# Compte Rendu de la séance finale

## I. Introduction

Le but de ce projet est de réaliser une version plus simpliste du célèbre jeu *Flappy Bird*. En effet, il s'agira d'une version beaucoup plus simpliste que l'originale, nous allons matérialiser l'oiseau par un ovale et le chemin sera matérialisé par une ligne, une ligne que l'ovale ne devra pas quitter. Le but du joueur est de faire avancer l'ovale tout en faisant en sorte qu'il ne quitte pas ladite ligne. Pour cela, le joueur devra cliquer dans l'espace de la fenêtre pour faire sauter l'ovale, qui redescendra tout seul, et ainsi le faire progresser, il ne devra pas le laisser aller en dehors de la ligne ou alors la partie sera perdue et il devra recommencer. La difficulté du jeu est avant tout de réussir à appréhender le saut et la chute de l'ovale afin de ne pas quitter la ligne et espérer arriver à la fin de celle-ci.


*Figure 1 : Aspect final espéré du jeu*

Sur la figure 1, on peut apercevoir l'ovale ainsi que la ligne rouge qu'il devra suivre. Pour gagner, joueur devra se mouvoir pour atteindre la fin de la ligne.

## II. Analyse Globale

Dans cette première partie, nous allons parler de certaines fonctionnalités, à ce stade du projet nous n'avons pas toutes les fonctionnalités qui sont mises en place. A ce stade du projet, deux sous fonctionnalités sont présentes : la création d'une fenêtre dans laquelle un ovale est dessiné et le fait que l'ovale monte sur l'écran à chaque clic de souris de l'utilisateur. Ce ne sont pas des fonctionnalités compliquées à implémenter mais très primordiales, elles sont la base de ce projet. En effet, la création de la fenêtre tient un rôle très important car sans elle, il n'y aurait pas de jeu. La fonctionnalité de saut de l'ovale est tout aussi importante car il s'agit en quelque sorte de la base de notre jeu : un ovale qui saute.

Lors de la seconde partie, nous avons implémenté plusieurs fonctionnalités comme une interface graphique (création d'une fenêtre et d'un ovale), une interaction entre l'utilisateur et l'ovale grâce au *MouseListener*, lors que cette séance ci nous devons implémenter la chute constante de l'ovale afin de donner un effet que l'ovale doit être maintenu en l'air, la génération de la ligne brisée ainsi que son déplacement et enfin la création et la suppression de points de la ligne brisée. Pour faire cela, l'utilisation des *Thread* nous sera très utile afin de réussir à tout gérer en même temps.

Lors de la troisième et dernière séance, nous avons implémenter un test de collision afin de pouvoir déterminer lorsque l'ovale sort de la ligne et donc que l'utilisateur perd. Ensuite nous avons dû implémenter une fenêtre contextuelle qui apparaît lorsque le test de collision est vérifié et qui affiche le score de fin de partie du joueur et qui rend l'interface du jeu non cliquable. Evidemment, lorsque ce test de collision est vérifié, les *Threads* du jeu s'arrêtent sauf celui de l'affichage qui rend « l'écran de fin de jeu ».

## III. Plan de Développement

L'avancement de ce projet lors de cette première séance s'est déroulé de façon très méthodique, voici chacune des tâches qui ont été réalisées. Tout d'abord, une lecture et une analyse du problème (environ 15 minutes) ensuite une étape de conception, de développement et de test d'une fenêtre contenant un ovale (environ 30 minutes) par la suite, une étape de conception, développement et de test du déplacement de l'ovale (environ 30 minutes), il s'en est suivis une acquisition des compétences nécessaire en Swing (environ 45 minutes) et enfin, la réalisation de la documentation du projet (environ 60 minutes)

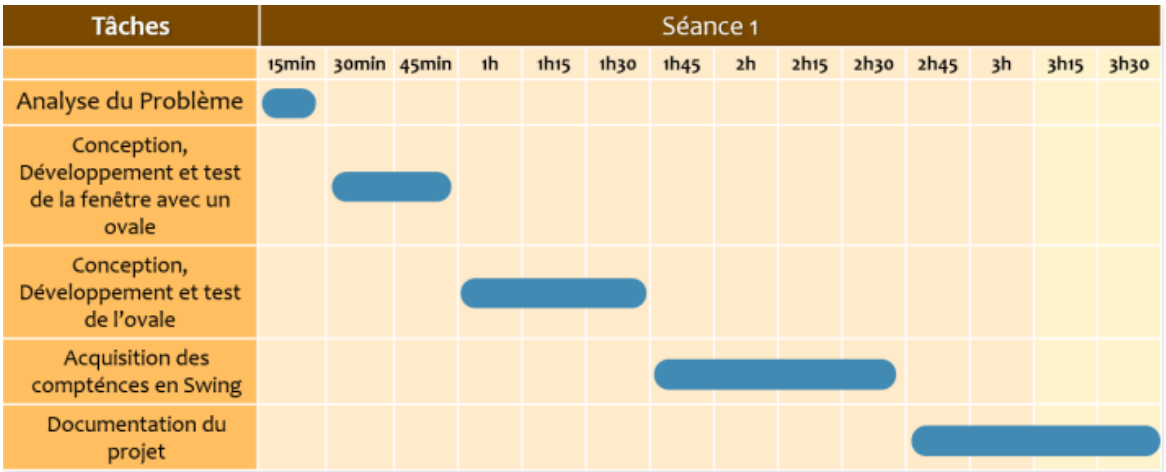


*Figure 2 : Diagramme de Gantt du déroulé de la séance*

L'avancement de ce projet lors de cette deuxième séance s'est également déroulé de façon très méthodique, voici chacune des tâches qui ont été réalisées. Tout d'abord, une lecture et une analyse du problème (environ 15 minutes) ensuite une étape de conception, de développement et de test d'une sorte de gravité pour que l'ovale descende en permanence (environ 30 minutes) par la suite, une étape de conception, développement et de test de la ligne brisée (environ 30 minutes), il s'en est suivis de la création d'une ligne brisée infinie (environ 45 minutes) et enfin, la réalisation de la documentation du projet (environ 60 minutes)

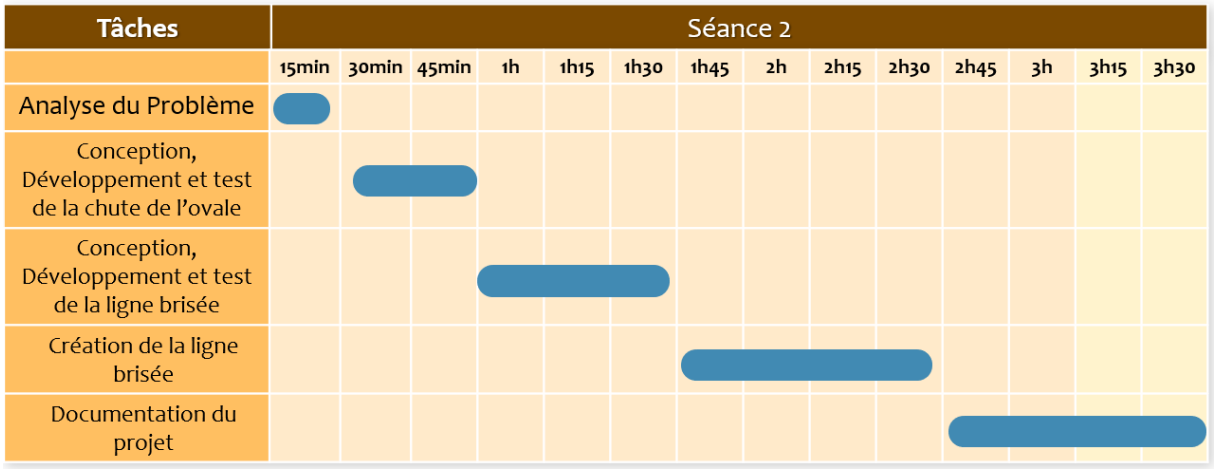


*Figure 3: Diagramme de Gantt du déroulé de la séance 2*

L'avancement de ce projet en cette dernière séance s'est également déroulé de façon très méthodique, voici chacune des tâches qui ont été réalisées. Tout d'abord, une lecture et une analyse du problème (environ 15 minutes) ensuite une étape de conception, de développement et de test du tes de collision (environ 60 minutes), par la suite, une étape de conception, développement et de test de l'affichage de fin de partie (environ 45 minutes), il s'en est suivis de faire en sorte que les *Thread* s'arrêtent lorsque l'affichage de fin de partie est la (environ 30 minutes) et enfin, la réalisation de la documentation du projet (environ 60 minutes).

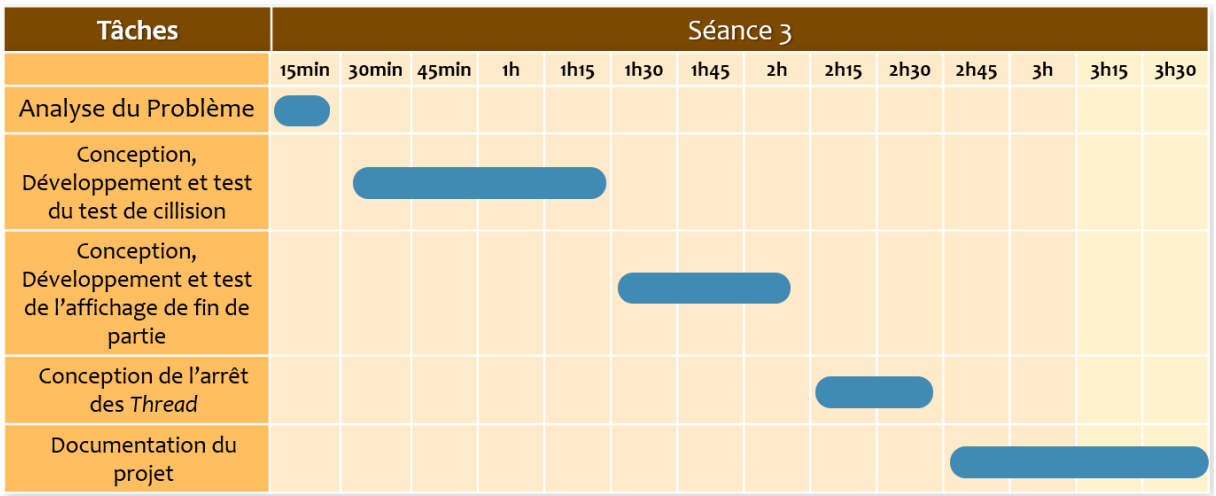


*Figure 4: Diagramme de Gantt du déroulé de la séance 3*

## IV. Conception Générale

Pour la réalisation de ce projet, on va d'abord utiliser le motif MVC pour le développement de notre interface graphique. La première de nos fonctionnalités, la création d'une fenêtre avec un ovale dessiné, rentre dans la partie *View* du motif MVC, en effet il s'agit de ce que l'utilisateur voit, quant à la seconde, le fait que l'ovale monte lorsqu'un clic de souris est fait dans la fenêtre rentre dans la partie *Controller* du motif MVC, il s'agit ici de la partie de contrôle du jeu, invisible à l'utilisateur.

Pour la réalisation de notre projet en cette deuxième séance, il ne restait plus qu'à continuer à le respecter le motif MVC. Dans le *Model*, il suffisait d'implémenter la méthode *moveDown* et la création du parcours que l'ovale devrait parcourir, dans la *View*, il y a eu quelques modifications à apporter afin d'afficher la ligne brisée et enfin il n'y avait rien à changer dans la partir *Controler* du projet.

Pour la réalisation de notre projet lors de cette dernière séance, il nous restait qu'à faire une fonction de test dans le *Model* et de faire un affichage de fin de partie dans la *View*. Et également de stopper les *Thread* du *Model* afin que le jeu se stoppe et ne soit plus jouable lors de la fin du jeu.

## V. Conception Détaillée

Afin de faire une fenêtre avec un ovale, l'API Swing et la classe *JPanel*, nous avons défini plusieurs constantes telles que les dimensions de l'ovale ou encore les dimensions de la fenêtre. Ensuite, pour le déplacement de l'ovale, on a utilisé la programmation évènementielle avec la classe *MouseListener*. Nous avons également défini des constantes pour la hauteur de l'ovale ainsi que la taille du saut.


*Figure 5 : Diagramme de classe séance 1*

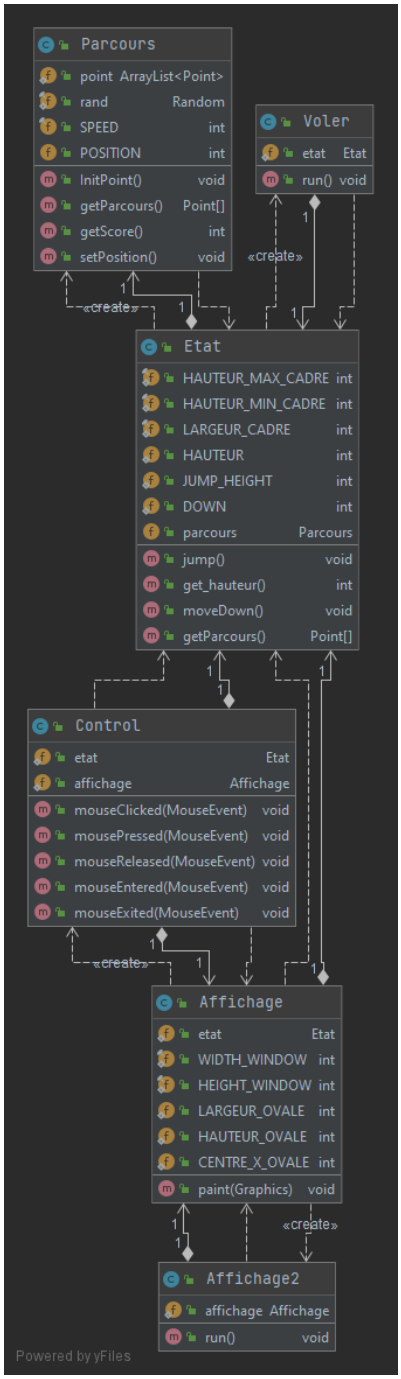


*Figure 3 : Diagramme de classe séance 2*

Afin de simuler la chute de notre ovale sans utiliser la méthode *repaint()* vu que nous sommes dans la partie *Model* de la MVC, on va créer deux *Thread*, un premier dans la classe *Etat* qui va modifier la hauteur de l'ovale et un deuxième dans la classe *Affichage* qui va redessiner la fenêtre ou bout d'un lapse de temps défini au préalable. Pour ce qui est de faire une ligne brisée infinie et la création de point il nous a suffit de modifier la méthode *getParcours()* afin de supprimer les points qui sortait du cadre et d'en regénérer un à chaque suppression.

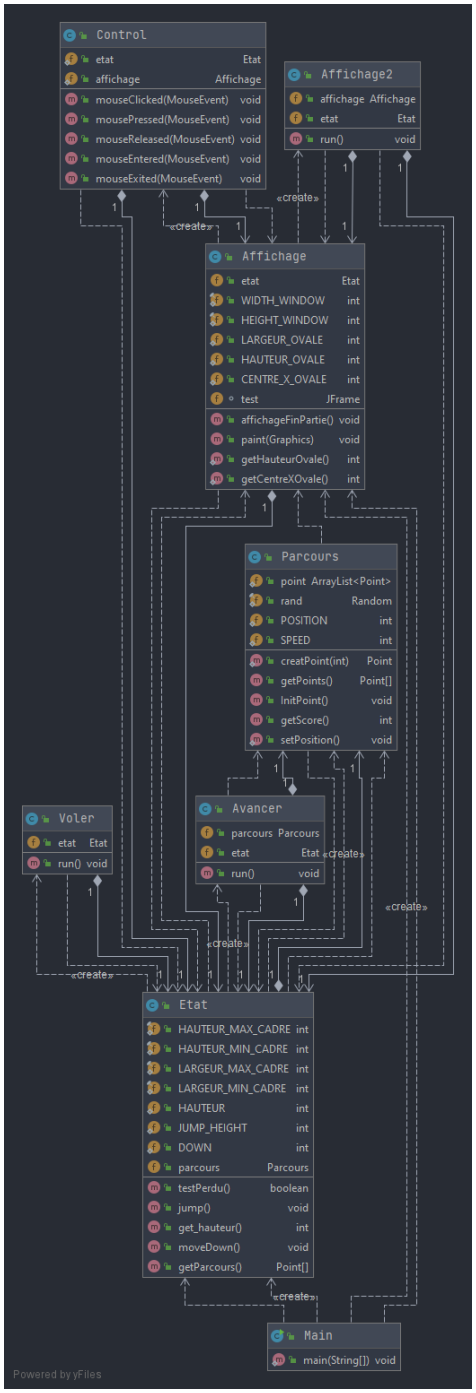


*Figure 6 : Diagramme de classe séance 3*

Afin de faire notre test de collision nous avons implémenté une méthode *testPerdu()* qui calcule le point sur la ligne où se situe l'ovale et qui compare les ordonnées. Pour faire un affichage de fin de partie nous avons utilisé les affichages de *JOptionPane* afin de générer un affichage propre. Nous avons créé une méthode *affichageFinPartie()* dans la classe *Affichage* et nous l'appelons à la fin du *Thread* dans *Affichage2*, lorsque la boucle *while()* se fini on appelle la fonction d'affichage de fin de partie afin de montrer le score et ensuite d'*exit* le jeu.

## VI. Résultats

Ci-dessous les premiers résultats obtenus lors de cette première séance de projet, on peut y voir que la fenêtre y est implémentée ainsi que l'ovale y est bien dessiné. Une autre chose est présente mais non visible sur la figure 4 est le saut vertical de l'ovale qui a également été implémenté.

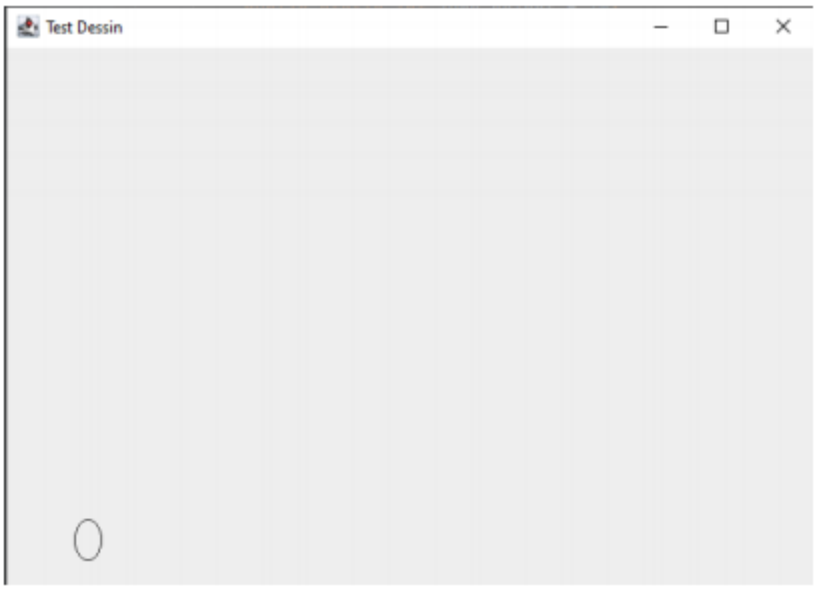


*Figure 7 : Visuel du jeu à la fin de la séance 1*

Ci-dessous les résultats obtenus lors de cette deuxième séance, on peut y voir la ligne brisée ainsi que l'ovale qui descend si l'utilisateur ne le maintien pas en l'air. Nous pouvons remarquer que ce visuel se rapproche de celui présenté en *figure 1*.

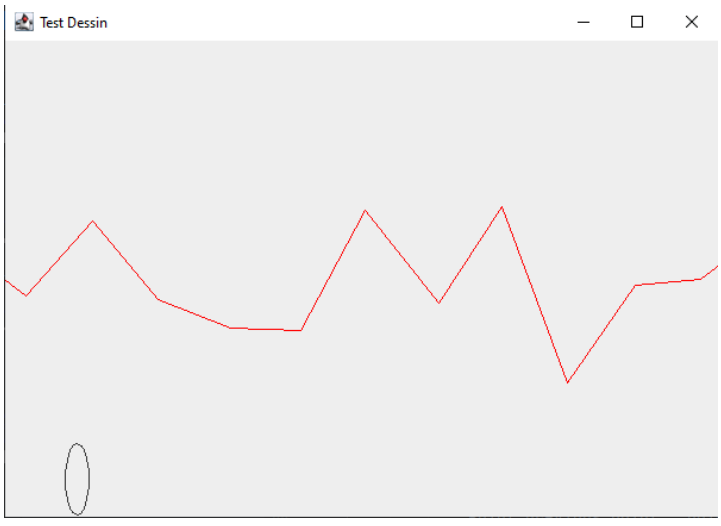


*Figure 8 : Visuel du jeu à la fin de la séance 2*

Ci-dessous le résultat de l'affichage à la fin de la séance 3 avec également l'affichage de fin de partie. On y voit le score ainsi que la ligne brisée infinie et enfin l'écran de fin de jeu.

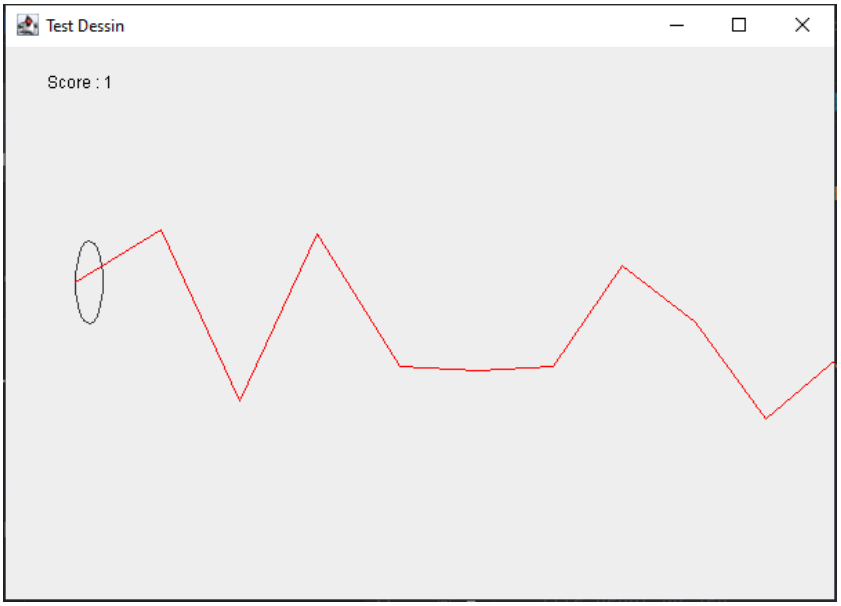


*Figure 8 : Visuel à la fin de la séance 3*

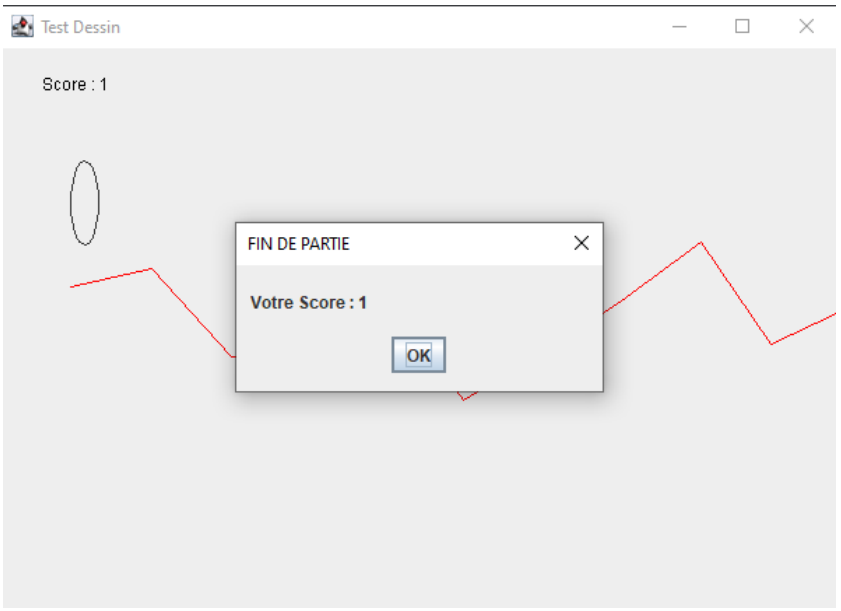


*Figure 9 : Visuel de l'écran de fin de partie à la fin de la séance 3*

## VII. Documentation Utilisateur

Pour pouvoir jouer au jeu il y a quelques petites choses à savoir. Tout d'abord, un IDE Java est nécessaire (ou Java seul si jamais vous avez fait un export *.jar* en exécutable). Si vous êtes dans le premier cas et que vous avez un IDE Java, il vous faut importer le projet dans votre IDE, sélectionner la classe *Main* et enfin lancer l'application '*Run as Java Application*'. Vous pouvez cliquer sur la fenêtre pour faire monter l'ovale. Si vous être dans le second cas, que vous possédez un exécutable *.jar*, il vous suffira de double cliquer sur le fichier exécutable et de cliquer dans la fenêtre pour faire monter l'ovale.

## VIII. Documentation Développeur

Nous en sommes à la fin de ce projet, nous avons rencontré des difficultés sur de nombreux points comme la génération et la suppression de la ligne brisée ou encore de faire un test de collision étant 100% opérationnel, ce qui n'est pas encore le cas, il reste des défauts dans la fonction. Les classes *Etat* et *Affichage* sont des classes extrêmement intéressantes de par toutes les fonctionnalités qu'elles ont à gérer et comment elles les gèrent.

## IX. 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*. Certaines difficultés nous ont ralenti mais nous ont également permis d'apprendre. Toutefois, il reste de nombreuses choses à implémenter comme par exemple une interface graphique plus élaborée ou encore un system de *Retry* à l'écran de fin de jeu.