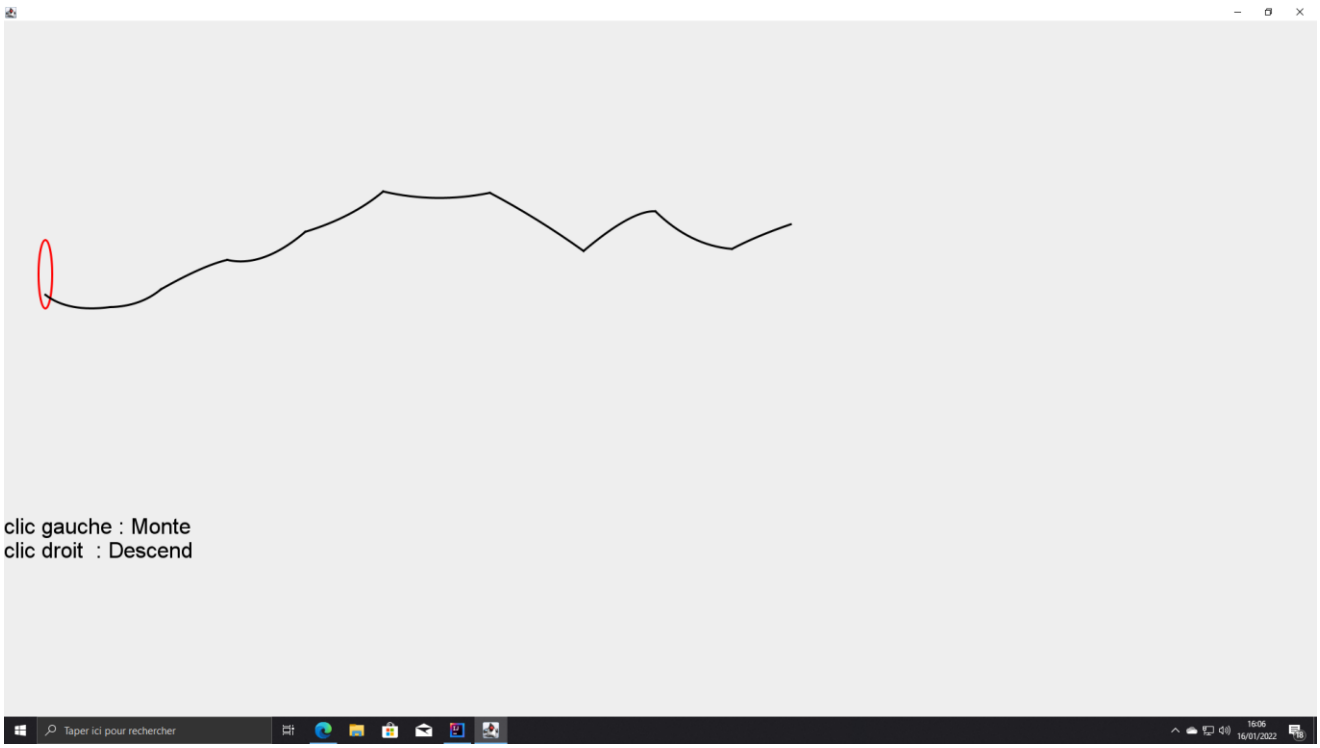
Introduction :

Le but du projet est de créer une version simple de Flappy bird. Flappy Bird consiste à éviter les obstacles en se déplaçant dans des anneaux en faisant voler ou descendre l'oiseau en tapotant sur notre écran mobile. Notre version du jeu sera une version simplifié avec un ovale représentant l'oiseau qui devra rester dans une ligne brisée sans la touchée.

Le jeu a été implémenter en Java en utilisant comme représentant du oiseau un ovale (comme dis précédemment), cette fois ci, le joueur utilisera la souris pour cliquer a chaque fois il souhaite faire voler ou descendre l'ovale.

Voici ci-dessous un exemple du début de projet */!\ ATTENTION cela est une version non terminé du projet /!* :



Analyse Globale :

Notre projet se compose en plusieurs fonctionnalité (trois principalement) : Nous avons premièrement l'interface graphique, qui, dans le modèle MVC sera dans Vue, l'interface permettra le controle de la ligne brisée ainsi que l'ovale represdentant l'oiseau. Il y ensuite la reaction automatique de la ligne brisée par rapport aux actions du joueurs (defilement etc) qui sera implémenté dans Controleur. Puis enfin la réaction de l'ovale par rapport aux actions du joueurs qui seront aussi implémentés dans Controleur.

Dans la première partie du projet (séance 1), nous nous intéressons uniquement 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 car elles utilisent l'Api Swing de base necessitant seulement la bibliothèque java.

Dans la deuxième partie du projet (séance 2), nous nous intéressons uniquement aux fonctionalités suivantes :

° Le déplacement de l'ovale et la réaction de la fenêtre par rapport à celle ci.

° La génération infinie d'une ligne brisée.

° La génération du score du joueur par rapport à sa position.

Dans la troisième troisième partie du projet (séance 3) nous allons nous occupé de deux choses indispensable :

° La detection de collision entre l'ovale et la ligne brisée.

° L'implèmentation graphique du décor et du jeu.

Plan de développement :

Interface graphique :

Le temps de travail estimé : 3h

— Analyse du problème : 30 min

— Acquisition de compétences en Swing : 60 min

— Conception et test de la fenêtre : 30 min

— Conception de l'ovale : 30 min

— Documentation du projet : 30 min

Déplacement de l’ovale :

Le temps de travail estimé : 1h20

— Analyse du problème : 20 min

— Conception des mouvements et des réactions : 1h

Gravité sur l’ovale :

Le temps de travail estimé : 1h

— Analyse du problème : 10 min

— Conception de la prise de mouvement et thread : 1h10

La ligne brisée :

Le temps de travail estimé : 1h15

— Analyse du problème : 15 min

— Conception, tests : 1h

Ligne brisée infinie :

Le temps de travail estimé : 1h15

— Analyse du problème : 15 min

— Conception, tests : 1h

Detection + Collision :

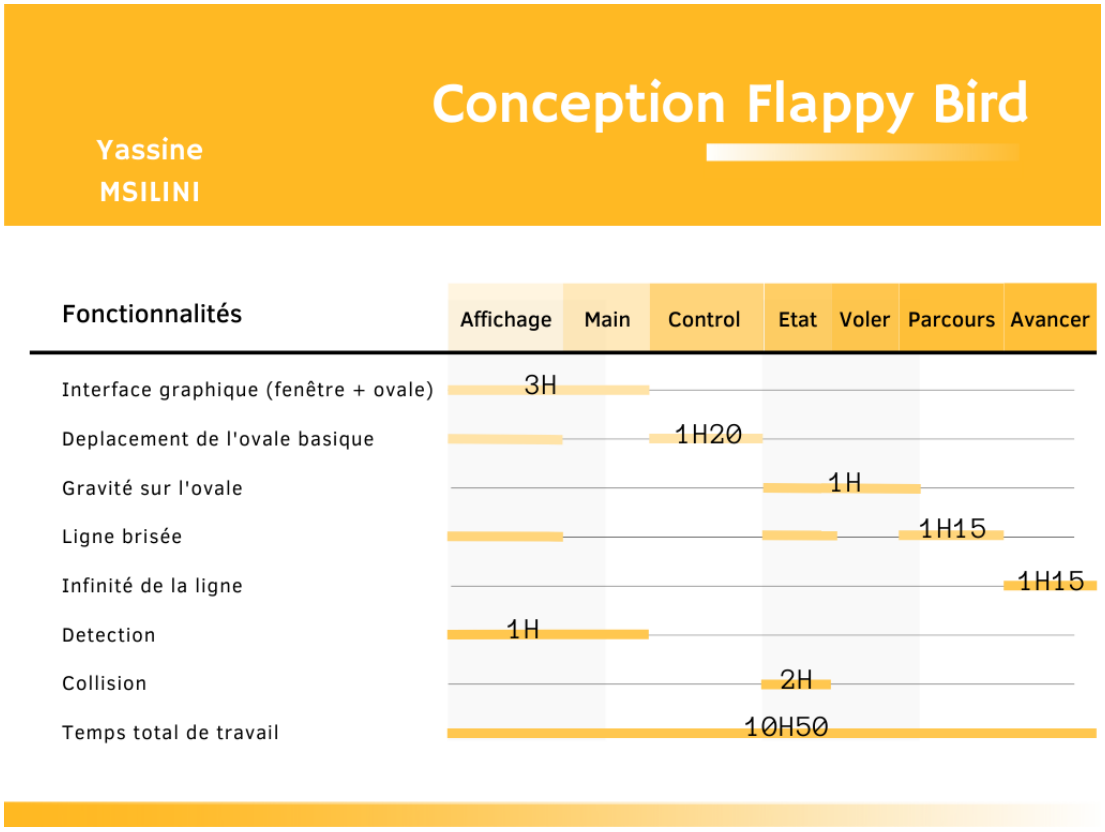
Le temps de travail estimé : 3h30

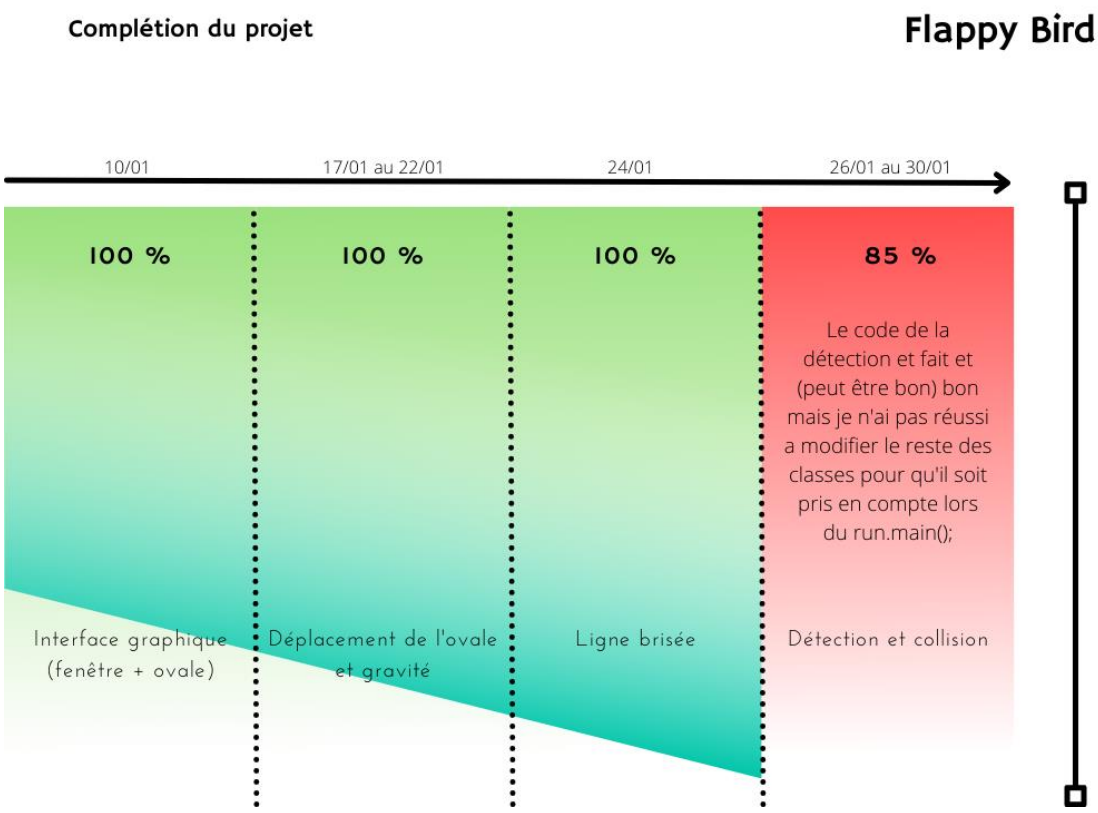
— Analyse du problème : 30 min

— Conception de la detection : 1h

— testPerdu() et test sur l'affichage : 2h

Diagramme de Gantt de la construction du projet :





Conception générale :

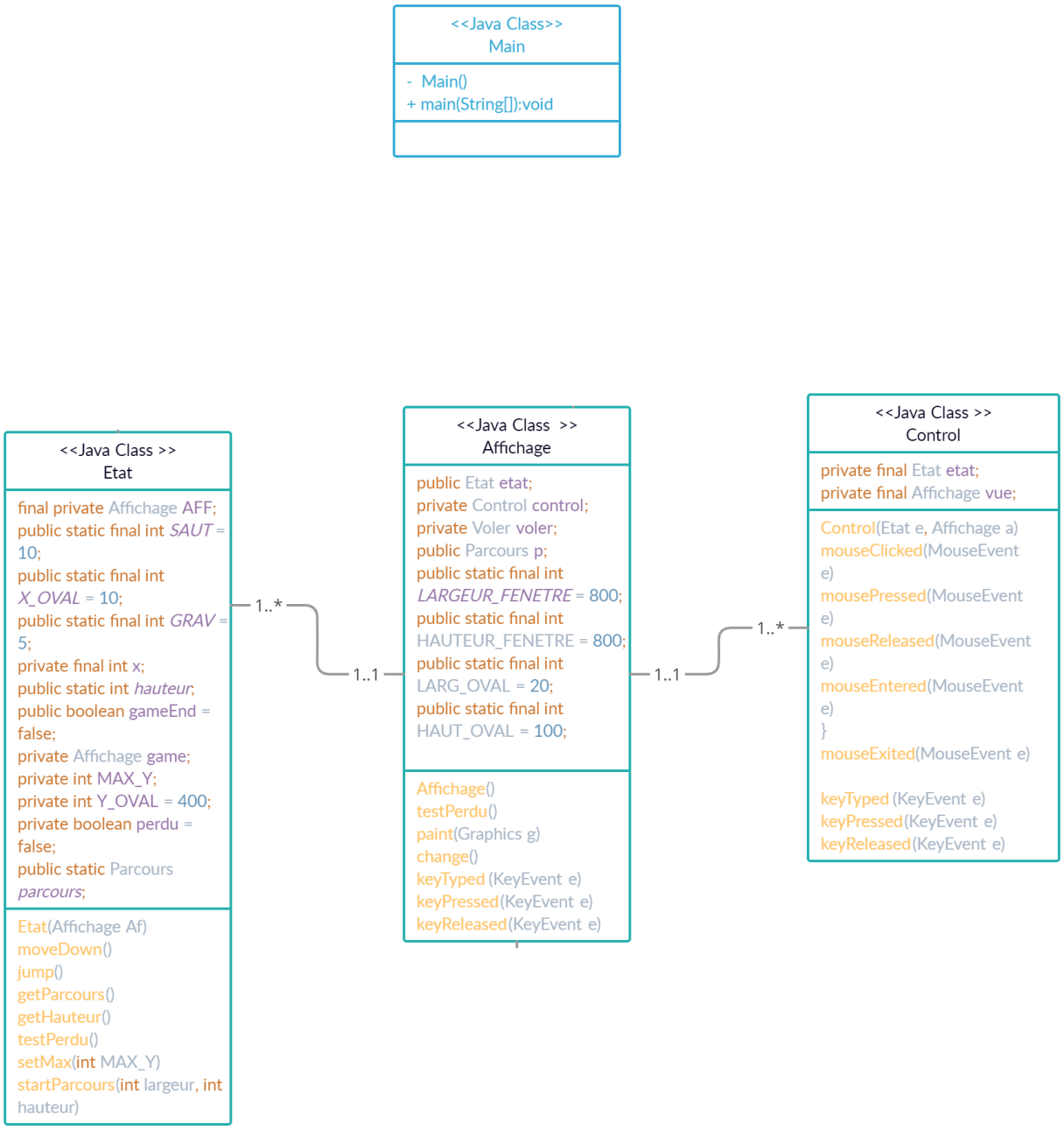
Nous avons adopté le motif MVC pour le développement de notre interface graphique pour plusieurs raisons logique :

Le MVC permet un travail propre et efficace sur les interfaces graphiques avec notamment la modification de l'environement produit après execution du programme (dans les jeux notamments) donc l'interface graphice produit par Affichage doit pouvoir subir des adaptations par rapport mouvement fait par l'ovale et la ligne brisée d'où l'utilisation d'un Controleur, Vue mais aussi Modele car nous allons à la fin du projet implémenter une interface graphique en 2D.

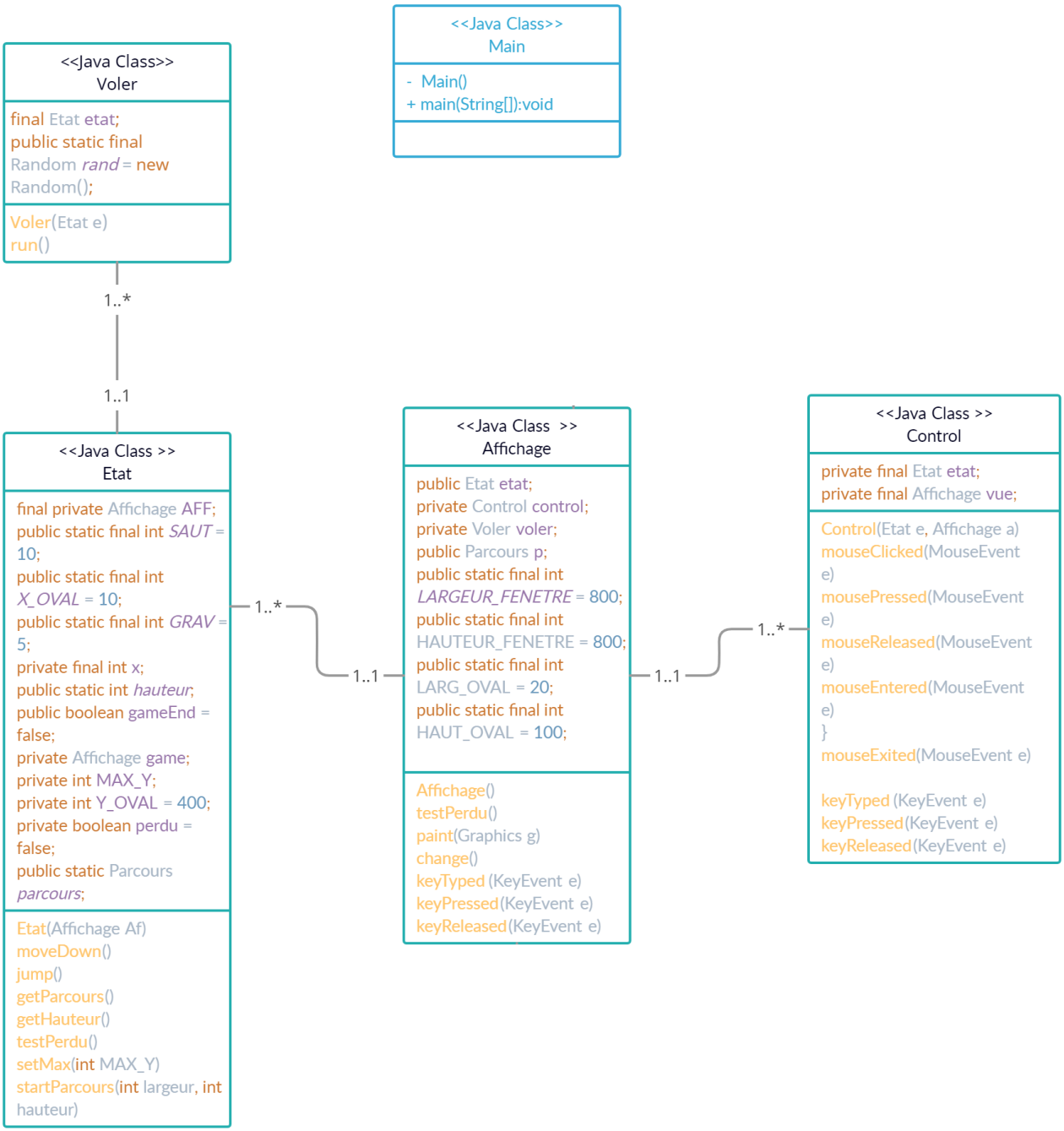
Conception détaillée :

Pour coder la fenêtre et dessinez l'ovale nous aurons besoin de l'API Swing et plus précisement de la classe JPanel les dimensions de la fenêtre et de l'ovale seront définit grâce aux variables suivantes

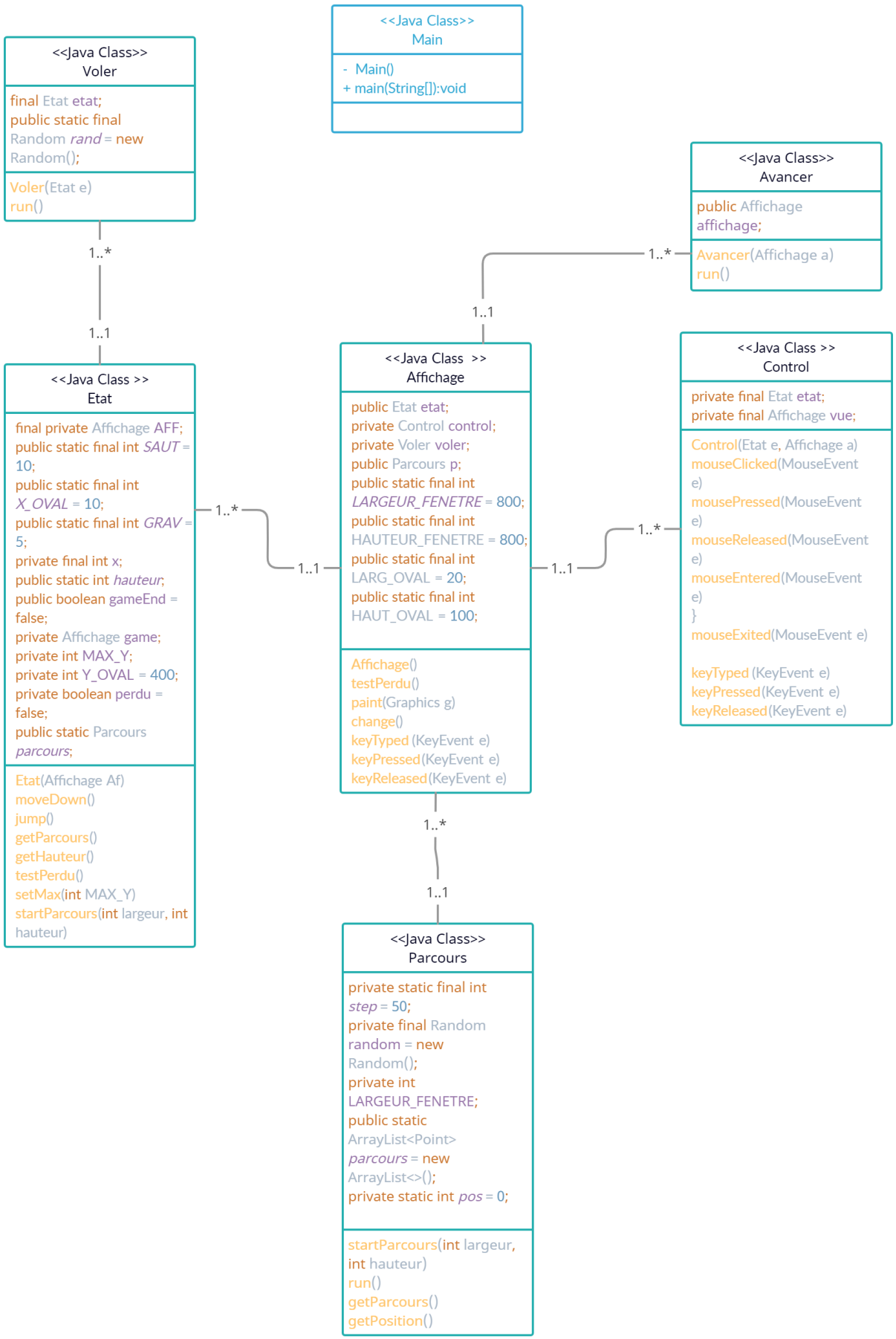
```
LARGEUR_FENETRE = 800;
HAUTEUR_FENETRE = 800;
LARG_OVAL = 20;
HAUT_OVAL = 100;
```



Pour implémenter la fonctionnalité du deplacement de l'ovale on avait besoin de la classe Java.thread . En effet la méthode run de notre classe Voler qui héritera de Thread contiendra une boucle infinie qui permettra de mettre à jour la position de l'ovale.



La ligne brisée sera représenter par la classe parcours qui s'occupe de la logique a travers plusieurs methodes et la ligne brisée infinie se fera grace à l'utilisation de Thread et l'actualisation de l'affichage.


<<Java Class>>
Main
- Main()
+ main(String[]):void
<<Java Class>>
Voler
final Etat etat;
public static final Random rand = new Random();
Voler(Etat e)
run()
<<Java Class>>
Avancer
public Affichage affichage;
Avancer(Affichage a)
run()
1..*
1..1
1..*
1..1
<<Java Class >>
Etat
final private Affichage AFF;
public static final int SAUT = 10;
public static final int X_OVAL = 10;
public static final int GRAV = 5;
private final int x;
public static int hauteur;
public boolean gameEnd = false;
private Affichage game;
private int MAX_Y;
private int Y_OVAL = 400;
private boolean perdu = false;
public static Parcours parcours;
Etat(Affichage Af)
moveDown()
jump()
getParcours()
getHauteur()
testPerdu()
setMax(int MAX_Y)
startParcours(int largeur, int hauteur)
1..*
1..1
<<Java Class >>
Affichage
public Etat etat;
private Control control;
private Voler voler;
public Parcours p;
public static final int LARGEUR_FENETRE = 800;
public static final int HAUTEUR_FENETRE = 800;
public static final int LARG_OVAL = 20;
public static final int HAUT_OVAL = 100;
Affichage()
testPerdu()
paint(Graphics g)
change()
keyTyped (KeyEvent e)
keyPressed(KeyEvent e)
keyReleased(KeyEvent e)
1..1
1..*
<<Java Class >>
Control
private final Etat etat;
private final Affichage vue;
Control(Etat e, Affichage a)
mouseClicked(MouseEvent e)
mousePressed(MouseEvent e)
mouseReleased(MouseEvent e)
mouseEntered(MouseEvent e)
}
mouseExited(MouseEvent e)
keyTyped (KeyEvent e)
keyPressed(KeyEvent e)
keyReleased(KeyEvent e)
1..*
1..1
<<Java Class>>
Parcours
private static final int step = 50;
private final Random random = new Random();
private int LARGEUR_FENETRE;
public static ArrayList<Point> parcours = new ArrayList<>();
private static int pos = 0;
startParcours(int largeur, int hauteur)
run()
getParcours()
getPosition()

Documentation utilisateur :

° Prérequis : Java avec un IDE (ou Java tout seul si vous avez fait un export en .jar exécutable)

° 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 faire monter l'ovale.

° Mode d'emploi (cas .jar exécutable) : double-cliquez sur l'icône du fichier .jar. Cliquez sur la fenêtre pour faire monter l'ovale.

Conclusion et perspectives :

Nous avons une fenêtre fonctionnelle avec l'apparition d'un ovale qui est sujet à une gravité vers le bas constante et qu'on faire monter en utilisant les clics d'une souris, la ligne brisé est bien infinie avec une production de point par rapport aux extremités qui est correcte.

Deux gros problème s'impose, tout d'abord la non actualisation du score qui reste bloqué à 1, ensuite la detection de collision, le code a bien été fait mais je n'ai pas reussis à modifier le reste des classes (notamment **Affichage** et **Main**) se qui fait que lorsque le joueur est censé perdre rien ne se produit, voici en dessous le code de la detection qui utilise la formule mathématique de l'interpolation linéaire.

```
public boolean testPerdu() {
    Point p0 = null;
    Point p1 = null;
    double x = Affichage.LARG_OVAL + Affichage.WIDTH/2;
    double haut = this.hauteur;
    ArrayList<Point> points = this.getParcours();
    for (int i = 0; i < points.size()-1; i++) {
        if (points.get(i).x <= x && points.get(i+1).x >= x) {
            p0 = points.get(i);
            p1 = points.get(i+1);
        }
    }
    double pente = (p1.y - p0.y)*1.0/(p1.x - p0.x);
    double b = (p1.x*p0.y - p0.x*p1.y)*1.0/(p1.x-p0.x);
    double y = pente * x + b;
    if (y <= (haut + Affichage.HEIGHT) && haut <= y) {
        return false;
    }
    return true;
}
```

Ci-dessous le resultat que j'ai pu obtenir.

[Flappy Test]