Rapport de Projet

1 Introduction

L'objectif de ce projet est de réaliser une version simple d'un mini-jeu inspiré de *flappy bird*. En effet, au lieu de faire avancer un oiseau dans un environnement à défilement horizontal en tapotant sur l'écran tactile, tout en évitant des obstacles (tuyaux); il faut faire avancer un ovale, dans un environnement à défilement horizontal, le long d'un chemin représenté par une ligne. Le joueur clique également sur l'écran pour faire monter et descendre l'anneau d'une hauteur constante. Le but du jeu pour le joueur est de faire avancer l'anneau le plus loin possible sans qu'il rentre en contact avec la ligne. La difficulté du jeu vient de l'aspect de la ligne qui n'est pas droite.

Dans une première partie, nous avons construit une interface interactive en commencant par l'interface graphique (Figure 1) puis en réalisant des sauts à l'anneau en utilisant la programmation évènementielle. La seconde partie est sur la programmation concurrente. Le but est de faire perdre de l'altitude à l'anneau quand il n'y a pas d'interaction avec l'utilisateur et d'introduire un mouvement à une ligne de chemin qui sera brisée. Pour finir, il faut détecter les collisions entre la ligne et l'anneau et améliorer l'interface graphique en ajoutant des éléments de décors.

2 Analyse globale

L'ensemble du projet peut être découpé en 3 fonctionnalités : - L'interface graphique - Défilement automatique de la ligne - "Sauts" de l'ovale lorsque le joueur clique avec la souris

L'interface graphique peut être réalisé à l'aide de certaines fonctionnalités telles que la création d'une fenêtre dans laquelle un ovale noir et une ligne brisée bleue sont dessinés. Pour plus d'esthétisme et d'impression de défilement, un fond a été rajouté ainsi qu'un oiseau (voir Figure 1). Pour aider l'utilisateur, l'ovale a été laissé en plus de l'oiseau. De plus, il a fallu implémenter une fenêtre contextuelle lorsque le joueur perd avec son score. Lorsqu'on ferme cette fenetre, tout le jeu se ferme.

Le défilement automatique de la ligne est réalisé en utilisant un compteur qui avance à chaque pas, grâce à un thread, et qui joue le rôle de position 0 sur le parcours. Ce Thread utilisera une boucle infinie.

Les "Sauts" de l'ovale lorsque le joueur clique avec la souris sont réalisé par une fonctionnalité

MouseListener. Il faut également implémenter une chute constante de l'ovale afin de donner un effet que l'ovale doit être maintenu en l'air. De plus, il faut un test de collision afin de pouvoir déterminer lorsque l'ovale sort de la ligne et donc que l'utilisateur perd. Pour finir, à chaque saut de l'utilisateur l'oiseau va voler (voir Figure 2).

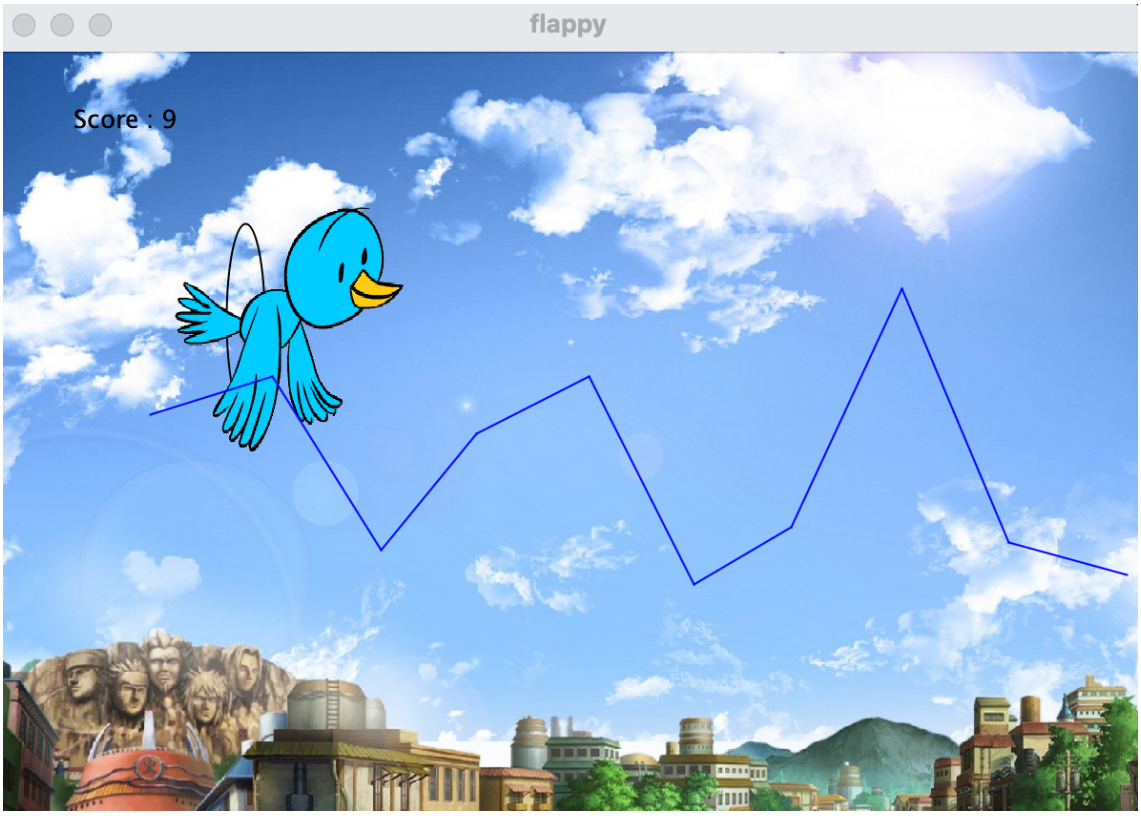

Figure 1: Interface Graphique

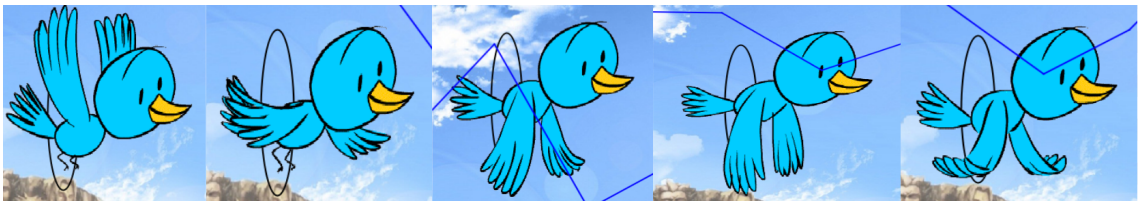

Figure 2: Quelques envolements de l'oiseau a chaque clic

3 Plan de développement

Pour mieux comprendre comment le projet a été réalisé, on présente la liste des tâches effectuées par un diagramme de Gantt (voir Figure 3). Chaque "morceaux" de chaque séance correspond à une durée de 15min.

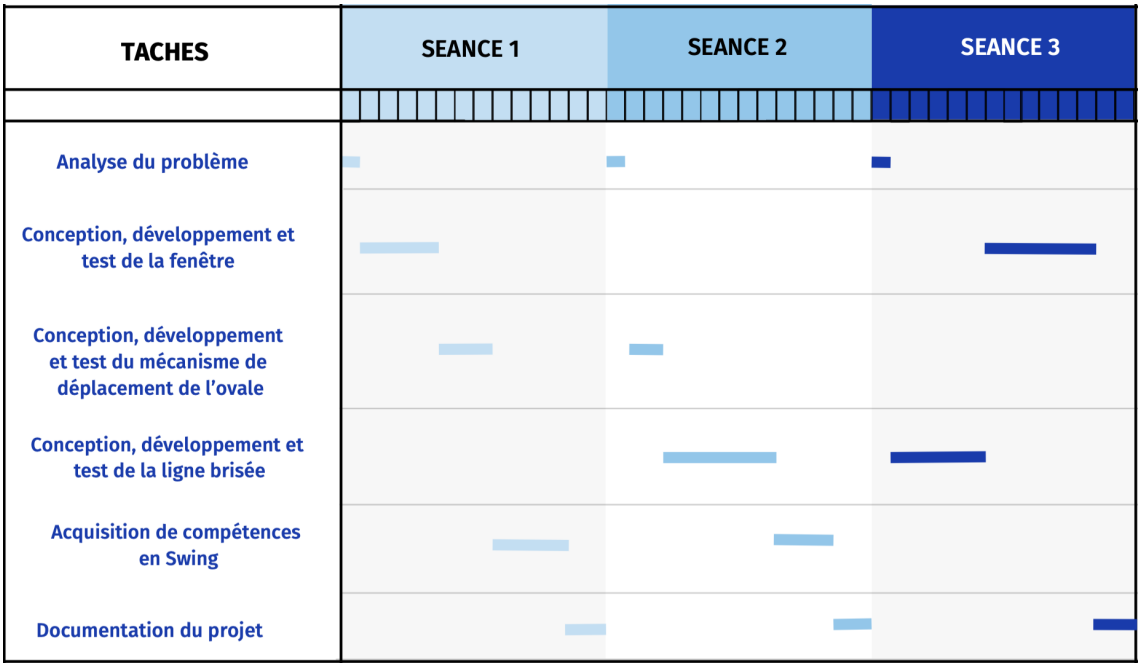

Figure 3: Diagramme de Gantt

4 Conception générale

Ce projet a été créé avec le motif MVC (voir Figure 4). Il y a la classe *état* pour le modèle (il s'agit de monter et descendre l'ovale), la classe *contrôle* pour le contrôleur (il s'agit du fait que l'ovale monte lorsqu'un clic de souris) et la classe *affichage* pour la vue (il s'agit de ce que l'utilisateur voit). Pour finir, il fallait stopper les Thread du Model afin que le jeu se stoppe et ne soit plus jouable lors de la fin du jeu.

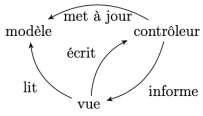

Figure 4: Motif MVC

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 :

- *LARG* : qui définit la largeur de la fenêtre
- *HAUT* : qui définit la hauteur de la fenêtre
- *wo* : qui définit la largeur de l'ovale
- *ho* : qui définit la hauteur de l'ovale
- *Xo* : qui définit la position de départ de l'ovale sur l'axe des abscisses
- *HAUTEUR* : 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 évènementielle 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. Le calcul de la coordonnée y se fait en fonction de la nouvelle coordonnée x afin d'éviter d'avoir des angles trop optus et que le jeu soit injouable. Un nombre aléatoire est pris entre 0 et hypotenus l'angle maximal possible, défini dans la classe Parcours, et la hauteur est ainsi calculée, car avec la valeur x, on peut calculer l'hypoténus. 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.

En effet, 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'un 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 :

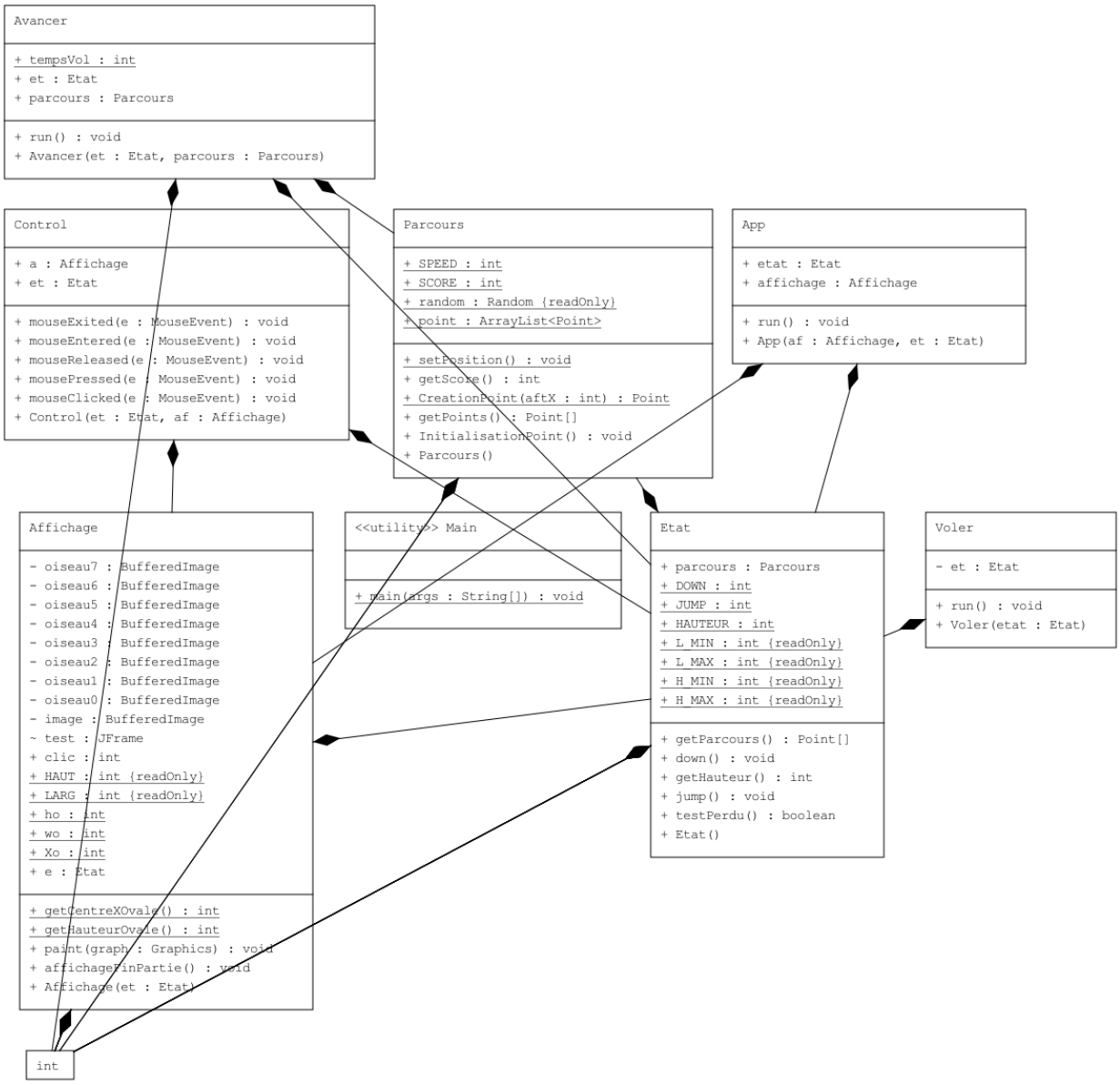

Figure 5: Diagramme de classe

6 Résultat

Voici l'évolution de l'interface graphique durant les trois séances :

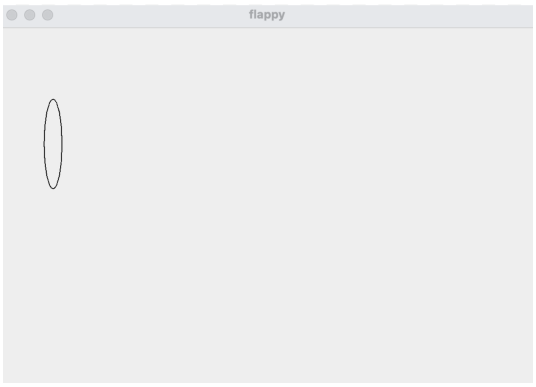

Figure 6: Seance 1

Figure 7: Seance 2

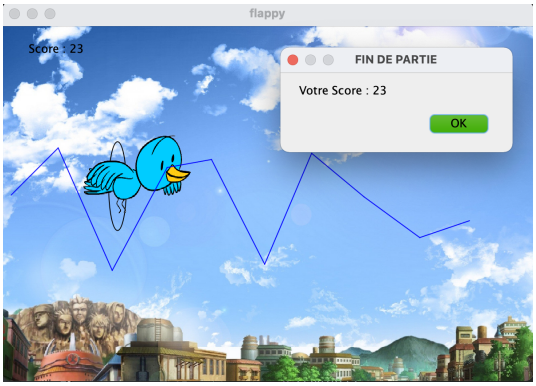

Figure 8: Seance 3

7 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.

8 Documentation développeur

Toutes les fonctionnalités majeures ont été implémenté, mais des améliorations pourraient être apportées au projet comme une meilleure adhésion de l'oiseau (sans ovale). 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, des bruits pour les clics ou quand l'oiseau sort du chemin , un menu d'accueil avec des paramètres comme la taille de la fenêtre ou la couleur de l'oiseau.

9 Conclusion et perspectives

Nous en sommes à la fin de ce projet, en conclusion on peut dire qu'il a été un très bon challenge personnel de devoir créer un jeu avec l'utilisation de Thread dans un language complètement nouveau. Chaque étape avait ses difficultés mais en cherchant cela permet de mieux comprendre.