs pour la qualité, les coordonnées et la couleur de la source.
La méthode **dessinerFleur**(Graphics g) dessine une représentation visuelle de la source.
La méthode **explorer**() marque la source comme étant explorée.

Source :
La classe Source représente une source de nectar récupéré par les abeilles .

Elle a des attributs :
x, y: Ces attributs définissent les coordonnées de la source sur le plan cartésien.
qualiteS: Indique la qualité de la source, c'est-à-dire sa richesse en nectar.
couleurS: La couleur de la source, utilisée pour la visualisation.
exploree: Un indicateur booléen pour savoir si la source a été explorée par une abeille.

Et les méthodes :
getqualite(): Renvoie la qualité de la source.
getColor(): Renvoie la couleur de la source.
dessinerFleur(Graphics g): Dessine visuellement la source sous forme de fleur en utilisant un objet Graphics.
getpositionSource(): Affiche les coordonnées de la source dans la console.
estExploree(): Renvoie true si la source a été explorée, sinon false.
explorer(): Marque la source comme explorée, modifiant ainsi la valeur de explorée à true.

Ruche :
Cette classe représente une ruche qui stocke les sources de nectar.
Elle a une liste d'objets Source stockés.
La méthode **dessinerRuche**(Graphics g, int qualiteS) dessine une représentation visuelle de la ruche.
La méthode **addSourceRuche**(Source meilleurSource) ajoute une nouvelle source à la ruche et met à jour la qualité de la ruche.

Ensuite, dans un deuxième temps nous avons réalisé la classe **Panneau** qui représente la zone de jeu où se déroule la simulation des abeilles et des sources de nectar.

Attributs statiques :

tileSize, SCREEN_ROW, SCREEN_COL, SCREEN_WIDTH, SCREEN_HEIGHT: Ces attributs définissent les dimensions de la grille de jeu ainsi que la taille de chaque tuile (en pixels). Ils sont statiques car ils sont partagés entre toutes les instances de la classe et ne changent pas pendant l'exécution.

Attributs d'instance :
r1: Une instance de la classe Ruche qui représente la ruche où sont stockées les sources de nectar.
e1, **Obs**: Instances respectivement des classes **Employee** et **Observatrice**, représentant une abeille employée et une abeille observatrice.
tabEclaireuse, **tabFleur**: Tableaux pour stocker respectivement les instances des classes **Eclaireuse** et **Source**.
indiceSourceActuelle: Un compteur utilisé pour suivre l'indice de la source actuellement explorée.
Constructeur :
Le constructeur initialise la taille et la couleur de fond du **panneau**, ainsi que les instances de la ruche, des abeilles et des sources en utilisant des valeurs aléatoires pour leurs positions et leurs couleurs.
Méthodes :
debutJeu(): Lance un nouveau thread pour commencer la simulation du jeu.
gameOver(): Affiche une boîte de dialogue "Game Over" et termine l'exécution du programme.
paintComponent(Graphics g): Méthode héritée de **JPanel** pour dessiner les composants graphiques du jeu, y compris les abeilles, les sources et la ruche.
run(): Méthode principale de la classe qui contient la logique de jeu. Elle exécute en boucle la simulation du comportement des abeilles et des sources, en vérifiant les collisions, les explorations et en mettant à jour l'affichage à chaque itération.

- Stratégie technique :

Au démarrage du programme, une instance de la classe **Panneau** est créée, initialisant ainsi la zone de jeu avec les éléments nécessaires, notamment les abeilles, les sources de nectar et la ruche. Les positions des abeilles et des sources de nectar sont générées aléatoirement. Au début de jeu

La méthode **debutJeu**() est appelée pour lancer la simulation du jeu en créant un nouveau thread.

Le **thread** exécute la méthode **run**() en boucle, gérant ainsi la logique du jeu. La méthode **run**() est exécutée dans une boucle **while**(true), ce qui signifie qu'elle s'exécutera en continu jusqu'à ce que le programme soit arrêté manuellement.

Vérification de l'exploration des sources de nectar :

Une variable booléenne **toutesExplorees** est initialisée à **true**. Cette variable sera utilisée pour vérifier si toutes les abeilles ont exploré les sources de nectar disponibles lors de cette itération de la boucle.

Une boucle for itère sur chaque abeille éclaireuse dans le tableau **tabEclaireuse**.

Pour chaque éclaireuse, si elle n'a pas encore détecté de source de nectar **(!this.tabEclaireuse[i].aDetecteSource),** elle explore la zone à la recherche de la source la plus proche en appelant la méthode **exploreSource()** avec les coordonnées de la source de nectar actuelle **(this.tabFleur[i].x et this.tabFleur[i].y).**

La position de la source de nectar actuelle est affichée en appelant la méthode **getpositionSource**() de la classe Source.

La position de l'éclaireuse est affichée en appelant la méthode **getposition**() de la classe **Eclaireuse**.

Si une collision est détectée entre l'éclaireuse et la source de nectar **(this.tabEclaireuse[i].detecterCollision(this.tabFleur[i])),** l'éclaireuse se déplace vers la ruche pour stocker le nectar en appelant **this.tabEclaireuse[i].deplacerAbeille(this.r1.x, this.r1.y)**, la source de nectar est marquée comme explorée **(this.tabFleur[i].explorer())** et l'éclaireuse marque la source comme détectée **(this.tabEclaireuse[i].marquerDetectionSource()).**

Si aucune collision n'est détectée, la variable **toutesExplorees** est mise à **false**, indiquant qu'au moins une abeille n'a pas encore exploré toutes les sources de nectar.

Passage à la source de nectar suivante :

Après que toutes les éclaireuses ont exploré les sources de nectar disponibles, la boucle vérifie si l'indice de la source actuelle est inférieur ou égal au nombre total de sources de nectar **(indiceSourceActuelle <= tabFleur.length).**

Si c'est le cas, l'employée (représentée par **e1**) explore la nouvelle source de nectar en appelant **e1.exploreSource(sourceActuelle.x, sourceActuelle.y)** et récupère sa qualité en appelant **sourceActuelle.getqualite().**

La position de l'employée est affichée en appelant **this.e1.getposition().**

La méthode **comparerSource()** de la classe **Observatrice** est appelée pour comparer la qualité de la nouvelle source de nectar avec celles déjà enregistrées dans la liste **Obs.stockSourceObs.**

Si une collision est détectée entre l'employée et la nouvelle source de nectar **(e1.detecterCollision(tabFleur[indiceSourceActuelle]))**, l'employée effectue une danse pour signaler la découverte. Si la source actuelle est la dernière dans la liste **(indiceSourceActuelle == tabFleur.length - 1)**, le jeu se termine et une boîte de dialogue "Game Over" est affichée, sinon l'indice de la source actuelle est incrémenté.

Rafraîchissement de l'affichage et pause :

À la fin de chaque itération de la boucle, la méthode **repaint()** est appelée pour rafraîchir l'affichage du jeu.

Un petit délai est ajouté avec **Thread.sleep(2)** pour éviter que la boucle ne s'exécute trop rapidement et utilise trop de ressources système.

2. La stratégie schématique:

Bilan

Au cours de ce projet, j'ai eu l'opportunité de mettre en pratique mes compétences en programmation orientée objet en travaillant sur plusieurs classes interdépendantes pour simuler un environnement d'abeilles. Voici un bilan spécifique par rapport aux classes implémentées :

- Classe Abeille :
  - J'ai appris à concevoir une classe abstraite qui sert de modèle de base pour différents types d'abeilles.
  - J'ai compris comment définir des méthodes abstraites pour les comportements spécifiques à chaque type d'abeille.
- Classes Éclaireuse, Observatrice et Employée :
  - J'ai développé une meilleure compréhension des principes d'héritage en créant des sous-classes spécifiques d'abeilles avec des comportements distincts.

- J'ai appris à implémenter des méthodes propres à chaque type d'abeille, telles que la détection de sources, la comparaison de sources et la récolte de nourriture.

- Classe Source :

  - J'ai exploré comment modéliser des éléments de l'environnement, comme des sources de nourriture, en utilisant des attributs pour représenter leurs caractéristiques.

  - J'ai compris comment utiliser des méthodes pour simuler des interactions avec les abeilles, telles que l'exploration et la détection.

- Classe Ruche et Panneau :

- J'ai acquis de l'expérience dans la création d'une interface graphique pour visualiser la simulation en utilisant les bibliothèques Swing et AWT.

- J'ai appris à gérer les éléments visuels et à les mettre à jour dynamiquement en fonction des interactions dans le jeu.

En résumé, ce projet m'a permis de renforcer mes compétences en programmation orientée objet tout en m'initiant à la conception d'une interface graphique pour une simulation. J'ai également réalisé l'importance d'une planification minutieuse et d'une collaboration efficace entre les différentes parties du projet pour obtenir un résultat fonctionnel et cohérent.

Conclusion

Ce projet de simulation d'abeilles a été une expérience enrichissante qui m'a permis de plonger dans le monde de la programmation orientée objet et d'explorer les concepts clés de ce paradigme. En développant différentes classes pour modéliser les comportements des abeilles et des éléments de leur environnement, j'ai pu approfondir mes connaissances en programmation Java.

Les classes que j'ai implémentées, telles que Abeille, Éclaireuse, Observatrice, Employée, Source, Ruche et Panneau, ont été conçues avec soin pour représenter fidèlement les différents aspects d'une colonie d'abeilles. Chaque classe a été conçue de manière modulaire et extensible, ce qui m'a permis d'ajouter de nouveaux comportements ou fonctionnalités facilement au fur et à mesure de l'évolution du projet.

L'utilisation de Swing pour créer une interface graphique interactive a ajouté une dimension supplémentaire à la simulation, permettant une visualisation en temps réel des interactions entre les abeilles, les sources de nourriture et la ruche. Cette composante visuelle a rendu le projet plus captivant et a facilité la compréhension des mécanismes sous-jacents de la simulation.

Malgré les succès rencontrés dans la conception et l'implémentation des classes, certains aspects pourraient être améliorés. Par exemple, la complexité des mouvements des abeilles pourrait être approfondie pour tenir compte des interactions entre les individus et des obstacles dans leur environnement. De plus, l'optimisation des performances pourrait être explorée pour garantir une exécution fluide, en particulier pour des simulations avec un grand nombre d'abeilles.

En conclusion, ce projet de simulation d'abeilles a été une expérience précieuse qui m'a permis d'approfondir mes compétences en programmation orientée objet tout en explorant le domaine fascinant du comportement des colonies d'abeilles. Il m'a également donné un aperçu des défis et des opportunités associés à la modélisation et à la simulation de systèmes complexes dans un environnement informatique.