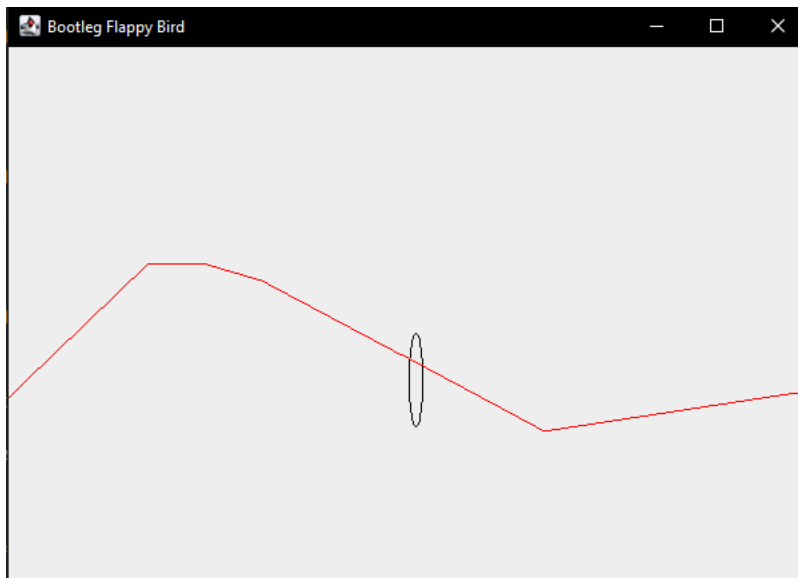


1 Introduction

L'objectif de ce tutoriel est de réaliser un mini-jeu qui reprend le principe de *Flappy Bird* de manière simplifiée. Le joueur sera un anneau autour d'un fil qu'il ne doit pas toucher. Pour ce faire, le joueur peut cliquer sur l'écran pour faire monter un peu l'anneau, et après celui-ci redescend tout seul.

Dans la première partie, nous avons dû réaliser uniquement les sauts de l'anneau lorsque le joueur clique avec la souris. Dans la seconde partie du projet, nous avons introduit le défilement de la ligne brisée ainsi que la génération de celle-ci. Voici à quoi ressemble l'interface graphique du jeu :



La dernière partie du projet a consisté à rajouter la détection de collision entre l'ovale et la ligne brisée, ainsi le joueur meurt s'il touche la ligne. De plus, le score est affiché à la fin du jeu.

2 Analyse Globale

L'ensemble du projet peut être découpé en 3 fonctionnalités :

- L'interface graphique (ligne brisée et ovale)
- Défilement automatique de la ligne
- "Sauts" de l'ovale lorsque le joueur clique avec la souris

La première séance a été consacré à la réalisation de plusieurs sous-fonctionnalités :

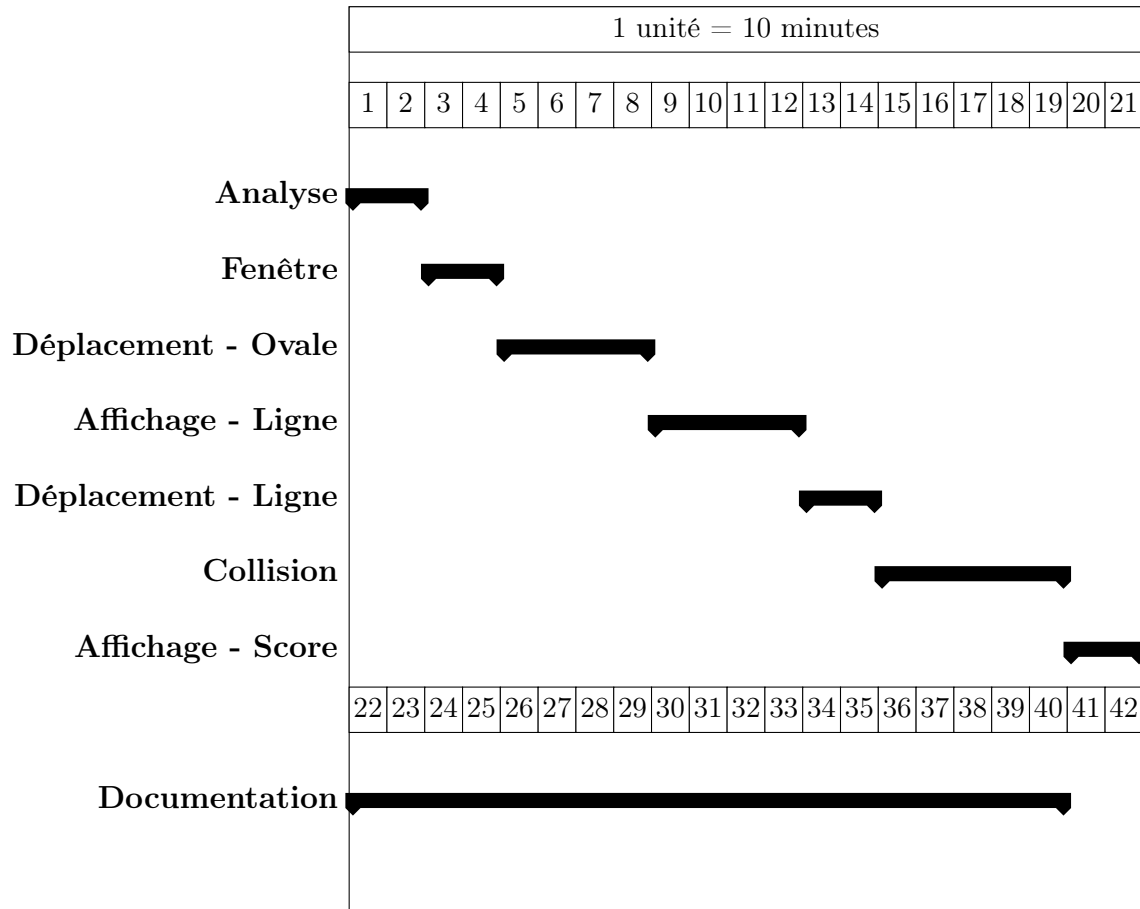
- La création d'une fenêtre
- Affichage de l'ovale

Ainsi que de la fonctionnalité des sauts de l'ovale lorsque le joueur clique avec la souris. La seconde séance, elle, a été consacrée à la réalisation de l'affichage de la ligne brisée, ainsi que le défilement automatique de celle-ci. La dernière séance a été consacré à la réalisation de la détection de collision entre l'ovale et la ligne brisée.

3 Plan de développement

Liste des tâches :

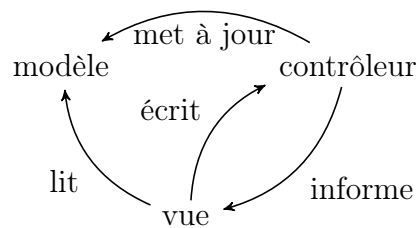
- Analyse du problème (20 minutes)
- Conception, développement et test d'une fenêtre avec un ovale (20 minutes)
- Conception, développement et test d'un mécanisme de déplacement de l'ovale (40 minutes)
- Conception, développement et test de la création et l'affichage de la ligne brisée (40 minutes)
- Conception, développement et test du défilement automatique de la ligne brisée (20 minutes)
- Conception, développement et test de la collision (50 minutes)
- Conception, développement et test de l'affichage du score en jeu et à la mort du joueur (10 minutes)
- Documentation du projet (190 minutes)



4 Conception générale

Ce projet est créé avec le motif MVC.

Ainsi, nous avons la classe *etat* pour le modèle, la classe *controle* pour le contrôleur et la classe *affichage* pour la vue.



5 Conception détaillée

La fenêtre est créée à l'aide de l'API swing et la classe JPanel. Les dimensions de la fenêtre ainsi que les dimensions de l'ovale sont données par des constantes :

- *width* : qui définit la largeur de la fenêtre
- *height* : qui définit la hauteur de la fenêtre
- *ovalWidth* : qui définit la largeur de l'ovale
- *ovalHeight* : qui définit la hauteur de l'ovale
- *x* : qui définit la position de départ de l'ovale sur l'axe des abscisses
- *y* : qui définit la position de départ de l'ovale sur l'axe des ordonnées

Le déplacement de l'ovale est lui géré avec de la programmation événementielle et la classe *MouseListener*. Le saut, lui, est défini par une constante. Lorsque le joueur clique sur la fenêtre, le contrôleur augmente la position y de la constante de saut, et appelle la fonction *repaint* pour que l'affichage se mette à jour avec la nouvelle hauteur.

La ligne brisée est créée à l'aide d'une suite de points via la génération suivante :

```
fonction : initParcours () : void
  x <- largeur_fenetre + largeur_ovale/2
  y <- hauteur_fenetre + hauteur_ovale/2
  Point p <- new point(x, y)
  parcours.add(p)

  tant que(x <= largeur_fenetre)
    x = random entre MAX_LENGTH + p.x et MIN_LENGTH + p.x
    y = createNewY(x, p.x, p.y)
    p = new point(x, y)
  fin tant que
fin fonction

fonction : createNewY(x, oldX, oldY : x) : int
  y <- 0
  faire
    hypotenus = valeur absolue de x - oldX
    sinus = sinus(random entre 0 et MAX_ANGLE)
    hauteur = hypotenus * sinus
    si random entre 0 et 1 == 0
      alors hauteur = hauteur * -1
    fin si
  y = valeur absolue de hauteur + oldY
```

```
    tant que y < hauteur_ovale OU y > hauteur_fenetre - hauteur_ovale  
    renvoyer y  
fin fonction
```

Le calcul de la coordonnée y se fait en fonction de la nouvelle coordonnée x afin d'éviter d'avoir des angles trop obtus et que le jeu soit jouable. Pour ce faire, une règle de la trigonométrie est utilisée : $\sin(\text{angle}) = \frac{\text{oppose}}{\text{hypotenuse}}$. Un nombre aléatoire est pris entre 0 et l'angle maximal possible, défini dans la classe *Parcours*, et la hauteur est ainsi calculée, car avec la valeur x, on peut calculer l'hypoténuse. Pour savoir si on va vers le haut ou vers le bas, un entier aléatoire est pris entre 0 et 1 et on multiplie par -1 si c'est 0 sinon par 1. Quand il n'y a plus assez de points pour former la ligne brisée, un nouveau point est généré avec cette méthode. De plus, quand il y a 2 points hors de l'écran (à gauche) le dernier point est supprimé.

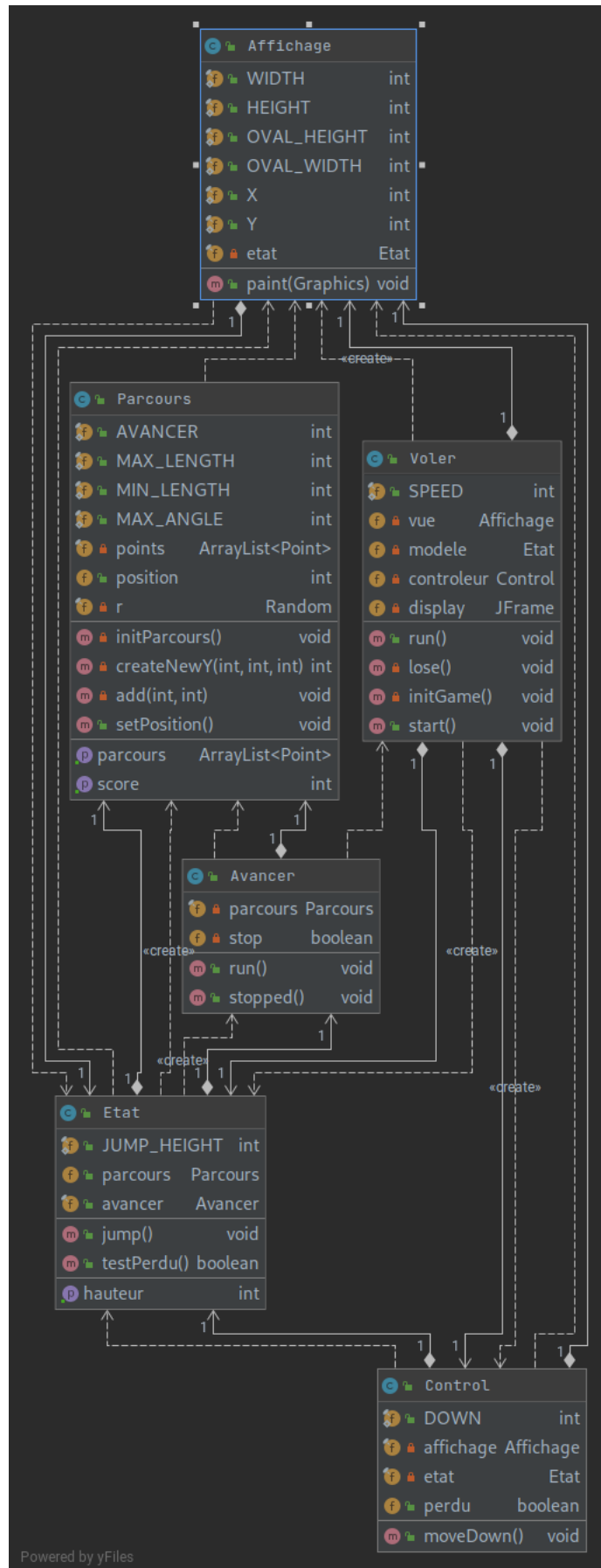
Pour créer l'effet de défilement, on soustrait une constante à l'ensemble des coordonnées x des points du parcours à chaque tick du thread *Avance*.

La détection de la collision se fait de manière simple : on trouve les points de part et d'autre de l'ovale et on détermine l'équation de la droite avec le code suivant :

```
coeff <- (point_apres.y - point_avant.y)/(point_apres.x - point_avant.x)  
ordo_origine <- point_avant.y - coef * point_avant.x  
y <- coef * Affichage.x + ordo_origine
```

L'équation de la droite est simple : il s'agit d'une droite affine (de type $y = ax+b$). On peut déterminer a très facilement (première ligne : le coefficient directeur) à l'aide des deux points. Une fois a obtenu, le calcul de b est trivial vu qu'il s'agit d'une équation à 1 inconnu. On l'applique sur l'un des points et on trouve b, l'ordonnée à l'origine. Avec a et b, on a pu retrouver l'équation de la droite et, il suffit juste de l'appliquer en remplaçant x par Affichage.x (qui est la position x de l'ovale) pour calculer la coordonnée y de la ligne brisée à l'endroit de l'ovale. Il suffit ensuite de vérifier que cette coordonnée y est bien comprise entre la coordonnée y de l'ovale et la coordonnée y + la taille de l'ovale. Sinon, le joueur a perdu car la ligne a été touchée.

Nous avons donc le diagramme de classe suivant :



6 Documentation utilisateur

Pré-requis : Java avec un IDE.

Mode d'emploi : importer le projet dans votre IDE, sélectionner la classe Main puis "Run as Java Application". Cliquez sur la fenêtre pour faire monter l'ovale.

7 Documentation développeur

Toutes les fonctionnalités majeures ont été implémenté, mais des améliorations pourraient être apportées au projet comme un meilleur affichage graphique tel que des éléments de décor, l'affichage du temps ou du score. Ce sont principalement des éléments graphiques car le coeur du projet a été réalisé et il ne peut pas y avoir de modifications de ce côté-là, si ce n'est de l'optimisation de code.

On pourrait aussi penser à rajouter de la musique, un menu d'accueil, avec aussi des paramètres tels que la taille de la fenêtre ou le volume de la musique.

8 Conclusion et perspectives

Le jeu est fini dans son fonctionnement, mais de grosses améliorations graphiques sont possible afin de faire devenir le jeu un vrai jeu.