

Programmation Concurrente et Interfaces Interactives

Janvier 2022

Projet Flappy Bird “revisit  ”

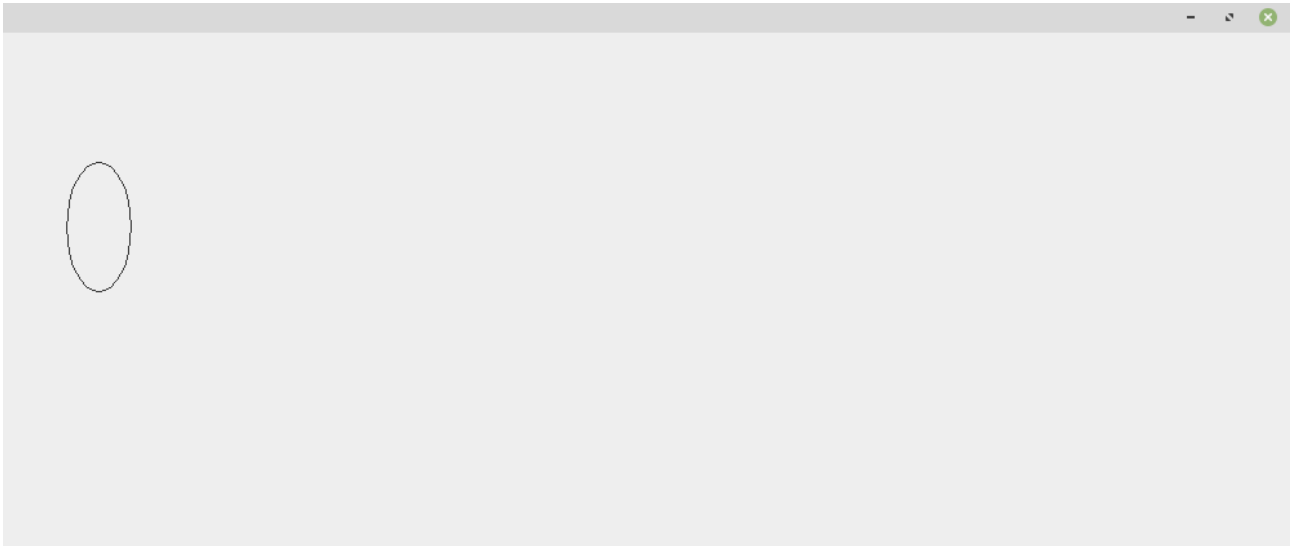
Universit   Paris-Saclay

Antonin Boyon
L3 Informatique parcours MIAGE

Introduction

Nous souhaitons réaliser un jeu inspiré de « Flappy-bird ». L'utilisateur doit aller le plus loin possible tout en gardant l'ovale (ou ici la fée) le long de la ligne. Si l'ovale sort de la ligne, une collision est comptée. L'utilisateur doit en faire le moins possible. Pour ce faire, l'utilisateur peut cliquer pour monter l'ovale, ou attendre que l'ovale redescende petit à petit.

Exemple d'interface du jeu



1ère interface du jeu.



Interface finale du jeu.

Analyse globale











Il y a 4 étapes majeures dans la réalisation de ce jeu :

1. Réalisation de l'interface graphique comportant l'ovale et la ligne brisée.
2. Changement de position de l'ovale à chaque clics de l'utilisateur
3. Défilement du jeu
4. Interaction entre l'ovale et la ligne brisée

Plan de développement

Liste des tâches :

- Analyse du problème (10 min)
- Acquisition des compétences en Swing (60min)
- Conception et réalisation d'une première maquette avec une fenêtre graphique et un ovale (30 min)
- Conception et réalisation du déplacement de l'ovale (60mn)
- Acquisition des compétences sur les Thread (30min)
- Défilement de la fenêtre vers la droite (120 min)
- Génération de la ligne brisée qui suit le défilement (90 min)
- Gestion des collisions entre l'ovale et la ligne brisée (120 min)
- Ajout d'élément de décors et sprite (90 min)
- Documentation du projet (120 min)

TÂCHES	Semaine 1	Semaine 2	Semaine 3
Analyse du problème			
Acquisition des compétences en Swing			
Conception et réalisation d'une première maquette avec une fenêtre graphique et un ovale			
Conception et réalisation du déplacement de l'ovale			
Acquisition des compétences sur les Thread			
Défilement de la fenêtre vers la droite			
Génération de la ligne brisée qui suit le défilement			
Gestion des collisions entre l'ovale et la ligne brisée			
Ajout d'élément de décors et sprite			
Documentation du projet			

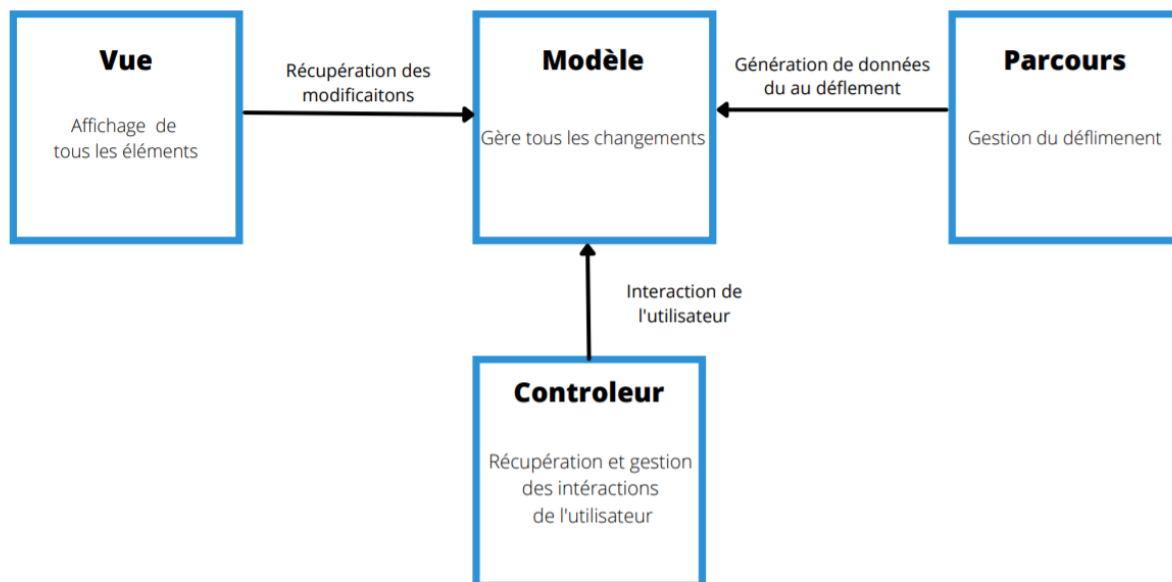
Conception générale

Nous avons adopté le modèle MVC pour le développement de notre interface graphique.

Le 'Controller' a pour rôle de gérer toutes les interactions de l'utilisateur avec l'application. Ici ce sont les cliques de souris permettant de faire monter l'ovale.

Le 'Model' (Classe Etat) gère tous les changements des composants de l'application. Dans notre cas, le modèle s'occupe de gérer la position de l'ovale et de la fonction permettant de modifier sa position (à chaque clique de l'ovale).

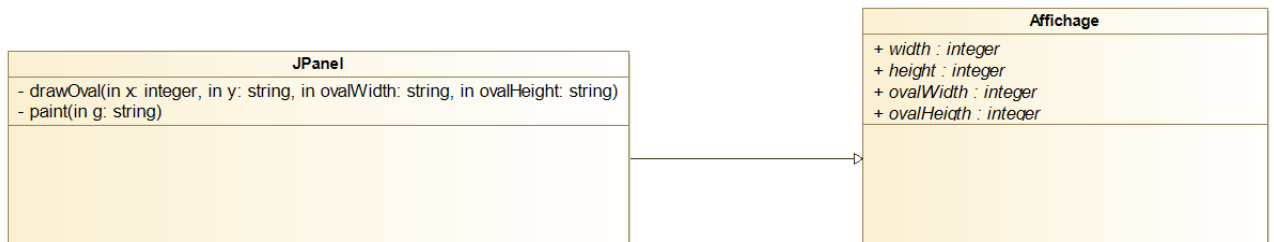
Enfin la classe Affichage ('View') permet d'afficher l'ovale tout au long de l'exécution de l'application.



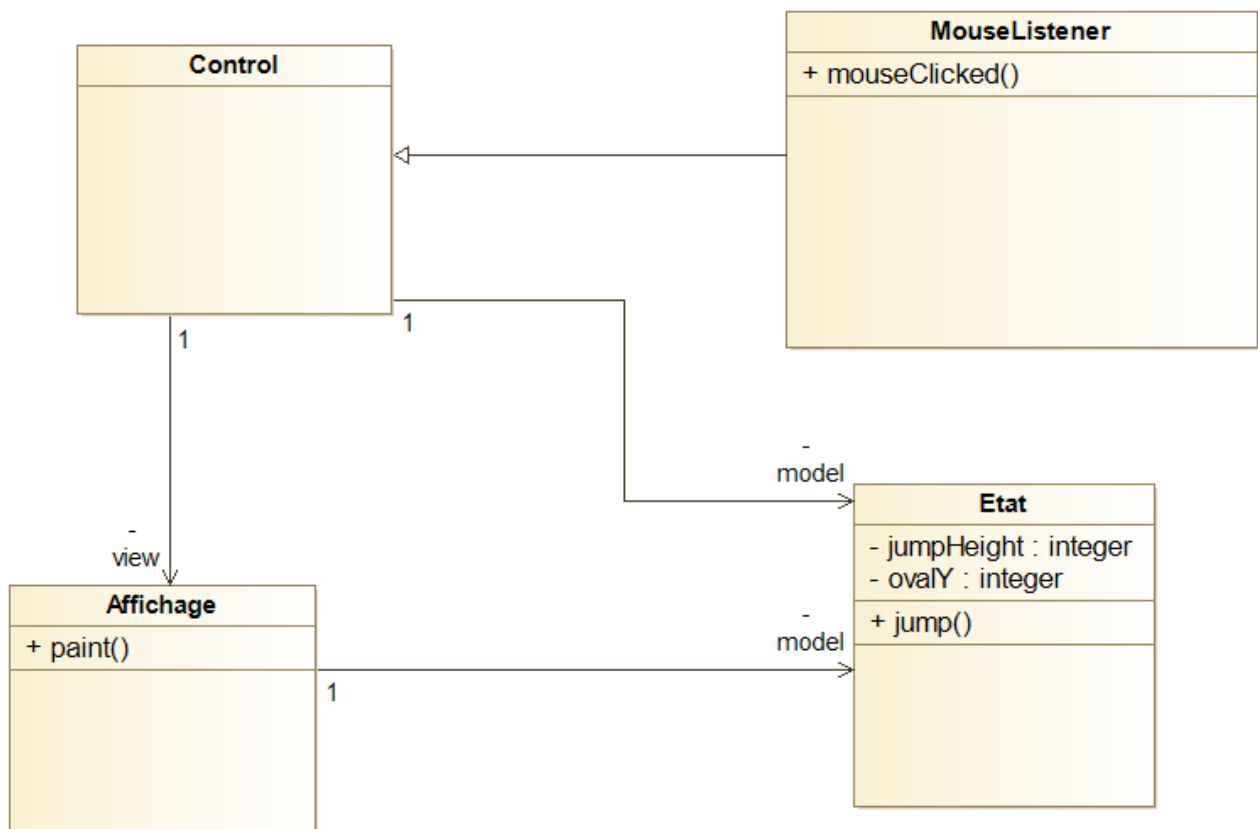
Conception détaillée

Pour la fenêtre avec un ovale, nous utilisons l'API Swing et la classe `JPanel`. Nous définissons les dimensions de l'ovale et de la fenêtre dans des constantes :

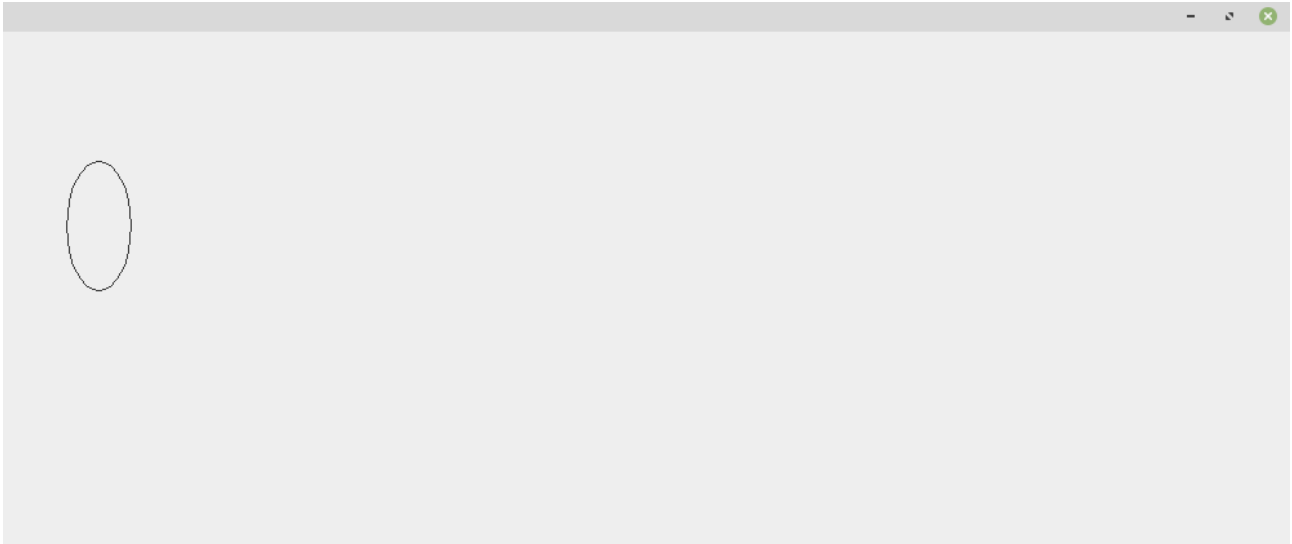
```
- public static final int width = 1000;  
public static final int height = 400;  
  
/* Default oval dimension  
*/  
  
- public static final int ovalWidth = width/20;  
- public static final int ovalHeight = height/4;
```



Pour le déplacement de l'ovale, nous utilisons la programmation événementielle avec la classe `MouseListener` et la hauteur est définie dans une constante.



Résultat



Afin de simuler un défilement de la fenêtre vers la droite, on diminue l'abscisse, à l'aide d'un thread, de tous les points visibles (ainsi que le premier point à gauche et à droite en dehors de la fenêtre afin de garder la continuité de la ligne brisée). La vitesse de défilement est exponentielle, elle accélère au fur et à mesure du jeu.

Génération de la ligne brisée

AjouterPoint () {

ratioAbscisse = AleatoireEntre(0.2 * largeurFenetre , 0.4 * largeurFenetre)

ordonne = AleatoireEntre(hauteurOval, hauteurFenetre - hauteurOval)

**ligneCassee.ajouter(
Point(abscisseDuDernierPoint + ratioAbscisse*largeurFenetre))**

}

On crée les premiers points visibles + un à droite (dans le constructeur de la classe Parcours). A chaque fois que le déplacement 'avance' va afficher le prochain point :

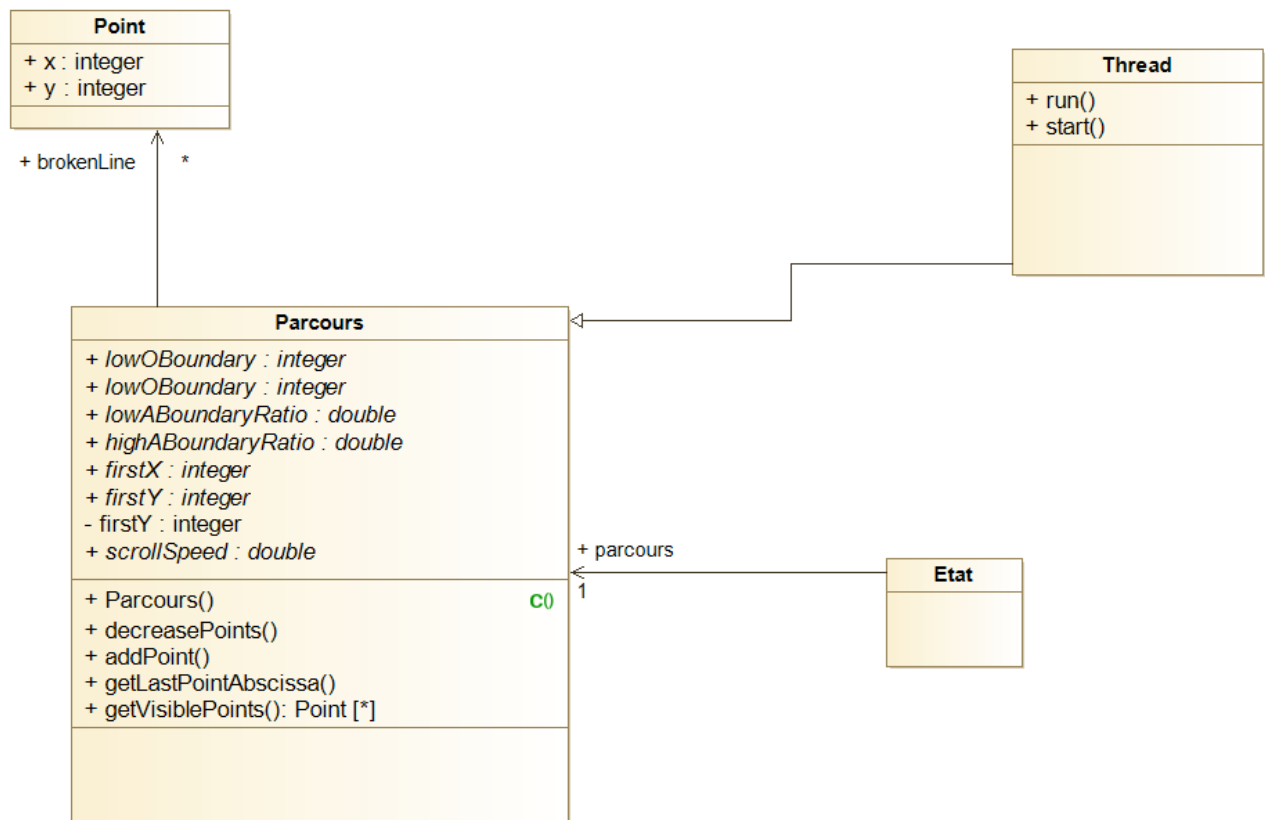
Si (abscisseDuDernierPoint < Affichage.width + ratioAbscisseMinimum)

Alors ajouterPoint

On supprime le dernier point si la position de l'avant dernier point est inférieur à zéro (plus utile pour l'affiche). (Verbose : "point deleted" dans la console afin de confirmer cette suppression).

Si (abscisseDuDeuxiemePoint < 0)

Alors supprimer(dernierPoint)



Collision

Il faut déterminer l'ordonnée de l'oval afin de savoir s'il rentre en collision avec la ligne brisée.

Tout d'abord il faut calculer l'ordonnée de l'oval.
Pour se faire on utilise ces formules :

distanceEntreLesDeuxPointsPlusProcheOval = sqrt ((Point2.y - Point1.y)² / (Point2.x - Point1.x)²);

Puis on calcule l'ordonnée de l'oval avec la formule :

**ordonneeOvalCalculee = Point1.y +
distanceEntreLesDeuxPointsPlusProcheOval * (Point2.y - Point1.y);**

Enfin on teste si l'oval est sorti de la ligne :

return (ordonneeOval > ordonneeOvalCalculee) ou (ordonneeOval + hauteurOval < ordonneeOvalCalculee))

Ajout du décor

Une image a été ajoutée à l'arrière-plan de la fenêtre.

Une musique se joue au démarrage du jeu (et qui boucle à l'infini).

Des fées ont également été ajoutées. Chaque fée est un thread. Une fée est générée toutes les 3 à 6 secondes (à l'aide d'un thread également). L'impression de déplacement des ces fées est gérée indépendamment de la ligne brisée. Chaque fée se déplace à sa propre vitesse grâce à son thread personnel.

Documentation utilisateur

Prérequis : Java avec un IDE

Mode d'emploi (cas IDE) : Importez le projet dans votre IDE, sélectionnez la classe Main à la racine du projet puis « Run as Java Application ». Cliquez sur la fenêtre pour lancer le jeu et faire monter l'ovale.

Documentation développeur

La méthode main est dans la classe "Main".

Les classes 'clés' sont "Parcours" et "Etat".

Les constantes **width** et **height** de la classe "Affichage" règlent la taille de la fenêtre. (Attention : les images (arrière-plan, héro et fées) ne s'adaptent pas en fonction de la taille de l'écran (afin de respecter la qualité de l'image).

Améliorations :

- Image grande et scalable afin de pouvoir manger la taille de la fenêtre selon l'envie de l'utilisateur.
- Les points deviennent des objets à ramasser (plutôt qu'un chemin à suivre). Le joueur doit en ramasser le plus possible.
- Les fées peuvent devenir des monstres à éviter. Si le joueur entre en collision avec un monstre, il a perdu.
- Bug de collisions
- Menu pour permettre à l'utilisateur de modifier la difficulté (vitesse, monstre et temps de jeu) et le volume du son.
- Ramasser X pièces donne le pouvoir de tirer sur un monstre pour l'éliminer.
- Avec le temps, la vitesse de chute du héro, de défilement, et le nombre de nombre générés augmentent.

Conclusion et perspectives

L'application est actuellement constituée d'un héros qui peut monter à chaque clique de souris et qui descend avec le temps. Une ligne brisée défile vers la droite, et l'utilisateur doit garder son héros dans la ligne car les collisions sont comptées. Une musique commence peu après le lancement du jeu.

Difficultés rencontrées :

- Défilement de la fenêtre (décrémentation de l'abscisse des points)
- Gestion des points "visibles" (surtout les 2 points "en dehors" de la fenêtre graphique qui permettent d'afficher la continuité de la ligne brisée)
- Compréhension de la structure du modèle MVC (échange avec la classe Etat)
- Gestion des collisions (calcul de l'ordonnée de l'oval)
- Démarrage du jeu au premier clic de l'utilisateur (et non pas au démarrage de l'application)