**Compte Rendu de la seconde séance PCII**

1. **Introduction**

Nous voulons réaliser un mini-jeu inspiré du célèbre jeu ***flappy bird*** en une version simplifiée. Dans lequel nous avons un ovale en guise d’oiseau qui se déplace le long des obstacles représentés par une ligne brisée à la place de tunnel. Le but du jeu est d’éviter n’est pas d’éviter les tunnels mais plutôt que l’ovale ne sorte de la ligne brisée. Pour cela, le joueur peut cliquer sur l’écran pour faire monter l’ovale (l’équivalent du vol pour l’oiseau), qui redescend ensuite tout seul. Voici à quoi pourrait ressembler l’interface graphique de notre jeu :

****

Figure 1 : Rendu attendu du jeu

Le but du jeu est assez simple : l’ovale doit suivre la ligne rouge pour ne pas perdre. Une fois que la ligne rouge n’est plus dans l’ovale, le joueur aura perdu

1. **Analyse globale**

A la première séance, nous sommes encore aux prémices du jeu car nous implémentons que 2 sous fonctionnalité pour l’instant : la création de la fenêtre et de ses dimensions et de la création de l’ovale et de son affichage, puis de l’action qui permet de faire monter l’ovale dans la fenêtre à chaque clic de souris de l’utilisateur. A première vu ces fonctionnalités ne sont pas importante mais ce sont les 2 bases sur laquelle se base le jeu entier. Parce que sans fenêtre nous n’avons pas d’interface d’affichage et sans action pour faire monter l’ovale, le jeu n’aurait aucun intérêt.

A la première séance nous avons implémenté deux fonctionnalités comme une interface graphique avec la création de la fenêtre et de l’ovale et d’une interaction de l’utilisateur avec l’ovale via MouseListener. Cette séance nous avons ajouté un semblant de gravité en ajoutant une chute constante de l’ovale pour qu’elle soit maintenu dans les airs, la création de la ligne brisée avec le déplacement de la ligne et non de l’ovale, et de la suppression des points non utiles que l’ovale a dépassée. Dans cette partie l’utilisation de Thread a été très importante pour pouvoir gérer plusieurs choses en même temps.

Nous voilà maintenant à la dernière séance et le jeu est presque fonctionnel. Le mouvement des lignes brisées et l’implantation de méthode pour que le mouvement soit infini et l’ajout de la pseudo-gravité. Pour que le jeu soit complet il nous faut implémenter un test de défaite, c’est-à-dire un test de collision entre les extrémités haute et basse de l’ovale et la ligne brisée. Une fois que l’utilisateur a perdu ce qui veux dire que la collision a été vérifiée une fenêtre s’affiche avec le score final et un bouton pour pouvoir fermer la fenêtre de jeu.  
Les Threads qui servent de moteur au jeu s’arrêtent tous et affiche la fenêtre de score.

1. **Plan de Développement**

L’avancement du projet lors de la première séance a été très inégal, en effet la documentation des bibliothèques notamment Swing et la compréhension du sujet a été assez longue (Environ 1 heure)

Ensuite viens l’étape de la création de la fenêtre et de l’ovale qui une fois la documentation faite a été assez rapide (Environ 30min) puis il y’a eu l’interaction de l’utilisateur puis pouvoir faire bouger l’ovale vers le haut qui a nécessité beaucoup de test (Environ 45min) enfin la documentation du projet qui a été à mon sens le plus long (Environ 1 heure) et enfin la vérification globale (Environ 15min)

L’avancement de ce projet lors de cette deuxième séance s’est déroulé de façon méthodique, avec la liste des différentes choses qui ont été faite. La lecture du sujet et la compréhension de la problématique (Environ 15min), la conception et la mise en marche de la chute de l’ovale comme une pseudo-gravité (Environ 45min), la conception et la mise en marche de la ligne brisé (Environ 30min), puis la création de la ligne brisée continue sans fin (Environ 45min), puis pour finir la documentation qui a été légèrement moins longue cette fois ci (Environ 45min)

La finalisation de cette fin de projet a été assez facile à organiser ayant peu de chose à faire.  
Comme à chaque début de séance environ 15min a été consacré à la lecture du sujet et la compréhension de la problématique. La réflexion et création des collision a été la plus longue tache de cette séance (Environ 1h30), puis viens la création et l’affichage de la fenêtre de fin de partie (Environ 30min). La modification des ***Threads*** avec l’implémentation des collisions. Enfin La documentation du projet (Environ 1 heure). Ayant eu des soucis sur les tests de collisions et ne fonctionnant pas à 100% j’ai passé environ 1h30 sur le debugage des collisions.

1. **Conception Générale**

Pour le début de la réalisation de ce projet flappy bird simplifié, on a d’abord utilisé le motif MVC pour le développement de notre interface graphique. La première fonctionnalité est la création et l’affichage d’une fenêtre avec un ovale avec un emplacement variable, fait partie de *View* du motif MVC, en effet il s’agit de ce que l’utilisateur voit, et la seconde, le fait que l’ovale monte lorsqu’un clic de souris de l’utilisateur est fait dans la fenêtre rentre dans la partie *Controller*, il s’agit ici de la partie de contrôle du jeu, non visible par l’utilisateur.

Pendant la seconde séance, le motif MVC étant déjà présent on a tout simplement continué de respecter le motif MVC. On a développé la méthode ***moveDown()*** dans le Model. Pour la création des lignes brisées et leur mouvement infini celle-ci était dans la partie ***View***. Une fois ces choses faites nous pouvions passer à la troisième séance. Aucune modification a été apporté sur la partie ***Controller***

Pour la fin de notre projet, il y’a eu que 2 implémentations supplémentaires. La première les tests de collision entre l’ovale et la ligne brisée qui est implémenté dans la partie ***Model***, puis l’affichage de la fenêtre de fin de jeu qui a été implémenté dans la partie ***View***.

1. **Conception Détaillé**

Diagramme de Classe Séance 2 :

Afin de simuler la pseudo-gravité de chute de notre ovale sans la méthode *repaint(),* car pour rappel nous sommes dans la partie *Model* de la MVC, l’utilisation de *Thread* est obligatoire : dans un premier temps la création d’un *Thread* dans la classe *Etat* qui va modifier la hauteur de l’ovale et un second *Thread* dans la classe *Affichage* qui va redessiner la fenêtre ou bout d’un lapse de temps défini (Dans mon cas 300ms). Pour la création d’une ligne brisée infinie, nous avons dû modifier la méthode *getParcours()* afin de supprimer les points qui sortait du cadre et d’en régénérer un à chaque suppression pour avoir une ligne brisée infini.

Diagramme de Classe Séance 3 :

La méthode qui s’occupe de tester si il y’a collision entre l’ovale et la ligne brisée est ***testPerdu()*** définie dans la classe ***Etat*** qui calcule la position du point le plus haut et le plus bas de l’ovale et regarde si elle est au même point que la ligne brisée. Pour la fenêtre de fin de partie qui s’affiche l’utilisation de ***JOptionPane*** a été nécessaire, la fenêtre de fin partie est gérer par la méthode ***affichageFinPartie()*** dans la classe ***Affichage.***

**Une image contenant texte, équipement électronique

Description générée automatiquement**

Figure 2 : Diagramme de Classe Séance 2

**Une image contenant texte, équipement électronique

Description générée automatiquement**

Figure 3 : Diagramme de Classe Séance 3

1. **Résultat**

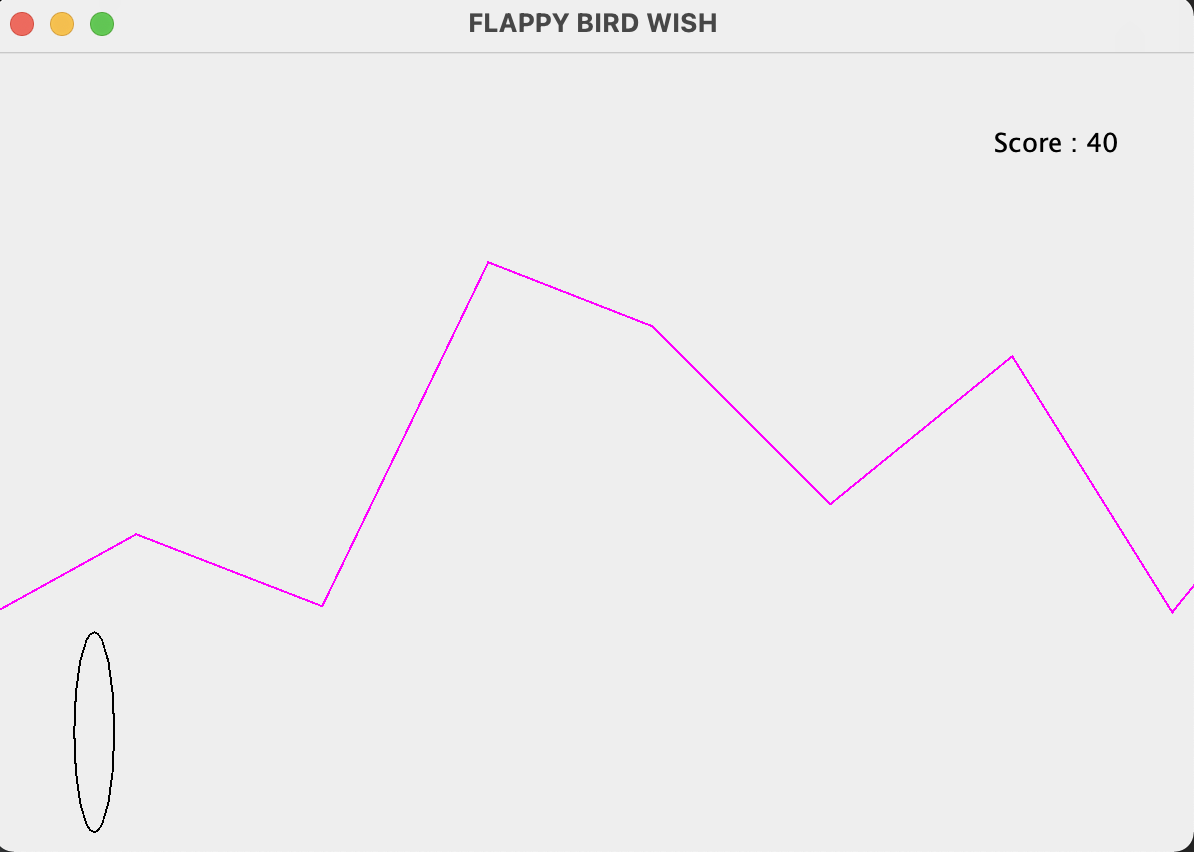
****

Figure 4 : Visuel du jeu à la fin de la séance 2

Figure 5 : Visuel du jeu au début de la séance 3



*Une image contenant texte

Description générée automatiquement*

Figure 6 : Visuel du jeu à la fin de la séance 3

1. ***Documentation Utilisateur***

Pour que l’utilisateur puissent jouer à « Flappy Bird Wish » plusieurs prérequis sont nécessaire : En premier lieu un IDE est nécessaire de préférence mais il est tout de même possible de lancer le jeu tout même uniquement avec Java si on exporte le fichier en .jar en exécutable. Une fois dans l’IDE, l’utilisateur dois importer le fichier, aller dans la class Main puis d’exécuter le fichier et ainsi le jeu se lancera et aura juste à cliquer avec sa souris ou le pavé tactile. Dans le cas où l’utilisateur n’utilise pas d’IDE, il aura juste à lancer le fichier exécutable et le jeu sa lancera et comme pour l’IDE, l’utilisateur aura juste à cliquer avec sa souris ou pavé tactile dans la fenêtre affichée.

1. ***Documentation Développeur***

Le tutoriel touchant à sa fin, plusieurs difficultés ont été plus difficile à gérer que d’autre. Les principales difficultés étaient l’ajout de la pseudo-gravité au fil du temps, le mouvement et affichage infini de la ligne brisée avec la création et suppression des points au bon moments ou encore la détection de l’ovale par rapport à la ligne brisée qui n’est pas encore totalement fonctionnel. Quelques « bug » sont encore présent dans la détection de l’ovale par rapport de la ligne brisée via l’utilisation de ***TestPerdu()*** . Le plus gros du tutoriel a été fait sur la class ***Etat*** et ***Affichage*** où plus de la moitié du tutoriel y est codé et est le pont-levis du bon déroulement du jeu.

1. ***Conclusion et Perspectives***

Nous voilà au terme de ce tutoriel de PCII dans laquelle l’utilisation de Thread et de la bibliothèque Swing nous a permis de créer et un jeu et de voir l’utilisation de multiple thread exécutant plusieurs taches en même temps. Les difficultés rencontrées m’ont permis de comprendre mes mauvaises habitudes sur le manque de commentaires sur mon code et m’ont permis d’apprendre l’utilisation de méthode sur les bibliothèques jamais utilisée.

Toutefois, il reste encore des soucis à régler sur les tests de défaite de l’utilisateur ou encore des fonctionnalités agréables pour l’utilisateur comme relancer une partie sans à avoir rerun les fichiers à chaque partie.