université
PARIS-SACLAY

# Rapport de projet PCII

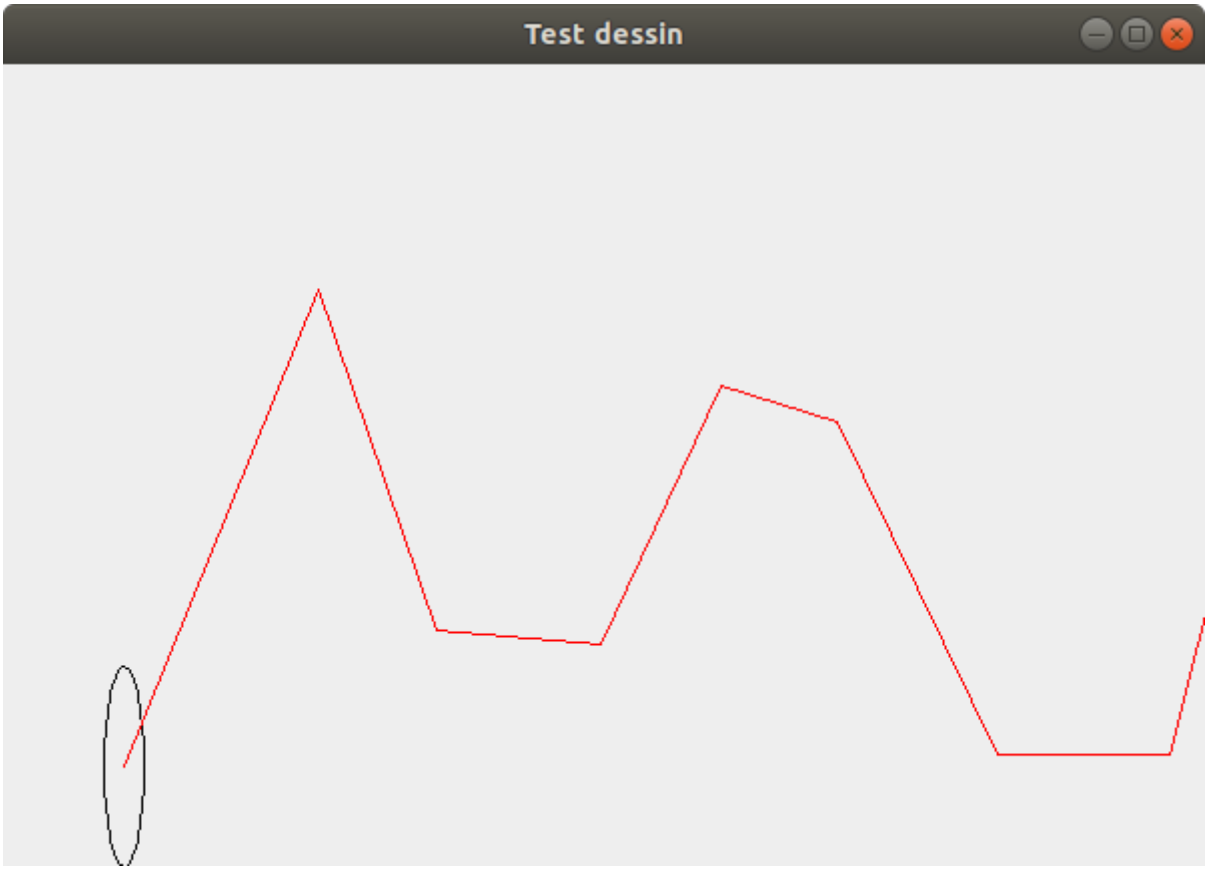
Réalisé par
Hongyu YAN

Sous la direction de
Thi Thuong Huyen Nguyen

Année universitaire 2020-2021
Licence Informatique

# Introduction

Le projet dont je suis en charge consiste à réaliser un mini-jeu en java. Ce jeu est inspiré de *flappy bird* dans lequel un ovale se déplace le long d'une ligne brisée. Le but du jeu est d'éviter que l'ovale sorte de la ligne. Pour cela, le joueur peut cliquer sur l'écran afin de faire monter l'ovale, qui redescend ensuite tout seul. Voici à quoi pourrait ressembler l'interface graphique de notre jeu:



# Analyse globale

Il y a trois fonctionnalités principales:

- l'interface graphique avec l'ovale et la ligne brisée
- le défilement automatique de la ligne brisée
- la réaction de l'ovale aux clics de l'utilisateur

Tout d'abord, je m'intéresse à un sous-ensemble de fonctionnalités :

- création d'une fenêtre dans laquelle est dessiné l'ovale
- déplacement de l'ovale vers le haut lorsqu'on clique dans l'interface

Ces deux sous-fonctionnalités sont prioritaires et simples à réaliser.

Ensuite, je m'occupe à trois fonctionnalités:

- déplacement de l'ovale vers le bas progressivement
- création d'une ligne brisée
- animation de la ligne

Ces fonctions sont plus difficiles et il faut maîtriser les connaissances multi-thread.

Et puis, je préfère réaliser certaines fonctions :

- détection des collisions
- si l'ovale sort de la ligne :
    - Les thread de vol et d'avancement du parcours s'arrêtent
    - L'interface ne réagisse plus aux clics souris
    - Un message s'affiche avec le score de l'utilisateur

Ces fonctionnalités ne sont pas trop difficiles, mais il faut appliquer une formule pour savoir si l'ovale est sorti de la ligne brisée.

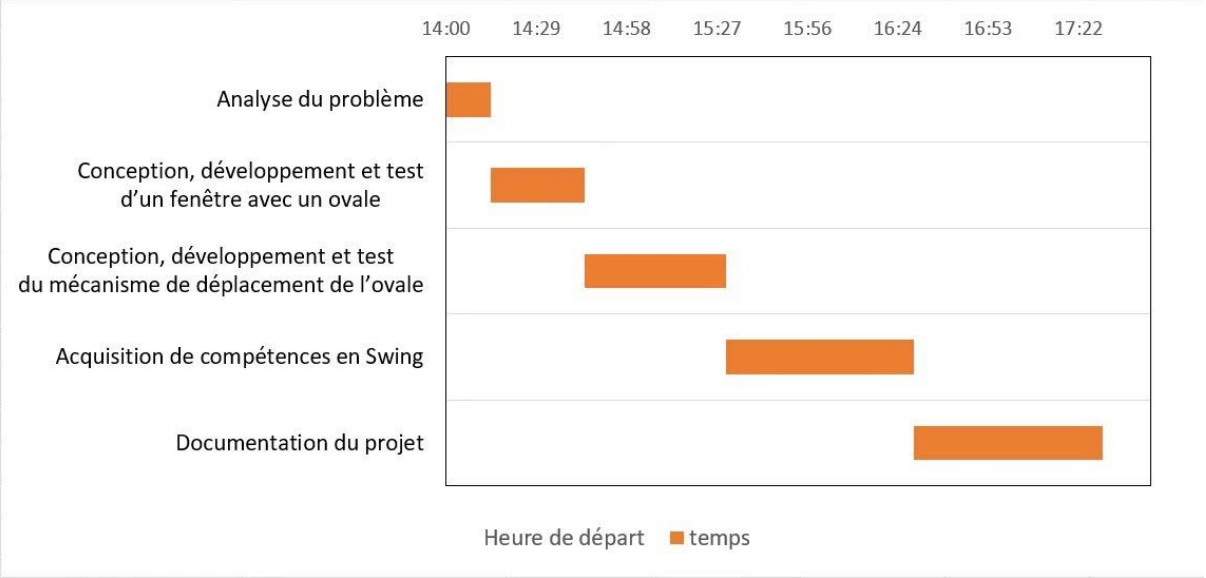
Enfin, je voudrais améliorer mon interface :

- Fixer l'épaisseur d'un trait des dessins
- Ajout d'éléments de décors

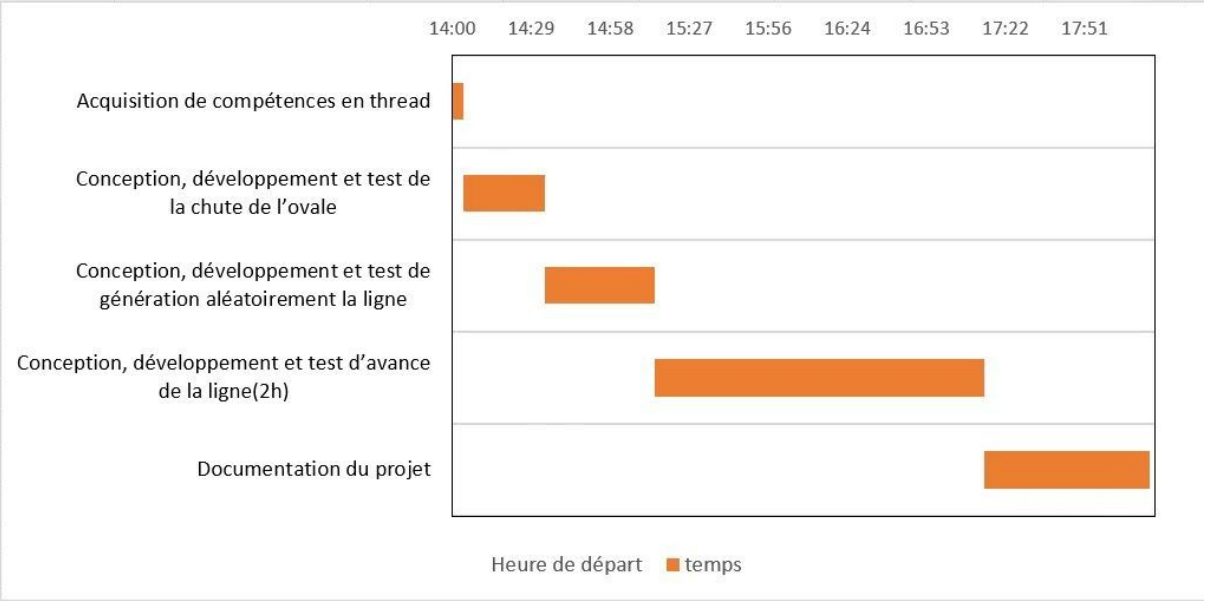
Ces améliorations ne sont pas prioritaires et sont optionnelles, mais elles sont beaucoup plus difficiles.

# Plan de développement

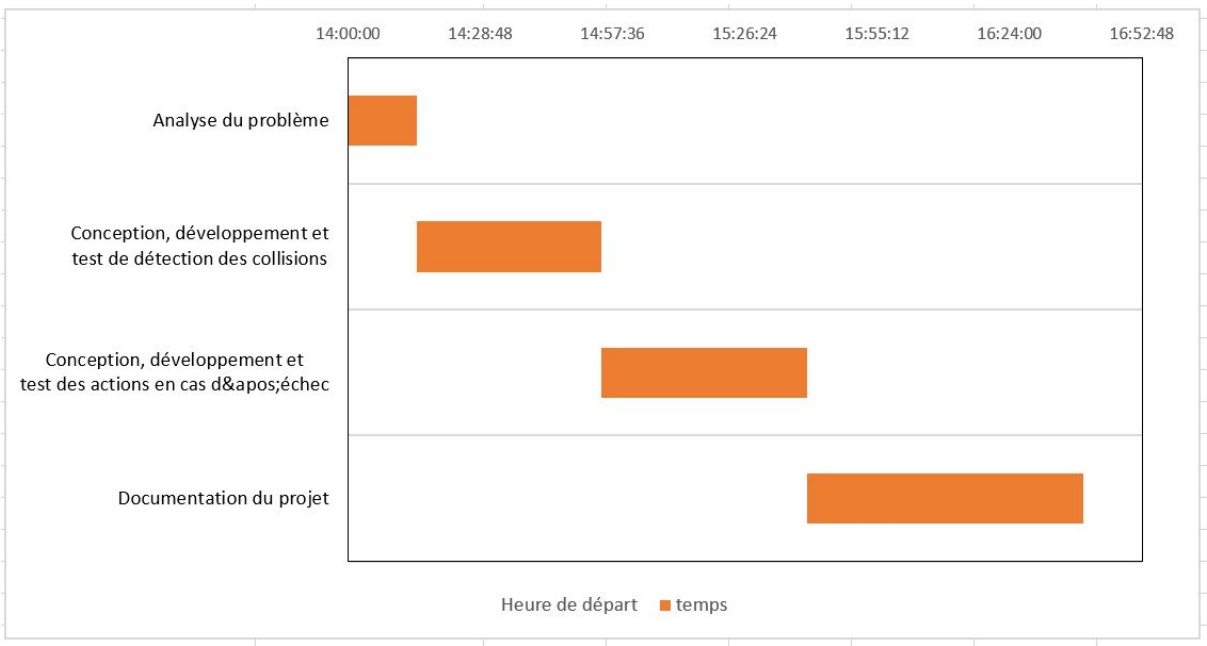
## Première Partie



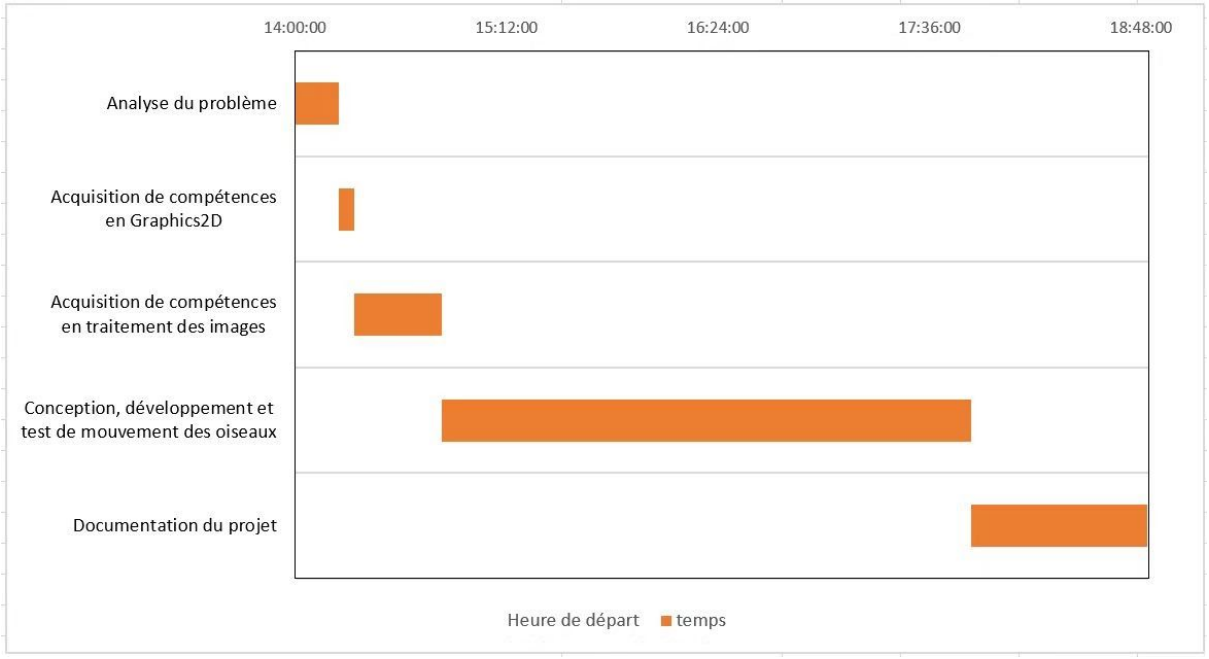

## Deuxième Partie



# Troisième Partie



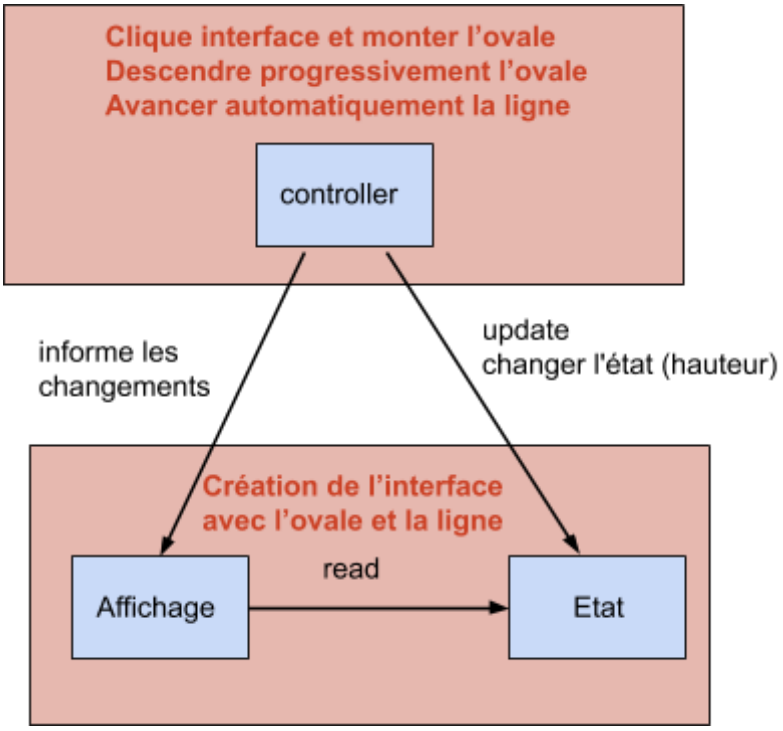

# Amélioration



# Conception générale

L'interface graphique s'est construite autour du modèle MVC. Le package ***Vue*** s'occupe de dessiner l'interface. Le package ***Model*** définit l'ensemble des données qui caractérisent l'état de l'interface. La modification de ces données correspond à un changement de l'affichage dans l'interface graphique. Le package ***control*** effectue les changements dans l'état et informe la ***vue*** d'un changement.

La classe ***Affichage*** pourrait dessiner un ovale et une ligne brisée avec les points définis dans la classe ***Parcours*** qui est déclarée dans la classe ***Etat***. Elle permet aussi d'afficher les oiseaux définis dans la classe ***VueOiseau***. Les classes du package ***control*** modifient les états des dessins afin de faire voler l'ovale, avancer la ligne et les oiseaux, puis ils informent la classe ***Affichage*** des changements pour redessiner. Lorsqu' on clique sur l'écran, c'est la classe ***Control*** qui gère les événements.



# Conception détaillée

Pour la fenêtre avec un ovale, j'utilise l'API Swing et la classe JPanel. Je définis les dimensions de l'oval et de la fenêtre dans des constantes.

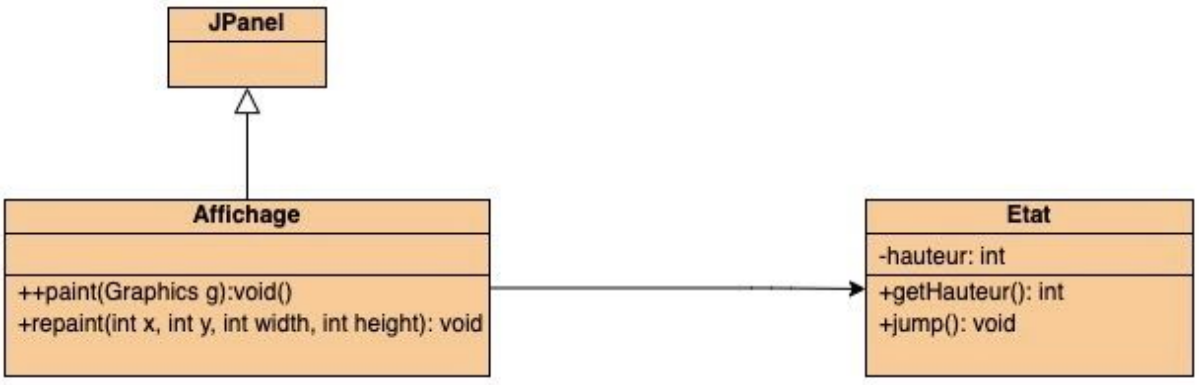
Constante(s) de la classe Affichage:

- LARG : Largeur de l'interface
- HAUT : Hauteur de l'interface

*L'ovale est dessiné à l'intérieur d'un rectangle*

- X : l'abscisse du coin supérieur gauche du rectangle
- Y : l'ordonnée du coin supérieur gauche du rectangle
- WIDTH : Largeur du rectangle (ovale)
- HEIGHT : Hauteur du rectangle (ovale)

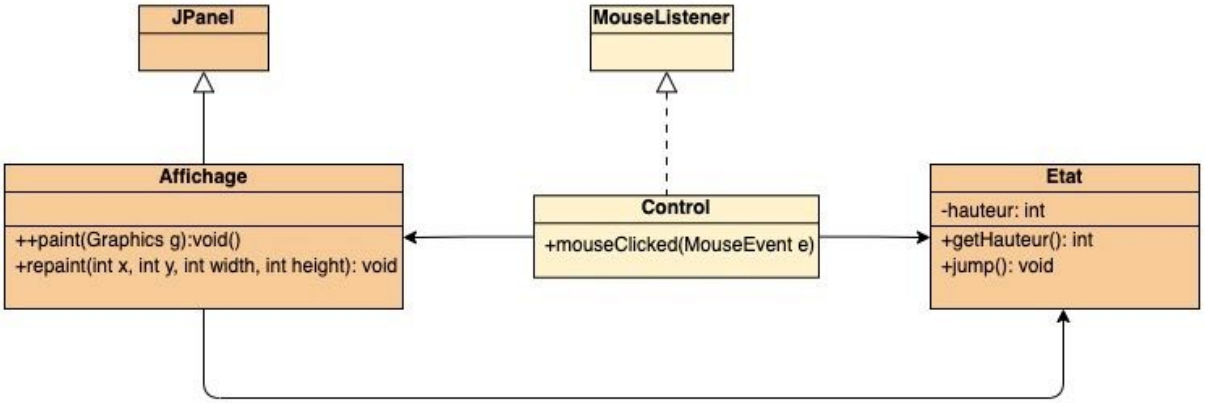
Voici un diagramme de classe pour afficher une fenêtre avec un ovale dedans.




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

Constante(s) de la classe Etat :

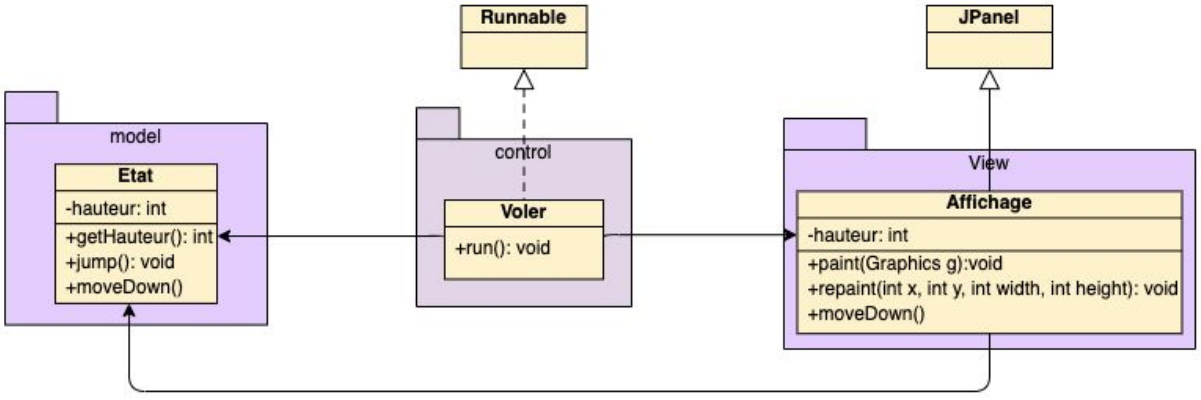
- SAUT : La valeur de la hauteur de quelques pixels vers le haut




Afin de réaliser la fonctionnalité de la chute de l'ovale, je demande de l'aide à la classe Thread. L'interface graphique est déjà un *thread* à part, l'exécution de l'autre thread sera entrelacée avec celle de l'interface. Et la hauteur est définie dans une constante.

Constante(s) de la classe Etat :

- DESC : La valeur de la hauteur de quelques pixels vers le bas




Pour le défilement automatique de la ligne brisée, je dois d'abord générer aléatoirement les coordonnées et j'ai proposé un algorithme. De plus, cet algorithme de génération de parcours devrait garantir que la pente n'est pas plus élevée que la vitesse de chute, qui dépend de la valeur de la constante.

**Parcours() : void**

x <- Affichage.***WIDTH***/2 + Affichage.***X***;

y <- Affichage.***HEIGHT***/2 + Affichage.***Y***;

Ajouter Point(x,y) dans la liste des points

**ajouterPoint() : void**

**tant que** x <= Affichage.***LARG*** **faire**

x <- Random.nextInt(250) + (**points**.get((**points**.size() - 1)).x + 60;

y <- Random.nextInt(Affichage.***HAUT*** - 100) + 50;

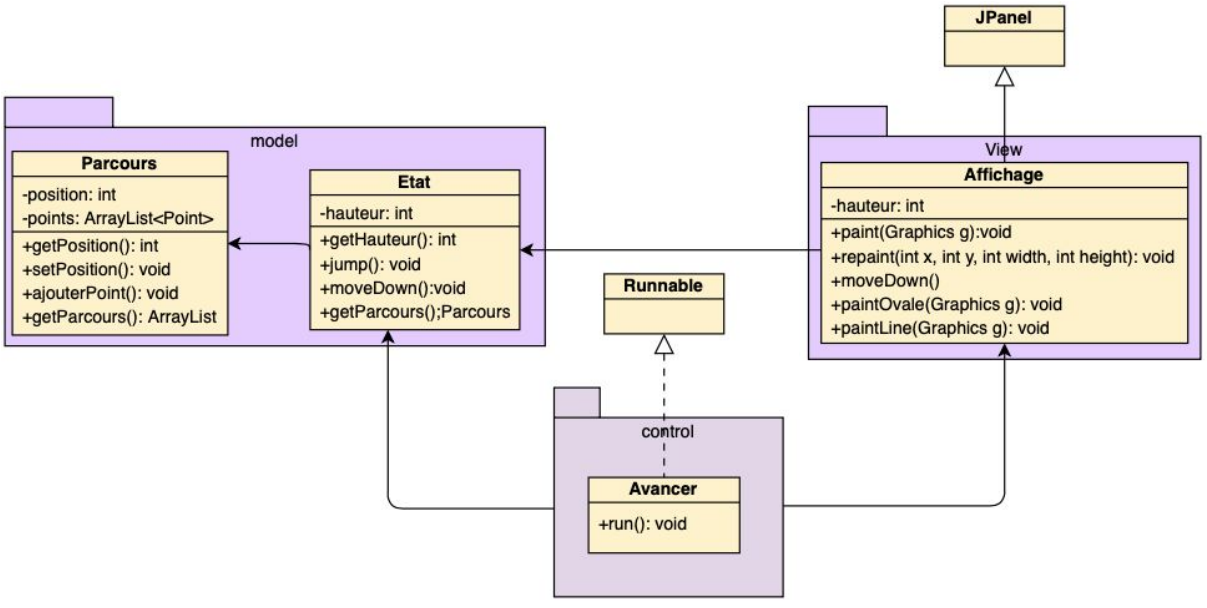
Ajouter Point(x,y) dans la liste des points

**fin tant que**

Ensuite j'ai besoin d'un autre thread pour dessiner la ligne, elle devrait être infinie et générée à la volée chaque fois qu'un nouveau point est nécessaire. Et la valeur du mouvement est définie dans une constante.

Constantes de la classe Parcours :

- AVANCE : la valeur de l'incrément de la position de la ligne brisé



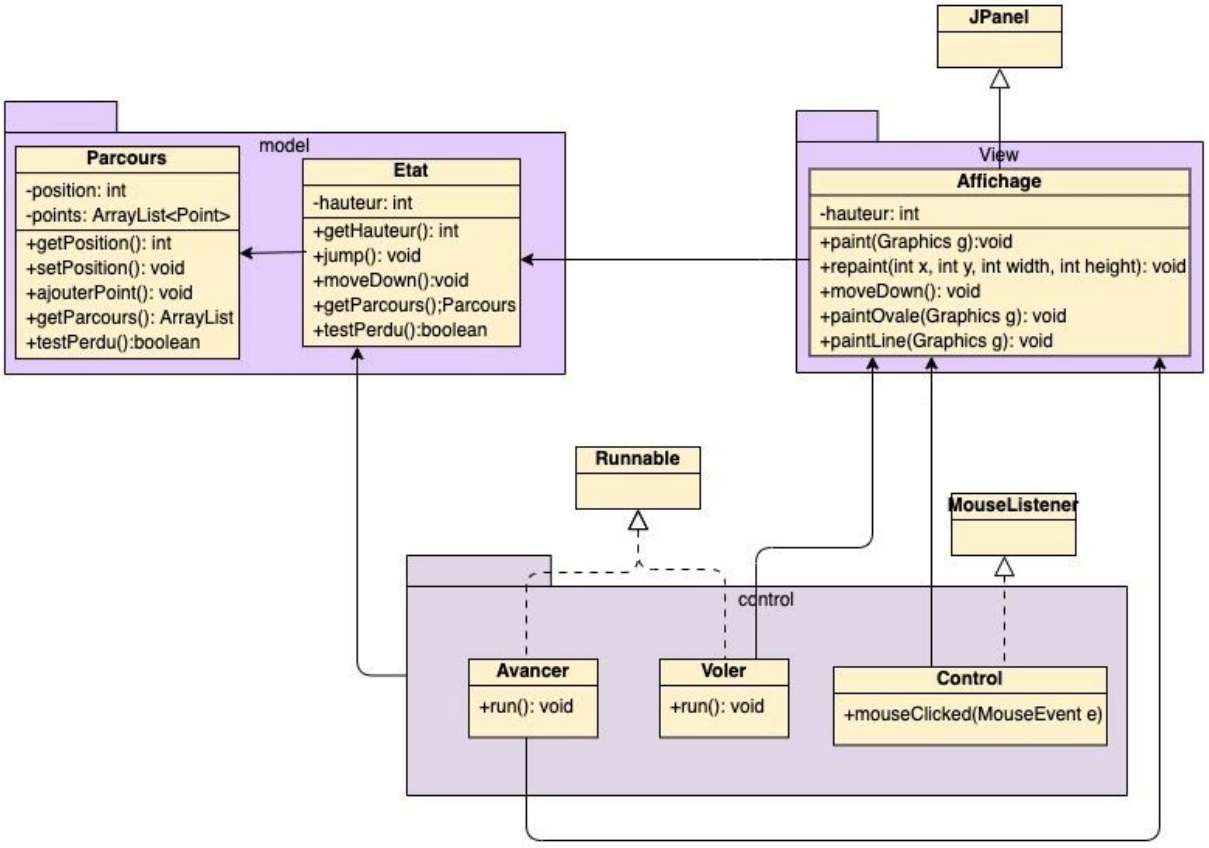
La dernière étape pour avoir un système fonctionnel est de détecter les collisions. Pour cela, nous allons déterminer la valeur de l'ordonnée sur la ligne brisée au point d'abscisse correspondant à la position 0 de l'ovale.

On sait que deux points déterminent une ligne avec la formule ci-dessous en connaissant les coordonnées de deux points:

$$y = kx + b$$
$$k = (y2 - y1) / (x2 - x1)$$
$$y1 = kx1 + b$$

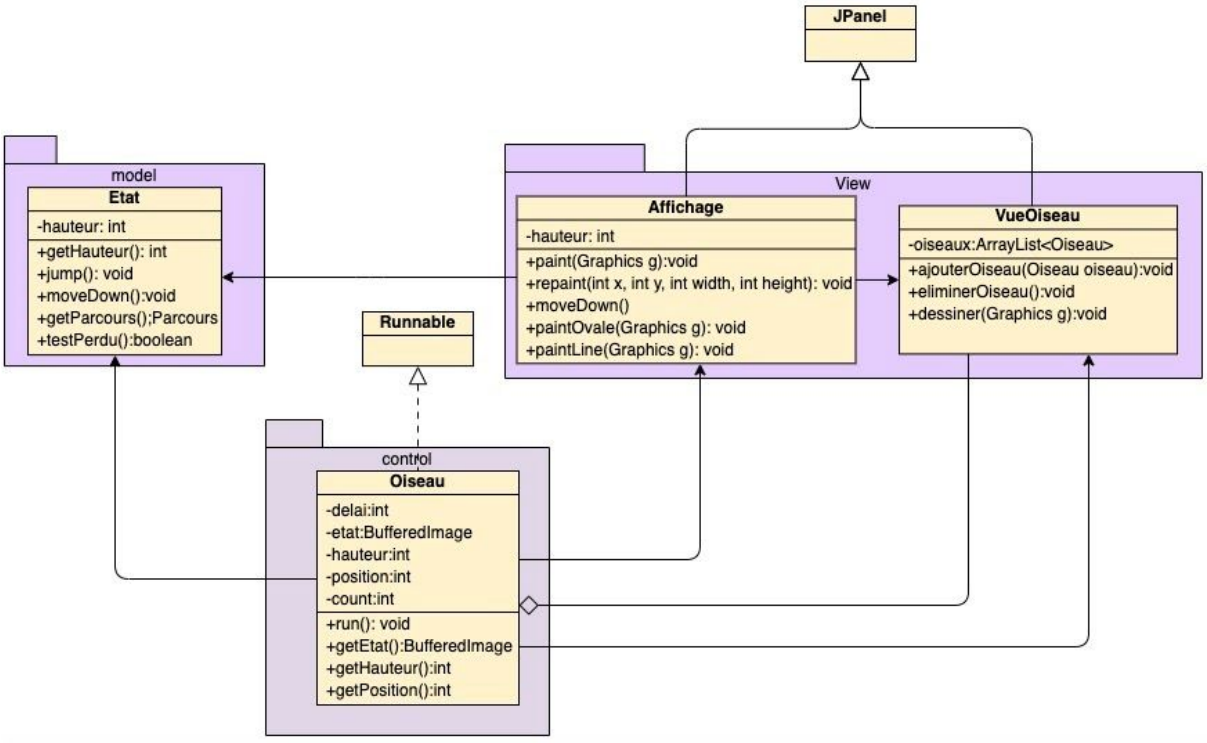
x: abscisse de la position 0 de l'ovale
y: ordonnée sur la ligne brisée correspond à ***x***

Si le jeu échoue, c'est-à-dire que l'ovale sort de la ligne brisée, l'ovale et la ligne arrêtent de bouger. De plus, on affiche un message de perdu en utilisant JOptionPane.*showMessageDialog.*

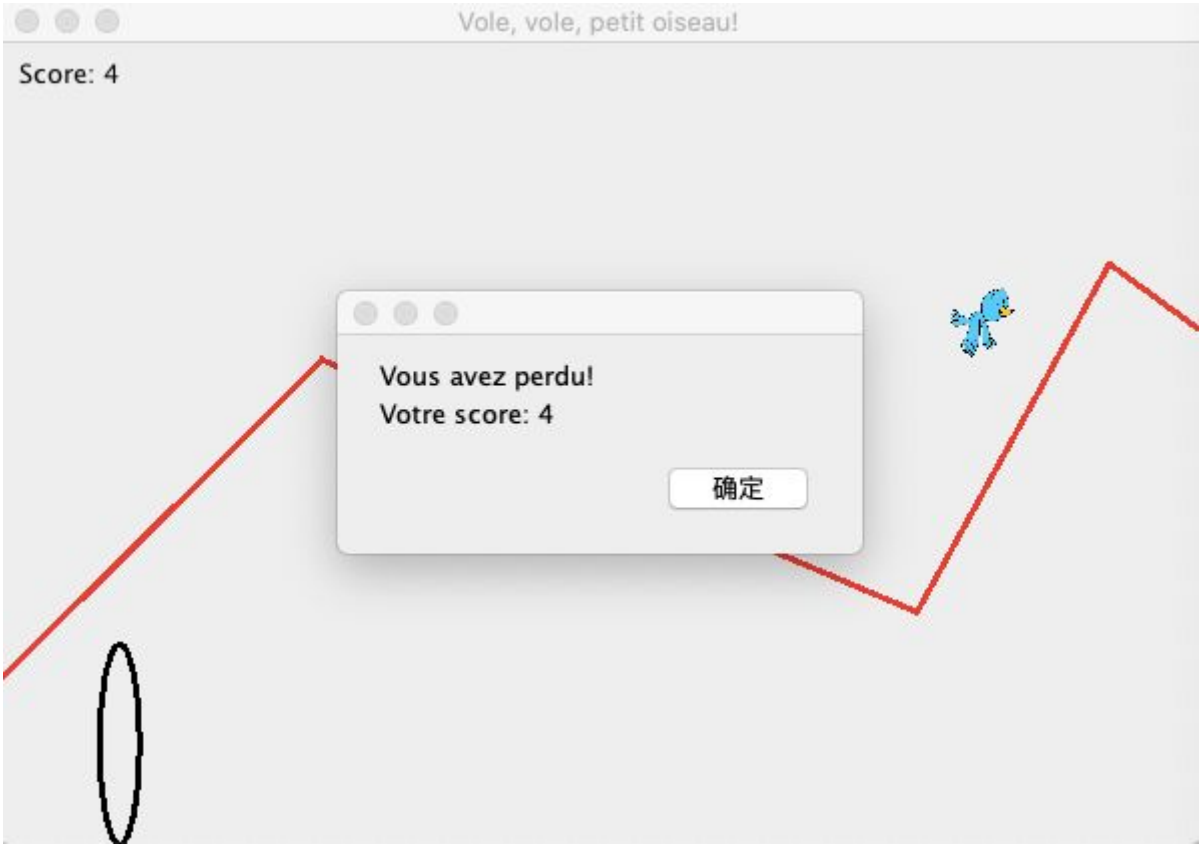


Afin d'améliorer la qualité de l'affichage, je vais d'abord utiliser différentes épaisseurs de traits et de couleur pour créer de la consistance en convertissant l'objet Graphics qui est passé à la méthode paint en instance de la classe Graphics2D, qui est une sous-classe de Graphics.

Et puis, j'utilise des images plutôt que des dessins faits à la main. Je télécharge une image de l'oiseau dans mon projet. Je crée ensuite une classe Oiseau qui implémente un *thread* avec un paramètre de vitesse variable, choisi aléatoirement à la création de l'instance. La vue du MVC doit alors afficher les oiseaux qui se déplacent dans le ciel, de droite à gauche, comme s'ils étaient «rattrapés» par l'ovale.



# Résultat




# Documentation utilisateur

Prérequis : Java avec un IDE

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. L'ovale perd de l'altitude, si vous ne cliquez pas.

# Documentation développeur

C’est la classe Main qui contient la méthode main. J’utilise le model MVC pour organiser mes codes.

Allez dans le package view pour voir comment afficher l’interface. Vous pouvez modifier les valeurs des constants définis dans la classe *Affichage* afin de changer la taille de la fenêtre et celle de l’ovale.

Le package model définit l’ensemble des données qui caractérisent l’état de l’interface. La modification de ces données correspond à un changement de l’affichage dans l’interface graphique. Vous pouvez contrôler la vitesse de montée ou de descente de l'ovale en modifiant les constantes SAUT ou DESC de la classe *Etat*. De même, vous pouvez constater le changement de la vitesse du mouvement de la ligne si vous changez le constante AVANCE de la classe *Parcours*.

Le package control vous permet de savoir comment gérer les événements et la manière dont l’état du modèle change.

Pour l’instant, je n’ai qu’un oiseau dans le background, nous pouvons munir la classe VueOiseau d’un générateur aléatoire d'oiseaux, avec une faible probabilité, pour avoir des oiseaux qui défilent sur l’écran.

# Conclusion et perspectives

J’ai créé une fenêtre dans laquelle est dessiné l’ovale et une ligne brisée qui avance toute seule. Lorsqu’on clique dans l’interface, l'ovale se déplace vers le haut et redescend si on arrête de cliquer. Si on perd le jeu, tous les threads s'arrêtent

Ce qui était difficile, c'est que je n'ai jamais été exposé à la conception d'interfaces en Java. Mais avec l’aide du tutoriel du professeur et des documents sur Internet, j’ai réussi à obtenir l’affichage. De plus, il faut bien organiser les codes afin de le rendre plus lisible. Heureusement, je l'ai résolu en utilisant le modèle MVC. Ensuite, on doit appliquer les notations des threads et bien définir les coordonnées des points pour construire une ligne correcte.

Nous pouvons encore apporter beaucoup d'amélioration, par exemple, pour avoir un parcours plus élégant, nous pouvons utiliser des courbes de Bézier.