1.INTRODUCTION

L'objectif est de réaliser un mini-jeu inspiré de FlappyBird dans lequel se déplace le long d'une ligne brisée. Le but est d'éviter que l'ovale sort de la ligne pour cela l'utilisateur peut clicher sur l'écran pour faire monter l'ovale qui redescend ensuite tout seul.

2.ANALYSE GLOBALE

Dans la première partie du projet(séance 1) nous nous intéressons au sous ensemble des fonctionnalités:
-Création de la fenêtre ou se dessine l'oval;
-Déplacement de l'ovale vers le haut quand on clique la fenêtre.
-Organisation du projet sous la forme du modèle MVC

Dans la deuxième partie du projet(séance 2) nous nous intéressons au sous ensemble des fonctionnalités:
-Utilisation des threads pour rafraîchir independent l'oval, l'affichage et la ligne brisée.
-La création d'une ligne brisée et son déplacement automatique pour donner l'impression que l'ovale avance le long de cette ligne.

3. PLAN DE DÉVELOPPEMENT

Liste de taches (Séance 1):
a.Analyse du problème (15 mn)
b.Conception, développement et test d'un fenêtre avec un ovale (30 mn)
c.Conception, développement et test du mécanisme de déplacement de l'ovale (45 mn)
d.Acquisition de compétences en Swing (60 mn)
e.Documentation du projet (60 mn)

Tache	15	30	45	60	75	90	105	120	135	150	165	180	195	210
a														
b														
c														
d														
e														

Liste de taches (Séance 2):
a.Analyse du problème (15 mn)
b.Conception, développement et test du thread pour déplacer l'oval vers le bas (30 mn)
c.Conception, développement et test de la construction de la ligne brisée (30 mn)
d.Conception, développement et test de l'animation de la ligne brisée (75 mn)
e.Documentation du projet (60 mn)

Tache	15	30	45	60	75	90	105	120 /2H	135	150	165	180 /3H	195	210	225	240 /4H
a																
b																
c																
d																
e																

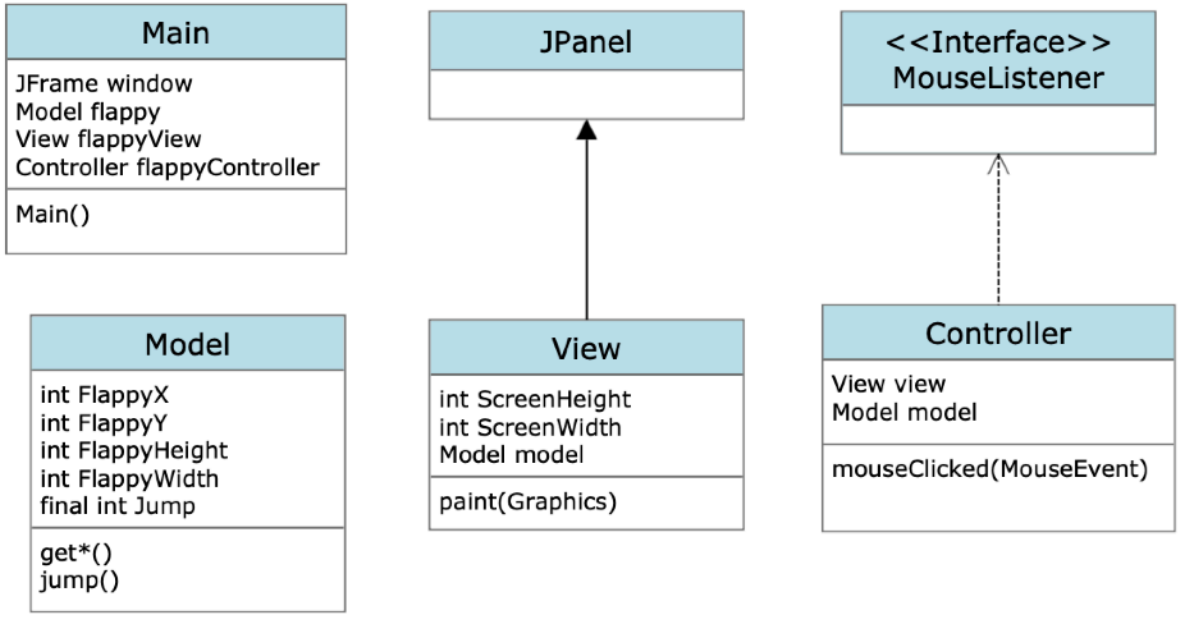
4. CONCEPTION GENERALE

Dans la première séance nous avons adopté le motif MVC pour le développement de notre interface graphique. Quand on clique la fenêtre on modifie les données de l'ovale (ses coordonnes en y). L'ovale est donc le modèle et la modification de ses coordonnées correspond à des actions du contrôleur. Puis toutes modifications subies par le modèle sont affichées dans l'écran (cet affichage est la view).

Dans la deuxième séance nous avons utilisé les threads pour metre a jour simultanément les différents elements du projet: Le modèle est mis a jour avec le thread Fly, l'affichage est mis à jour dans un thread à part qui fait appel à repaint et la ligne brisée (appelée Path) est mise a jour dans le thread RefreshPath.
Pour respecter le modèle MVC et on raison de l'augmentation du nombre de classes, l'utilisation de packages devient nécessaire:
Package Controller: Classes Controller et Fly.
Package View: Classes View et RefreshView
Package Model: Classes Model, Path et RefreshPath.

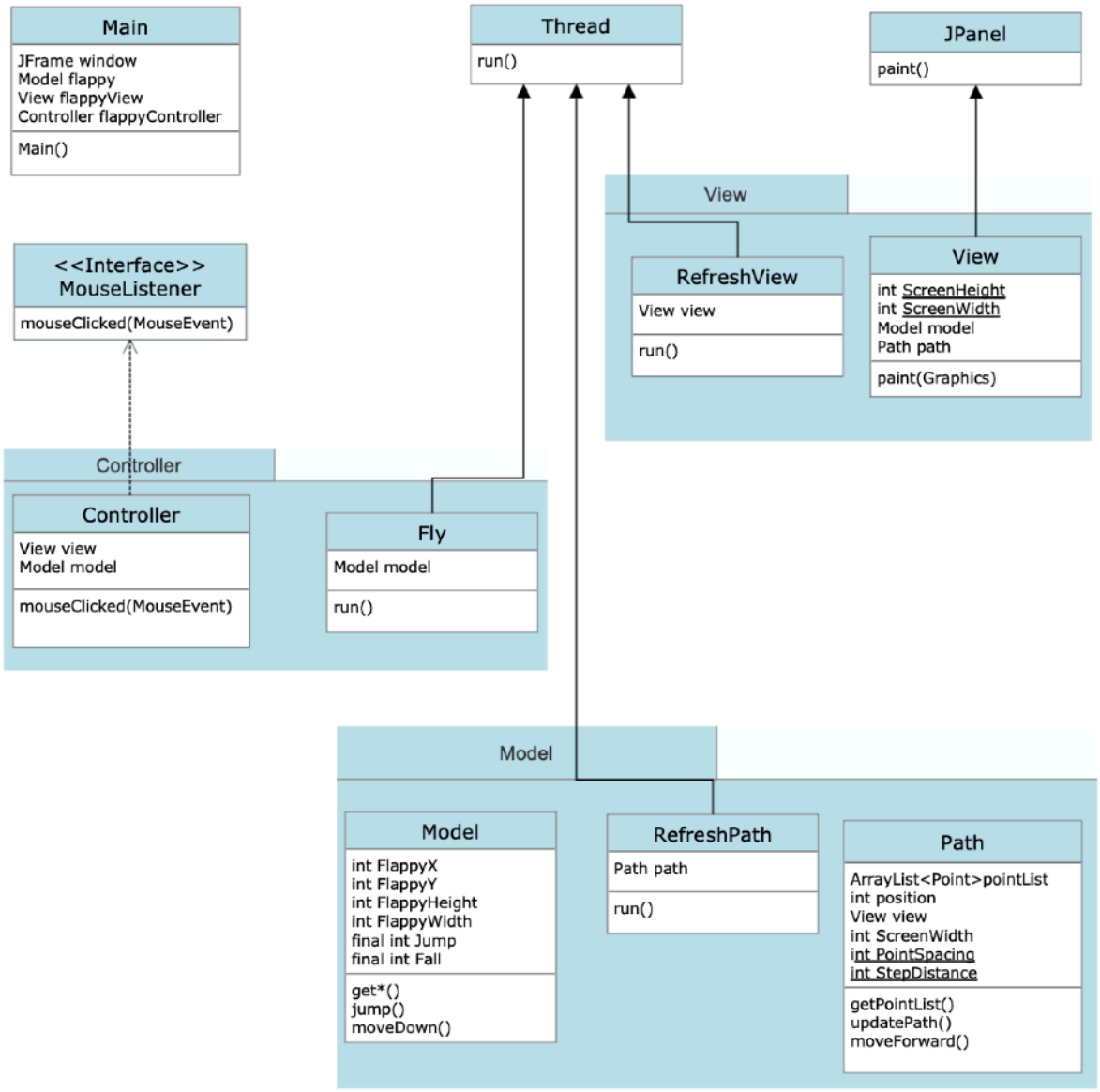
5. CONCEPTION DÉTAILLÉE

SÉANCE1:
Pour la fenêtre avec un ovale, nous utilisons l'API Swing et la classe `JPanel`. Nous définissons les dimensions de l'oval et de la fenêtre dans des constantes visibles dans le diagramme de classes.
Pour le déplacement de l'ovale, nous utilisons la programmation événementielle avec la classe `MouseListener` et la hauteur est définie dans une constante.



SÉANCE 2:
Implementation des Threads qui rafraîchissent indépendamment la view, et les elements du model: les classent héritent de la classe `Thread` puis on Override la méthode run pour rafraîchir nous objets constamment avec une while(true) loop.

Voici le diagramme UML après la séance 2:



Algorithmes de la classe Path:

L'algorithme qui crée la ligne brisée est codé dans le constructeur de la classe Path:

L'algorithme se sert de la constante:
PointSpacing: c'est l'espace entre chaque point de la ligne brisée, on le définit comme la width de l'écran divisé par 10.

Initialisation:
x=PointSpacing.
y=nombre aléatoire dans l'écran.

Tant que x reste dans l'écran faire:
Ajouter le point (x,y) à l'attribut pointList.
x+=PointSpacing
y= un autre nombre aléatoire dans l'écran.

Algorithme qui mets a jour la pointListe dans moveForward:
On se sert de la constante:
StepDistance: c'est la distance qu'on va décaler les points de la ligne brisée pour simuler que l'ovale se déplace vers la droite.

Pour tout point p de la ligne brisée (de l'ArrayList pointList):
p.x-=StepDistance

Algorithme qui mets à jour la pointListe pour qu'elle ne contienne que les points visibles ou nécessaires (implémenté dans la méthode updatePath):

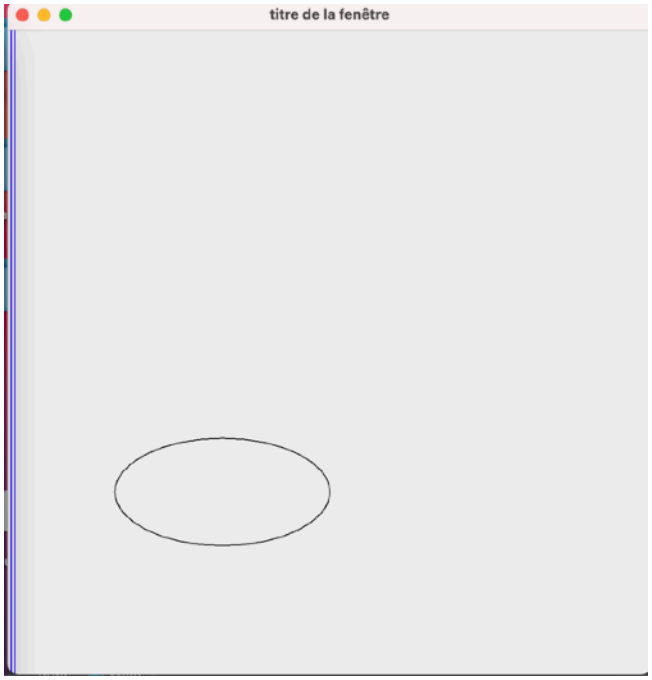
Initialisation:
On cree une copie de pointList

Si Flappy avance jusqu'à un point de la ligne brisée faire:
(On peut identifier cette situation si la position est un multiple de PointSpacing)

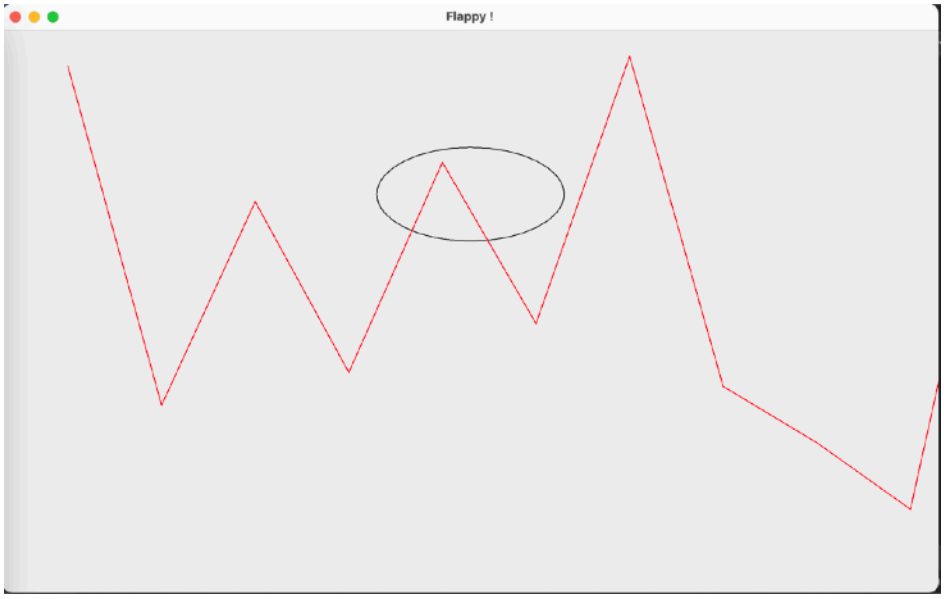
-Eliminer premier element de la copie
-Ajouter un point a la fin de la ligne brisée qui a pour coordonnées:
x : coordonnée x du dernier point de la copie plus PointSpacing.
y: un nombre aléatoire dans l'écran.

Mettre a jour pointList en la remplaçant par cette copie modifié.

6.RÉSULTAT
SÉANCE 1:



SÉANCE 2:



7. DOCUMENTATION UTILISATEUR
-Prérequis : Java avec un IntelliJ IDEA.
-Mode d'emploi: 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 faire monter l'ovale.

8.DOCUMENTATION DÉVELOPPEUR
Les prochaine fonctionnalité à implementer sera de la detection de collisions.

9.CONCLUSION ET PERSPECTIVES

Pendant la premiere séance, on a construit une interface interactive très élémentaire qui affiche un ovale et le déplace quand l'utilisateur click.
Le plus difficile était de inclure l'objet qui hérite de MouseListener pour qu'il attende des cliques lors de l'affichage de la fenêtre: ma solution temporelle était de créer un Objet contrôleur dans le constructeur de View. Le problème de cette solution est qu'elle ne permette pas l'independence entre les classes. J'ai ainsi créer un objet contrôleur dans le main et j'ai donné accès à la JFrame pour qu'il ajoute directement dans son constructeur le MouseListener. J'ai donc appris a séparer les elements du modele MVC en jouent avec les constructeurs des classes.

Pendant la deuxième séance j'ai appris a utiliser les threads en java et j'ai compris que leur importance est dans le fait qu'elle permettent le rafraîchissement en parallèle des différents elements du modele(ici l'ovale de Model et la ligne brisée de la classe Path).
La plus grande difficulté rencontrée à été lors de l'animation de la ligne brisée: dans un premier moment j'ai essayé de mettre a jour la ArrayList pointList (Attribut de la classe Path) dans chaque appel de la méthode paint() dans la classe View. Le problème de faire cela est que la Paint mon pointList peuvent pas se mettre a jour au meme temps. Pour résoudre le problème j'ai donc créé une méthode updatePath() qui est appelé par une thread qui a l'unique devoir de mettre a jour la valeur de pointList. J'ai donc appris a séparer mes mises à jour dans des différentes threads qui a leur tour mettent a jour les modèles.