**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 : 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.

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 ensuite 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)

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.

1. **Conception Détaillé**

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.

**Une image contenant texte, équipement électronique

Description générée automatiquement**

Figure 2 : Diagramme de Classe Séance 2

1. **Résultat**

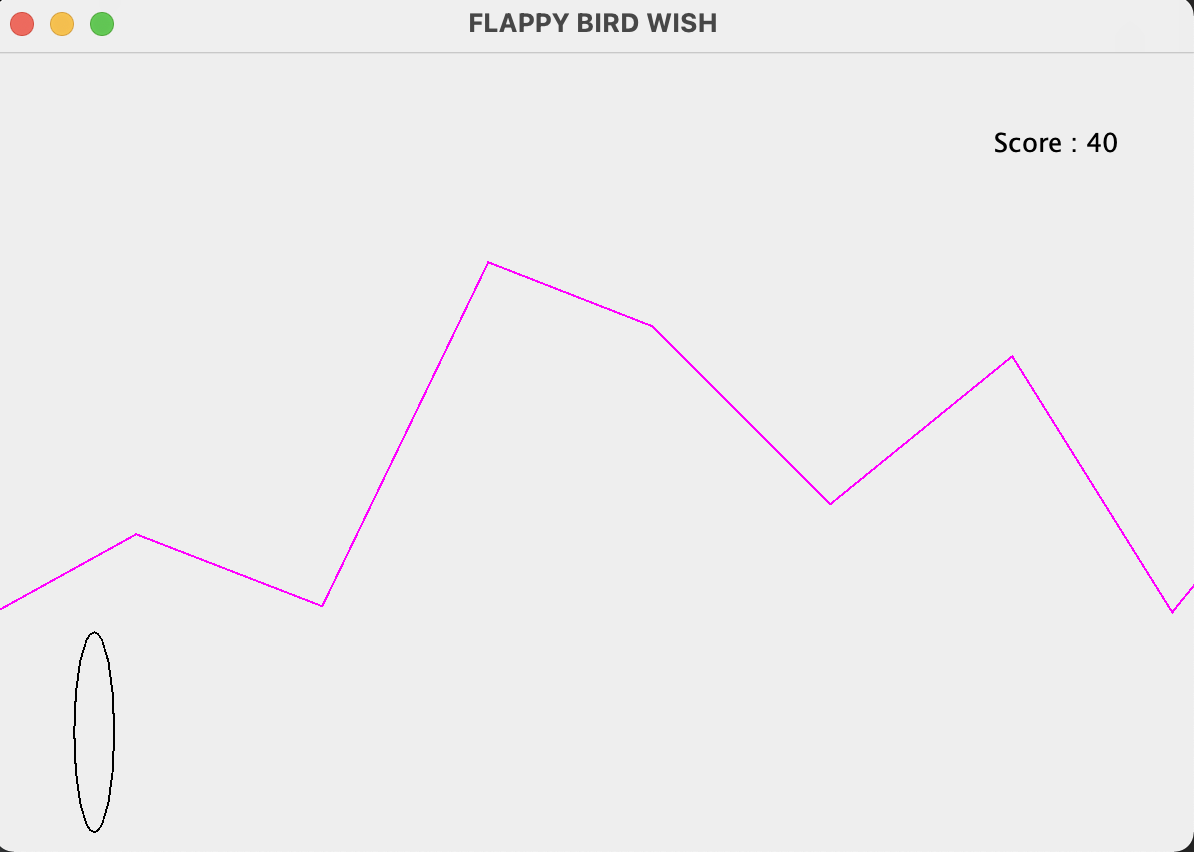
****

Figure 3 : Visuel du jeu à la fin de la séance 2