**Rapport de projet PCII**

1. **Introduction :**

L’objectif de ce projet est de réaliser une version simplifiée d’un jeu très populaire sur smartphone et tablette « flappy bird ». Ici l’oiseau sera remplacé par un ovale se déplaçant le long d’une ligne brisée. Le but du jeu étant que le l’ovale ne sorte pas de la ligne. Le joueur peut cliquer afin de faire monter l’ovale qui redescend ensuite instantanément sous l’effet de la gravité. Graphiquement cela peut ressembler à cela :



2 . **Analyse globale :**

Notre projet se découpe en trois grandes parties :

* L’interface graphique qui représente la vue du joueur où se trouve l’ovale et la ligne
* Le défilement automatique de la ligne brisée donnant l’impression que l’ovale se déplace
* Ainsi que l’interaction entre le joueur et l’ovale (l’ovale monte aux clics de souris)

La première partie de ce projet se concentre sur la création d’une fenêtre graphique dans laquelle nous représenterons l’ovale ainsi que l’unique interaction du joueur, c’est-à-dire la montée de l’ovale lorsque l’utilisateur effectue un clic de souris. Ces deux sous partie représente les fondations du projet et sont donc prioritaires. Bien qu’elles soient relativement simples à réaliser.

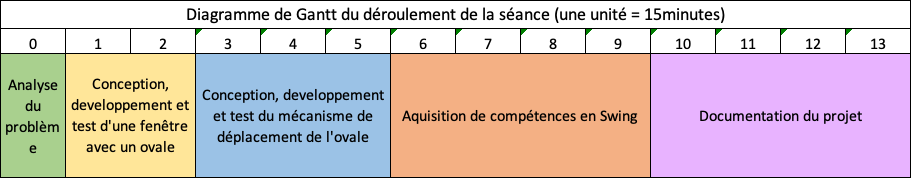
La seconde partie du projet a pour objectif de générer une ligne brisée aléatoire qui défile et représente le parcours du jeu. Mais aussi la retombée de l’anneaux de façon constante entre chaque clique de souris. Ces deux fonctionnalités sont essentielles pour le jeu et sont donc prioritaires. Elles sont de difficulté moyenne en raison de la gestion des threads afin que les actions s’effectuent en parallèle. Enfin l’algorithme de génération de la courbe est assez difficile à imaginer.

La troisième partie du projet a pour vocation de prendre en charge les collisions de l’ovale avec la ligne brisée. Il faut pour cela définir sous quelle conditions l’ovale entre en collision avec la ligne brisée. Il faut ensuite tester cette condition et interrompre le jeu. Enfin Lors d’une défaite il faut arrêter l’exécution des threads et afficher une nouvelle fenêtre affichant le score.

3 . **Plan de développement :**

Première partie :

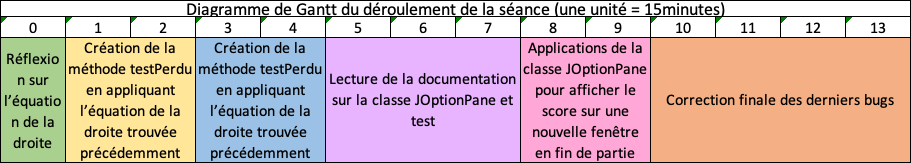
Liste des taches :

* Analyse du problème (15 min)
* Conception, développement et test d’une fenêtre avec un ovale (30 min)
* Conception, développement et test du mécanisme de déplacement de l’ovale (45 min)
* Acquisition de compétences en Swing (60min)
* Documentation du projet (60 min)

Deuxième partie :

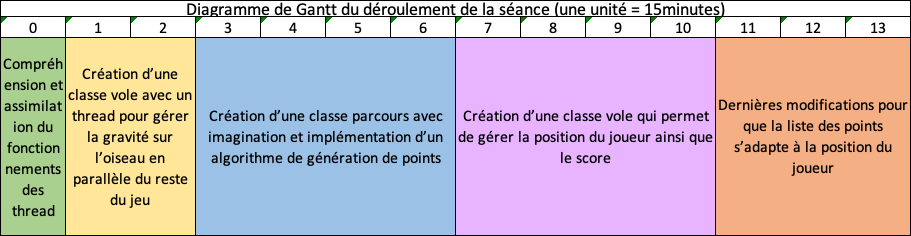
Listes des taches :

* Compréhension et assimilation du fonctionnements des thread (15min)
* Création d’une classe vole avec un thread pour gérer la gravité sur l’oiseau en parallèle du reste du jeu (30min)
* Création d’une classe parcours avec imagination et implémentation d’un algorithme de génération de points (60min)
* Création d’une classe vole qui permet de gérer la position du joueur ainsi que le score (60min)
* Dernières modifications pour que la liste des points s’adapte à la position du joueur (45min)



Troisième partie :

* Réflexion sur l’équation de la droite (15 min)
* Création de la méthode testPerdu en appliquant l’équation de la droite trouvée précédemment (30 min)
* Modification des fonctions run des threads des classes voler et avancer pour les interrompre lorsque l’on perd. (30 min)
* Lecture de la documentation sur la classe JOptionPane et test (45 min)
* Applications de la classe JOptionPane pour afficher le score sur une nouvelle fenêtre en fin de partie (30 min)
* Correction finale des derniers bugs (60 min)



4 . **Conception générale :**

Première partie :

Pour le développement de notre interface graphique nous avons reproduit le motif MVC (modèle vue contrôleur).

Notre première sous fonctionnalité au cœur de cette première partie, la fenêtre graphique fait partie de la vue car elle représente une partie de l’affichage. En effet elle définit le cadre dans lequel se déroule le jeu à la fois pour borner l’affichage mais aussi pour d’délimiter le champ d’action du joueur.

La seconde sous fonctionnalité s’inscrit dans la section control car elle fait le lien entre l’interface graphique et le reste du projet. Le mouvement de l’ovale est à cheval entre l’affichage et la modification de l’état du modèle. En effet lorsque l’utilisateur clic alors l’état de l’ovale (centre de l’ovale décalé vers le haut de l’image) est modifié et l’affichage est réactualisé.

Deuxième partie :

Ici l’objectif majeur c’est la création, l’affichage et le défilement de la ligne brisée. Et cela s’inscrit toujours dans un motif MVC.

Nous avons ici implémenté 3 sous fonctionnalités :

La première sous fonctionnalité permet la retombée de l’anneaux entre les clics de souris. Pour cela une classe voler qui hérite de thread a été créer. La méthode « run » du thread permet à notre action de s’effectuer en parallèle du reste de jeu. Une gravité modifie à intervalle régulier la position y de l’ovale pour que celui-ci soit déplacé d’un certain nombre de pixels vers le bas.

La seconde sous fonctionnalité permet la génération d’un ensemble de points afin de créer une ligne brisée. Cela s’effectue dans la classe parcours qui a pour but de généré une liste de points pour la ligne brisée en vérifiant que l’inclinaison de la pente de la ligne ne soit pas trop forte par rapport à la vitesse de retombée de l’anneaux.

Enfin la troisième et dernière sous fonctionnalité permet de gérer l’avancement de la position et du score. Pour cela une classe avancer a été créer avec là encore un héritage de la classe thread pour que l’avancement de la position se fasse en parallèle du reste du jeu. La position est incrémentée de façon constante d’un certain nombre de pixels défini par une constante.

Troisième partie :

Le but de cette dernière partie est la gestion de la fin du jeu :

Cette partie se compose de deux sous fonctionnalités :

La première consiste à faire une méthode testPerdu qui étant donné la position de l’ovale par rapport à la courbe renvoie false si l’ovale n’est pas en contact avec la courbe ou en dehors et true sinon. Cette méthode doit permettre lorsque l’on touche la ligne avec l’ovale de lancer les actions de fin de partie.

La seconde sous-fonctionnalité permet de définir ce qui se passe lors d’une fin de partie. C’est-à-dire interrompre les threads et afficher le score dans une nouvelle fenêtre.

5 . **Conception Détaillé :**

* Concernant la fenêtre avec l’ovale nous utilisons l’API Swing et la classe Jpanel. Les dimensions de l’ovale ainsi que celles de la fenêtre sont définies par des constantes. Pour la fenêtre on a width et height. Et pour l’ovale on a x et y qui correspondent à la position de son centre.
* Pour le déplacement de l’ovale nous utilisons le principe de la programmation évènementielle avec la classe MouseListener et la hauteur définie comme constante. Ici c’est la méthode mouseClicked qui nous intéresse pour permettre la monté de l’ovale à chaque clic de la souris. La constante JUMP définie ici l’amplitude du saut qu’effectue l’ovale. En effet cette constante est ajoutée à la position, en y de l’ovale, à chaque clic de souris.
* La retombée de l’ovale entre les clics de souris est gérée continuellement à l’aide d’un thread dans la classe voler. La méthode « run » applique un délai en milliseconde entre chaque chute de hauteur de l’ovale de la gravité. Un appel à la méthode move de la classe état pour décrémenter la valeur en y du centre de l’ovale d’un certain nombre de pixels vers le bas défini par la constante gravitée.
* La génération de la ligne brisée est gérée par la classe parcours. Un algorithme de génération de points aléatoire avec abscisse croissante est implémenté. Ce dernier prend en charge les bornes de la fenêtre. Il vérifie également que la pente n’est pas trop impotente par rapport à la vitesse de saut (constante jump) et de retombée de l’ovale (constante gravity). Enfin cela prend en compte la position du joueur et permet dans la méthode « incrementPos » d’incrémenté la position et de généré un nouveau point pour la suite de la ligne brisée.

Voici le pseudo code pour la génération de la ligne brisée :

Int xold = 0;

Int yold = 0;

Int xnew = 0;

Int ynew = 0;

Tant que xnew < largeur de l’écran

xnew = xold + (nombre aléatoire)

ynew = yold +(nombre aléatoire strictement inférieur à la hauteur de la fenêtre)

bool bonnePente = vérification de l’inclinaison de la pente (elle ne descend pas plus vite que la gravité et ne monte pas plus vite que la capacité du saut)

Tant que bonne Pente

xnew = xold + (nombre aléatoire)

ynew = yold +(nombre aléatoire strictement inférieur à la hauteur de la fenêtre)

bonnePente = vérification de l’inclinaison de la pente

fin tant que

Point pi = new point(xnew, ynew)

Ajout du point à la liste

xold = xnew

yold = ynew

fin tant que

xnew += xold + (nombre aléatoire)

ynew = yold +(nombre aléatoire strictement inférieur à la hauteur de la fenêtre)

Point pi = new point(xnew, ynew)

Ajout du point à la liste

* La classe avancer utilise un thread afin de gérer l’incrémentation de la position continuellement en parallèle. Elle effectue après un délai un appel à la méthode « incrementPos » de la classe parcours pour modifier sa position. La méthode paint est mise à jour pour permettre l’affichage du score qui est en fait l’état de la position et de dessiner des lignes entre deux points consécutifs du tableau des points de la classe parcours grâce à la méthode drawPolyline.
* La méthode testPerdu de la classe état permet de tester si l’ovale est autour de la ligne brisée ou en dehors. En effet grâce à l’équation d’une fonction affine : f(x) = ax + b on peut déterminer la hauteur de la ligne à l’endroit où se situe l’ovale. Ici a correspond à la pente calculée avec la formule (y2 – y1) /(x2-x1) avec 1 et 2 respectivement le premier et deuxième point de la liste. Et B étant l’ordonnée à l’origine calculé dans notre cas part y1 – (a \* x1). Si la valeur ainsi calculée n’est pas comprise dans les bornes de l’ovale alors on a perdu et la méthode rend true.
* Ensuite on modifie les threads des classes voler et avancer pour que la boucle de la méthode run s’interrompe lorsque testPerdu rend true. Pour cela on fait appel au méthode end qui modifie la valeur de la variable stopped à true et interrompt la boucle while de run.
* Enfin lorsque testPerdu est true on affiche dans le main une nouvelle fenêtre grâce à la classe JOptionPane qui permet d’indiquer au joueur qu’il a perdu en lui affichant son score grâce à la méthode getScore.

6 . **Résultats :**

Photo de test représentant la fenêtre de fin de partie :

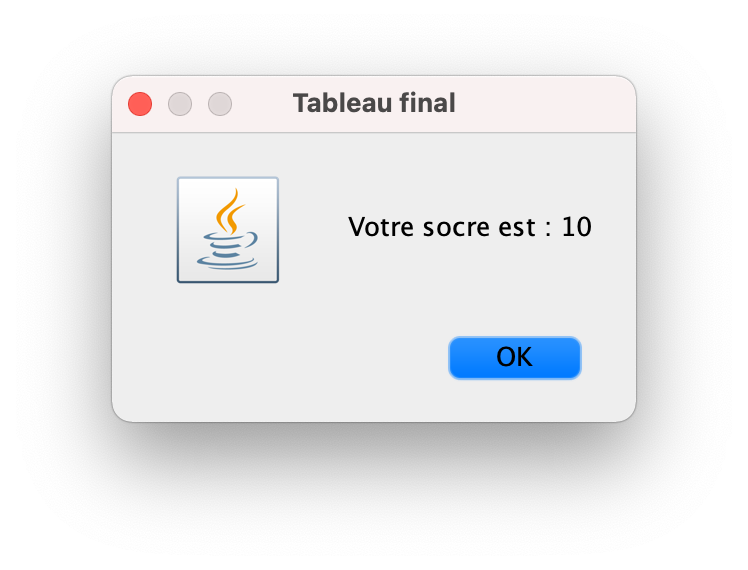
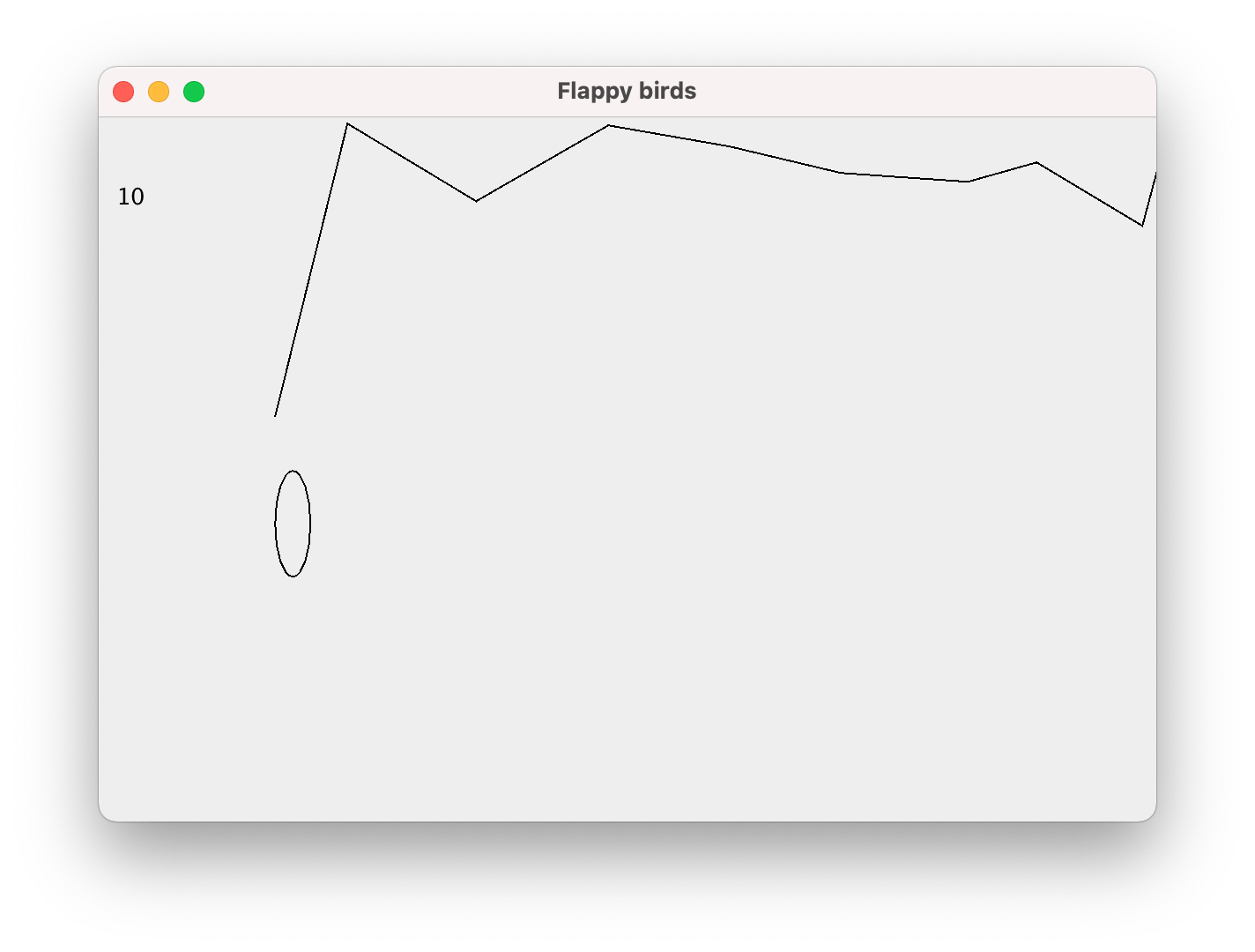


Photo d’un exemple de partie :



7 . **Documentation utilisateur :**

Prérequis : Java avec l’IDE eclipse.

Mode d’emploi : créer un nouveau projet java et déplacer le fichier source à la base du projet. Sélectionner le projet dans la liste à gauche et cliquer sur le bouton « run main » en haut à droite. Une fenêtre s’ouvre et vous pouvez commencez à jouer. Pour fermer le jeu il suffit de fermer la fenêtre.

8 . **Documentation développeur :**

* La classe main est dans le package main
* Les classes affichage état et control sont les plus importante de ce projet et forme la structure MVC. L’état regroupe les informations sur tous les objets du projet. La modification des constante JUMP gravity sont déjà un moyen de modifier la vitesse de saut ou l’ampleur de la gravité. Mais toutes les constantes de cette classe peuvent être modifier pour changer le comportement des objets.

L’affichage décrit l’apparence des objets. Modifier les constantes de cette classe peut modifier le visuel du jeu.

* Les fonctionnalités qui mériteraient d’être implémenté serait de transformer la ligne brisé en courbe lisse avec l’aide des courbes de Béziers. On pourrait également transformer l’anneau en oiseau grâce à un outil de manipulation d’image comme GIMP.

9 . **Conclusion et perspectives :**

Le projet est maintenant terminé. Les bases du jeu sont posées. Il ne reste plus qu’à améliorer l’aspect visuel pour un rendu plus esthétique et à assurer la maintenance.