# PCII

# Flappy Bird

## Table des matières



## I - Introduction

Un ovale qui tombe à une vitesse constante, un clic de la souris permet de faire remonter l'ovale d'une certaine hauteur, une ligne défile et est générée au fur et à mesure. Le but du jeu et de suivre la ligne avec l'ovale.

## II - Organisation du travail

| | Interface générale : fenêtre avec un motif ovale qui monte si on clique | Threads Avancer, Voler et RepaintScreen. Génération et animation de la ligne brisée. | Mise en place d'une courbe en remplacement des lignes brisées | Génération et animation des oiseaux |
|---|---|---|---|---|
| Analyse du problème | 15 min | 30 min | 40 min | 20 min |
| Conception | 30 min | 40min | 20 min | 20 min |
| Implémentation | 1h30 | 2h | 2h | 1h |
| Acquisition de compétences | 1h | 1h30 | 1h30 | 30 min |
| Documentation du code et commentaires et production du rapport | 1h30 | 2h | 1h | 1h |

## III - Analyse

- Chute : On affiche une ovale qui tombe continuellement jusqu'au bas de la fenêtre.

- Clic : On attend un clic de la souris sur la fenêtre pour faire remonter l'ovale.

- Avancer : la position de l'ovale (sur la courbe) augmente avec le temps.

- Génération parcours : Une courbe est générée aléatoirement au fur et à mesure que la position de l'ovale augmente.

- Score : le score correspond à la distance parcourue sur la courbe, c'est à dire la position de l'ovale.

- Collisions : quand l'ovale se décroche de la courbe, le jeu s'arrête et on affiche Game Over ainsi que le score.

- Oiseaux : à chaque rafraîchissement de la fenêtre on a une chance qu'un oiseau apparaisse

# IV - Conception

### Modèle MVC

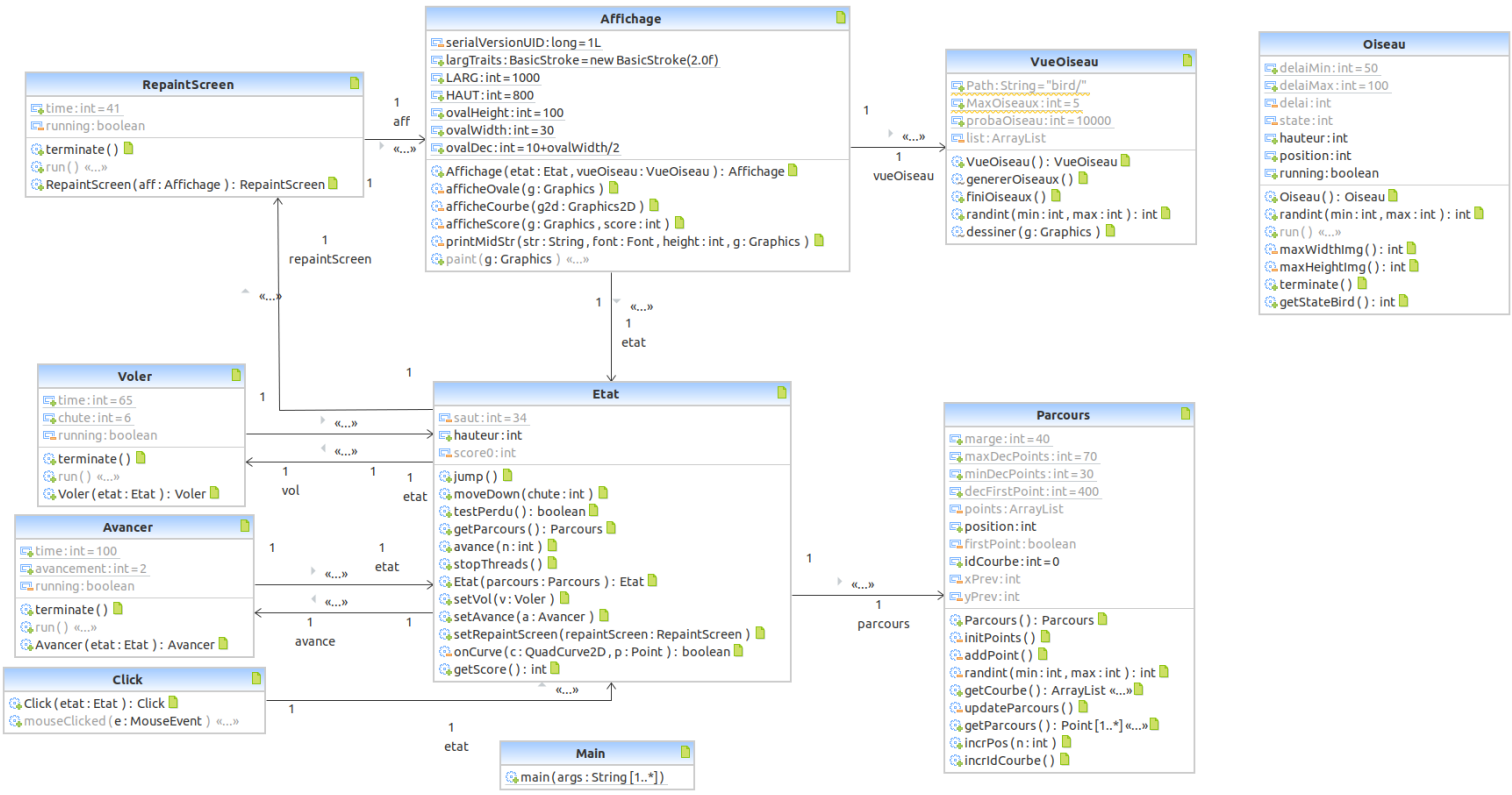
Le modèle MVC mis en place est le suivant :

model : contient les infos sur l'état du jeu, s'est à dire comment les différents éléments doivent être affichés.

view : affichage de la courbe, de l'ovale, des oiseaux et du score. Génération des oiseaux.

control : gestion des clics de la souris, génération et défilement de la courbe, chute de l'ovale, défilement des oiseaux rafraîchissement de l'affichage.

### Diagramme de classes :




### Liste des Threads (control) :

- Click, Voler et Avancer vont modifier l'État en faisant respectivement monter la hauteur de l'ovale, la faire descendre et faire avancer l'ovale sur la ligne brisée.

- Oiseau est un Thread représentant un oiseau dont la position et l'état change à une certaine vitesse.

- RepaintScreen appelle la fonction repaint de l'Affichage

**Algorithme de génération de la courbe :**

```
xPrev ← 0
yPrev ← 0

procédure init:
      TantQue xPrev < LARG
            ajoutePoint ()

procédure ajoutePoint:
      x ← randint(xPrev + minDecPoints, xPrev + maxDecPoints)

      VitesseChute ← distanceChute / tempsChute
      VitesseAvancement ← distanceAvancement / tempsAvancement

      ecartHauteur ← VitesseChute * (x-xPrev) / VitesseAvancement

      yMin ← xPrev - ecartHauteur
      yMax ← xPrev + ecartHauteur

      Si yMin < 0 Alors yMin ← 0
      Si yMax > HAUT Alors yMax ← HAUT

      y ← randint(yMin, yMax)

      newPoint <- Point(x,y)

      midX <- xPrev + (newPoint.x - xPrev)/2
      midY <- yPrev + (newPoint.y - yPrev)/2

      ajoute(points, Point(midX, midY))
      ajoute(points, newPoint)

      xPrev ← x
      yPrev ← y

procédure majPoints:
      Si dernierPoint - position < LARG
      Alors :
            ajoutePoint()
            ajoutePoint()

      Si deuxièmePoint < position
      Alors :
            retirer(premierPoint)
            retirer(premierPoint)

fonction getCourbe: liste de courbes
      res ← liste de courbes vide
      i ← 0
      Tant que i+2 < taille(points):
            A ← points[i]
            B ← points[i+1]
            C ← points[i+2]
            courbe ← Courbe(A,B,C)
            ajoute(res,courbe)
            i ← i+1
      renvoie res
```

**Méthode pour détecter quand l'ovale quitte la courbe :**

La méthode contains pour les courbes de bézier ne permet pas de savoir si un point est sur la courbe mais si ce point est dans l'aire de courbure de la courbe.

J'ai donc calculé si un point est sur une courbe de cette manière:

```
fonction onCurve(x,y) : boolléen
      Pour t allant de 0 à 1 :
            p1 ← c.getP1();
            p2 ← c.getCtrlPt();
            p3 ← c.getP2();
            x  ← (1 – t)² * p1.x + 2t(1 - t) * p2.x + t² * p3.x)
            y  ← (1 - t)² * p1.y + 2t(1 - t) * p2.y + t² * p3.y)
            Si p.x=x ET p.y=y Alors renvoie vrai
      renvoie faux
```

Et si un des points du segment qui coupe horizontalement l'ovale en deux est sur la courbe on n'a pas perdu, sinon on a perdu.

Cette méthode remplace l'ancienne qui avait été mise en place pour calculer la hauteur du point de la courbe au niveau de la position de l'ovale (ligne brisée), indiquée ci-dessous pour information :

```
A ← point juste avant la position de l'ovale
B ← point juste après la position de l'ovale

coefficientDirecteur ← (B.y – A.y) / (B.x – A.x)
hauteurDuPoint ← -A.x * coefficientDirecteur + A.y
```

**Améliorations supplémentaires mises en place :**

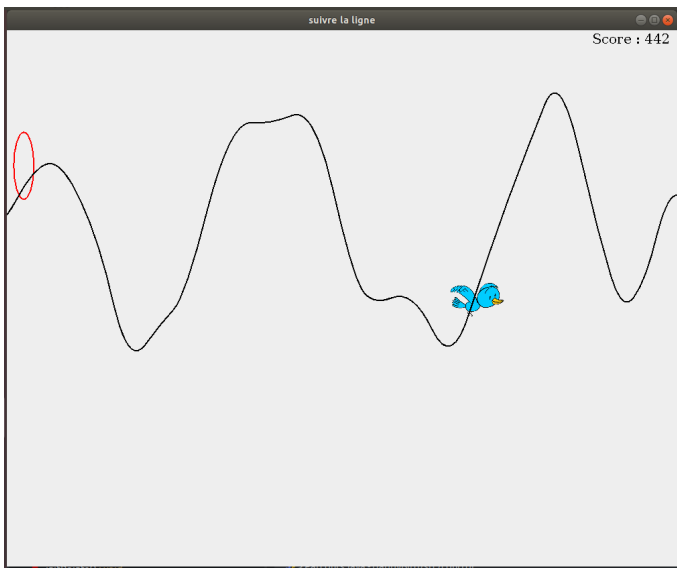
- limitation en hauteur des points générés pour éviter les points collés en haut ou en bas de la fenêtre

- décalage de l'ovale par rapport au bord gauche de la fenêtre

- décalage entre la position de début du jeu et le premier point de la courbe pour que le joueur ait le temps de se placer. Le score est donc pris en compte uniquement à partir du premier point de la courbe

- les oiseaux qui battent plus vite des ailes se déplacent moins rapidement vers la gauche

# V - Résultat




# VI - Documentation utilisateur

Prérequis: Java

Lancez le fichier flappy.jar, cliquez sur la fenêtre pour faire remonter l'ovale de façon à rester sur le trajet de la ligne brisée.

# VII - Documentation développeur

**Javadoc :**

/flappyBird/doc/index.html

**Constantes modifiables:**

Affichage:

- LARG et HAUT : taille de la fenêtre

- ovalWidth et ovalHeight : taille de l'ovale

- ovalDec : distance entre le bord gauche de la fenêtre et le milieu de l'ovale

Etat:

- saut : hauteur du saut de l'ovale lors du clic de la souris

Voler:

- time : temps entre chaque chute de l'ovale

- chute : hauteur de la chute de l'ovale

Parcours:

- marge : limite de hauteur (en haut et en bas) pour la génération de la ligne brisée (pour éviter d'avoir une ligne qui colle les bords)

- maxDecPoints : espace maximum entre 2 points de la ligne

- minDecPoints : espace  minimum entre 2 points de la ligne

- decFirstPoint : décalage entre la position 0 et le premier point de la courbe

Avancer:

- time : temps(en ms) entre chaque avancement

- avancement : distance de chaque avancement

VueOiseaux

- Path : chemin vers les images de l'oiseau

- MaxOiseaux : le nombre maximum d'oiseaux simultanément à l'écran

- probaOiseau : probabilité (à chaque raffraichissement de la fenêtre) qu'on oiseau apparaisse à l'écran

Oiseau

delaiMin et delaiMax : limites du délai entre chaque changement d'état de l'oiseau

## VIII - Conclusion et perspectives

J'ai réalisé une application en Java avec une interface graphique interactive programmée selon le modèle MVC et utilisant le Multi-Threading.

Ce projet m'a permis de mieux comprendre le modèle MVC. J'ai appris à utiliser des Threads pour gérer plusieurs tâches en parallèle, ainsi que javadoc pour construire une documentation à partir de commentaires.

**Évolutions futures envisagées :**

- nuages avec bruit de perlin

- vitesse qui augmente en fonction du score

- tableau de highscores