PCII

Membres :
Fedy Ben Naceur
Répertoire Github du projet : lien

1.Introduction :

Le but du projet est de créer une version simple de Flappy bird,un jeu mobile populaire. Le jeu consiste à éviter les obstacles en se déplaçant sur une ligne brisée,pour cela, le joueur peut cliquer sur l'écran pour faire monter l'ovale, qui redescend ensuite tout seul. Notre version du jeu sera implémenter en Java en utilisant comme représentant du oiseau un ovale,le joueur utilisera la souris pour cliquer a chaque fois il souhaite monter l'ovale .

2.Analyse globale :

Notre projet contiendra trois éléments principaux qui seront implémenter suivant le modèle MVC (model/vue/control) . Les trois fonctionnalités principales sont l'interface graphique qui représentera le jeu avec un ovale qui sera l'oiseau,une ligne brisée qui sera défilée automatiquement et la réaction de l'ovale aux clics de la souris de l'utilisateur qui devrait monter et redescendre.
La première partie s'intéresse a deux fonctionnalités du projet :

- création d'une fenêtre dans laquelle est dessiné l'ovale .
- déplacement de l'ovale vers le haut lorsqu'on clique dans l'interface. Les deux fonctionnalités listées ci dessus sont assez simple à réaliser et ne nécessitent que des connaissances basiques de l'API Swing . Elles sont aussi prioritaires et doivent être implémenté en premier .

La deuxième partie du projet s'intéresse principalement à deux autres fonctionnalités du projet :

- le mécanisme de descente qui permet de simuler la gravité dans cet environnement et permet au oiseau de voler .
- la ligne brisée que l'ovale se de laquelle l'ovale doit se promener .

La troisième partie sera consacrée à la gestion des collisions et l'affichage du message de la fin de partie .

3.Plan de développement :

Pour la première on aurait besoin de l'interface graphique ainsi que l'interaction entre l'utilisateur et le programme.

Interface graphique :

Le temps de travail estimé : 1h

- Analyse du problème : 15 min
- Conception,développement et test d'un fenêtre avec un ovale : 30 min

— Acquisition de compétences en Swing (60 min)
— Documentation du projet (60 min) : Cette partie se décompose en écriture de commentaires et la rédaction de la première partie du rapport du sujet .

Mécanisme de déplacement de l'ovale

Le temps de travail estimé : 20min
— Analyse du problème : 5 min
— Conception, teste : 5 min
— Découvrir l'interface MouseListener (5min min)

Pour la deuxième partie du projet on s'intéresse à l'implémentation du mécanisme de vol du oiseau ainsi que le défilement de la ligne brisée .

Le vol du oiseau

— Analyse du problème : 20 min
— Conception, teste : 15 min
— Acquisition de compétences en Java.thread : 10 min
— Documentation du projet (60 min) : Cette partie se décompose en écriture de commentaires et la rédaction de la deuxième partie du rapport du sujet .

La ligne brisée

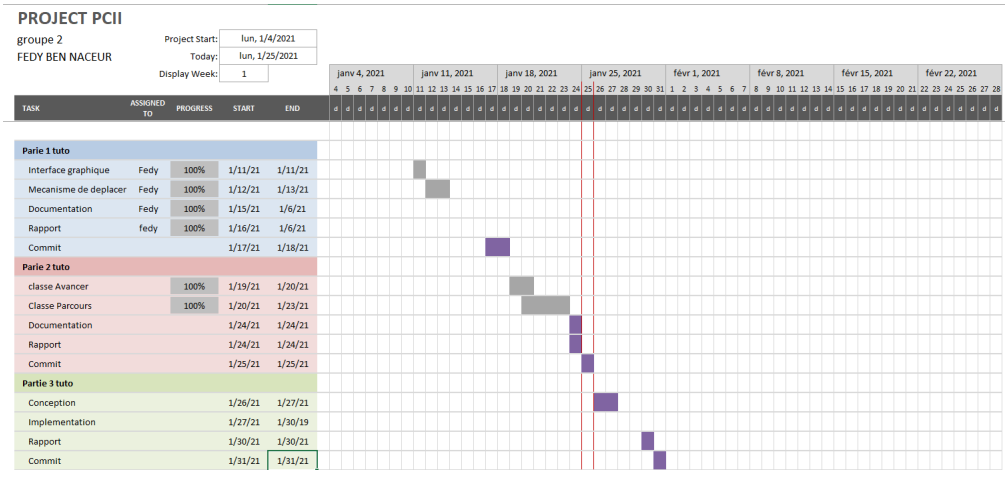
— Analyse du problème : 45 min
— Conception, teste : 1h30 min

La dernière partie s'est avéré un peu difficile puisque on gérait des indices qui étaient mis à jour de manière continue .

Gestion de la collision

— Analyse du problème : 1 jours
— Conception et test : 5 jours

Le diagramme de Grannt



4.Conception générale :

Puisque notre programme gère une interface graphique , nous avons adopté le design pattern MVC .
Notre fenêtre sera gérer par la vue qui s'occupera de l'affichage des éléments à dessiner sur l'écran l'ovale compris.Tandis que le mouvement de l'ovale suite à des clics de la souris sera gérer par le control qui implémente un listener et notifie la vue des changements qui ont eu lieu tout en modifiant les variables du modèle. Le modèle contiendra aussi la classe parcours,et la classe Avancer et Voler viendront s'ajouter au Control .

5.Conception détaillée

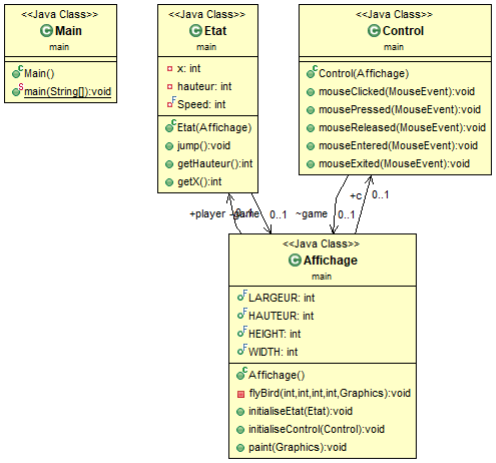
Fenêtre avec un ovale

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

- LARGEUR = 600 largeur de la Panel
- HAUTEUR = 400 hauteur de la Panel
- HEIGHT=100 hauteur de l'éclipse
- WIDTH=50 largeur de l'éclipse

Déplacement de l'ovale

nous utilisons la programmation événementielle avec la classe MouseListener et la hauteur est définie dans une constante Speed = 20 .



Le mécanisme de vol du oiseau

Pour implémenter cette fonctionnalité on aurait 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 a jour la position du oiseau . Cette classe contiendra deux attributs :

- player : de type Etat qui permet d'accéder à l'ovale et mettre à jour sa position .
- time_to_sleep = 800 : qui représente l'écart entre chaque mise à jour de la position de l'ovale .

La ligne brisée

La ligne brisée sera représenter par la classe parcours qui s'occupe de la logique a travers les méthode initialisePoints /addPoint/getParcours. Cette classe définit les attributs suivants :

points : la liste des points qui définissent notre ligne

game : la liaison entre le parcours et l'affichage

pos : qui indique le score du joueur

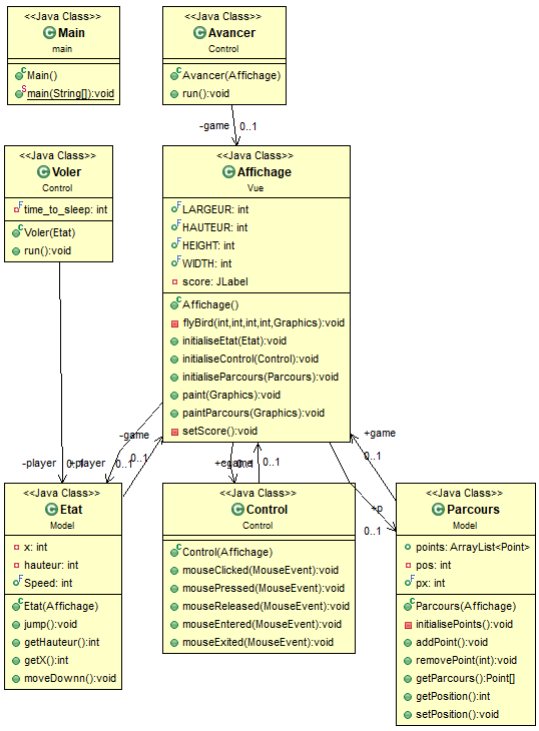
px : l'attribut qui indique avec combien incrémenter la position

Algorithme d'initialisation de la ligne brisée :

Procedure initialisePoints(ArrayList a,Afficahge a) :
prevx <- 10 + (game.WIDTH / 2) ;
prevy <- game.HAUTEUR - (game.HEIGHT / 2) ;
points.add(new Point(prevx, prevy)) ;
aff <- 0 ;
y <- prevy ;
Tantque (prevx <= game.LARGEUR)
x <- r.nextInt((prevx + 40) - (prevx + 20)) + (prevx + 20) ;
Si (aff !=1)
y <- r.nextInt(game.HAUTEUR) ;
aff ++ ;
Sinon
y <- r.nextInt((prevy + 2) - (prevy + 1)) + (prevy + 2) ;
if (y>game.HAUTEUR)
y = prevy ;
aff = 0 ;
finSi
points.add(new Point(x, y)) ;
prevx = x ;
prevy = y ;
Fin Tantque

Diagramme de classe pour la deuxième Partie :

Voici un diagramme mis à jour pour cette étape du projet :



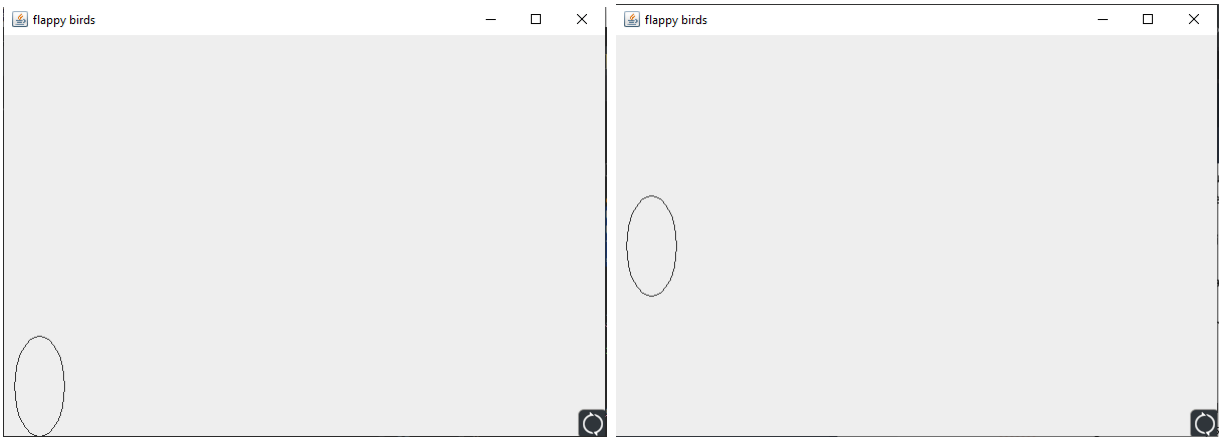
Gestion de la collision

Code de detection de collision :

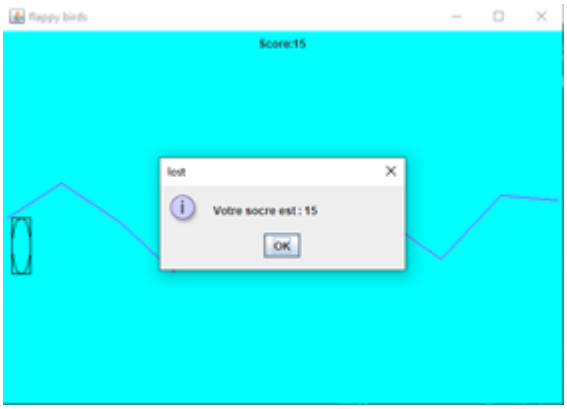
```
public void testPerdu() {
        Point[] tmp = game.p.getParcours();
        Rectangle rect1 = new Rectangle(this.x, this.hauteur ,
    this.game.WIDTH, this.game.HEIGHT);
        boolean res = true ;
        for(int i = 0 ;i<game.p.points.size()-2;i++) {
                Line2D line2 = new Line2D.Float(tmp[i].x,tmp[i].y,
                                tmp[i+1].y,tmp[i+1].y);
                        if (line2.intersects(rect1)) {
                                res = false  ;
                                break ;
                        }
                }
        if (res == true ) {
                        gameEnd = true ;
                        JOptionPane.showMessageDialog(game,
                        "Votre socre est : " + game.p.getPosition(),
                        "lost",JOptionPane.INFORMATION_MESSAGE);
                }

        }
Pour g rer les collisions on v rifie que le rectangle qui en-capsule l'
```

6.Résultat :

Partie 1 :



Partie 2/3 :



7.Documentation utilisateur :

- Pré requis : Java avec un IDE
- 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.

8.Documentation développeur :

Pour avoir une bonne compréhension du projet je propose que vous commencez par explorez la classe Parcours et Affichage . La méthode main se trouve dans le package main dans la classe Main. Pour avoir un peu de contrôle sur le jeu la modification de certaines variables est nécessaire :

L'oiseau :

Pour changer le comportement du oiseau on pourra changer les variables suivantes :

- Speed(px) : cette variable définit la vitesse avec laquelle l'oiseau fait ces sauts[Class Etat] .

— Time_to_sleep(ms) : cette variable définit le temps d'attente pour mettre à jour la position du oiseau [Class Voler].
— Height Width(px) : ces variables définissent la taille de l'ovale [Class Affichage].

ligne Brisée :

On pourra modifiez la vitesse de défilement de la ligne brisée en changeant :
— Time_to_sleep(ms) :définit le temps d'attente avant de faire avancer la file[Class Avancer].
— pos :définit la distance avec laquelle la ligne avance[Class Parcours].

Il nous reste encore la gestion des collisions à implémenter et la modification du rendu graphique pour avoir une expérience de qualité .

9.Conclusion et perspectives :

A la fin de cette partie on a réussi à implémenter une fenêtre dans laquelle est dessiné l'ovale , le déplacement de l'ovale vers le haut lorsqu'on clique dans l'interface ,la gestion de la ligne brisée et la gestion des collisions. Les deux dernières parties se sont avérée un peu compliquées, la gestion du défilement de la ligne brisée a nécessité beaucoup de temps , pour s'attaquer à ces problèmes nous avons utilisé le debuggeur intégré d'éclipse qui permet de suivre étape par étape l'état des variables . Notre projet a une marge d'amélioration importante en effet les graphiques du jeu peuvent être encore améliorer, on peut par exemple ajouter une vraie image d'oiseau au lieu de l'ovale , rendre la ligne plus courbée .. Ce projet nous a permis d'acquérir des compétences dans l'utilisation de l'API Swing et des threads ainsi que développer une bonne compréhension du modèle MVC .