Rapport de projet

Nathan Monsoro

Introduction :

Nous avons donc pour but de créer un jeu de course de véhicule. Le principe de notre projet sera le suivant : un ovale que l'on remplacera plus tard par un véhicule qui monte, descend et se déplace à gauche et à droite, qui se déplacera près d'une ligne infinie qui défilera verticalement. Il devra aller d'un point de contrôle à un autre en un certain temps. Pour diriger l'ovale, l'utilisateur n'aura qu'à utiliser les flèches directionnelles du clavier. L'ovale aura une vitesse et une accélération qui changera en fonction de sa position par rapport à la ligne. Pour réaliser cela, on utilisera les classes JFrame et JPanel d'abord créer une fenêtre vide, puis avec un ovale statique. On devra ensuite utiliser la classe KeyListener pour que notre programme réagisse aux flèches directionnelles. Grâce à cela, nous pourront faire déplacer notre ovale avec les flèches directionnelles. On mettra aussi un décor et un fond. Il nous faudra ensuite automatiser la décélération de l'ovale ainsi que la création et le défilement de la piste. Pour cela, on utilisera les bibliothèques Thread et Point. Grâce à ces bibliothèques, nous pourrons calculer le score de l'utilisateur, et lui donner l'impression que la piste est infinie. Finalement on détectera quand l'ovale à l'arrêt ou qu'il n'a plus de temps afin d'afficher l'écran de défaite.

Analyse globale :

Nous pouvons identifier trois groupes principaux de fonctionnalités dans ce projet de mini-jeu : d'abord l'interface graphique qui affichera l'ovale ainsi que la piste que ce dernier devra essayer de suivre dans la fenêtre et le décor, puis le défilement automatique de la piste et apparition de potentiels obstacles ainsi que le défilement du temps entre deux points de contrôle, et enfin les déplacements de l'ovale. Les déplacements de l'ovale sont soit des réactions aux flèches directionnelles de l'utilisateurs pour le déplacer, soit une accélération/décélération automatique en fonction de sa position par rapport à la piste. Dans un premier temps on s'est occupé de la création de la fenêtre avec le dessin de l'ovale, puis les déplacements de l'ovale vers la gauche et la droite lorsque l'utilisateur utilise les flèches directionnelles. Ces deux fonctionnalités sont sûrement les plus simples du mini-jeu. Évidemment, l'étape de création de la fenêtre bien qui pas spécialement compliquée est capitale dans notre projet, puisque sans cela, aucun espace pour afficher notre jeu, ni pour interagir avec l'utilisateur. Il nous faut ensuite dans cette fenêtre dessiner l'ovale. Cette étape est aussi très simple, mais importante aussi puisque l'ovale est l'élément que le joueur va le plus regarder, puisque c'est sa position par rapport à la piste et aux adversaires seront des éléments cruciaux dans la prise de décision de l'utilisateur. La fonctionnalité qui déplace l'ovale est aussi très importante, puisque cela permettra à l'utilisateur de stratégiquement gagner le plus de temps possible dans la course. Au début de la séance, l'ovale se déplace de façon fluide dans la fenêtre, et la piste se déplace dans la fenêtre quand on utilise les touches Q et D du clavier (en AZERTY). La piste défile à l'infinie. On

a créé le décor en fond et affiché la vitesse, du temps restant et du score.

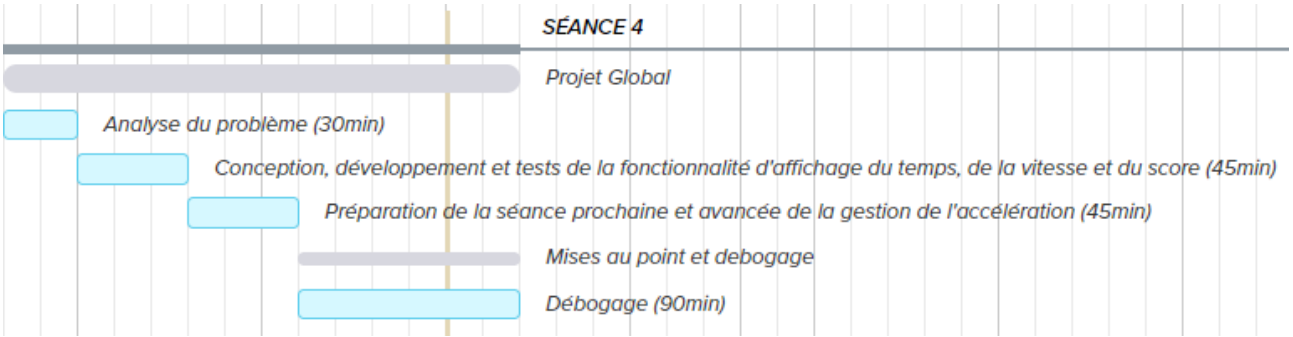
Lors de cette séance, on a terminé l'implémentation de la vitesse, et implémenté les points de contrôle.

Plan de développement :

Mon temps de travail s'est à peu près réparti comme suit :

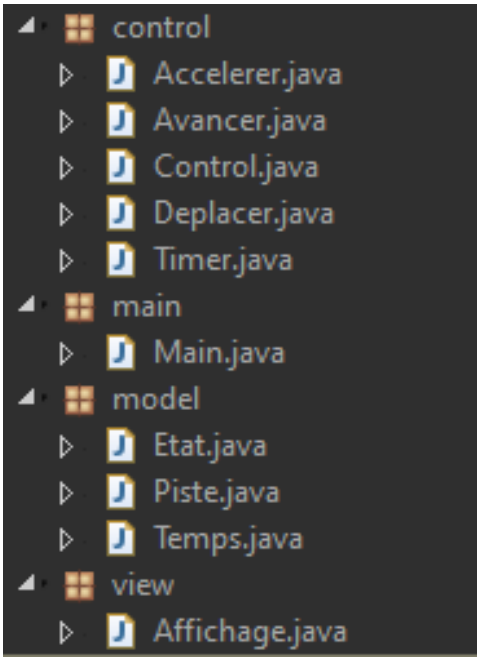
- Environ 30 minutes pour analyser le problème, les compétences que j'avais à acquérir et à réutiliser et ce que donnerait le projet à la fin de la séance, cette étape a été particulièrement longue car il m'a fallu retrouver et reprendre connaissance de ce que j'avais préparé lors de la séance précédente, plusieurs semaines en avance.
- J'ai ensuite réfléchi à la conception et au développement de la première fonctionnalité que nous avons implanté, qui est la gestion de la vitesse et de l'accélération. Cela m'a pris en tout 45 minutes.
- J'ai aussi passé 45 minutes à m'avancer sur les classes qui gèreront l'accélération, et le mécanisme de détection de quand on est sur la piste. Ce dernier n'est pas encore fonctionnel.
- Je n'ai pas fait de documentation cette séance, je m'en occuperai la prochaine fois.
- J'ai dû passer quelques temps au débogage, environ 90 minutes en tout, pour faire fonctionner toutes les fonctionnalités, et corriger mes algorithmes.

Ci-dessous le diagramme de Gantt qui représente la répartition de mon temps sur les 3h30 de la séance



Conception générale :

Pour ce projet, nous avons utilisé le modèle MVC. Nous avons donc organisé notre code dans 4 packages différents comme ci-contre.



- Le package model contient les classes suivantes :
    - La classe Etat qui représente l'état du jeu à afficher, elle contrôle la position de l'ovale et le décor.
    - La classe Piste qui gère la piste que devra suivre l'utilisateur.
    - La classe Temps qui gèrera le temps que l'utilisateur aura pour atteindre le prochain checkpoint.

- Le package control contient les classes suivantes :

    - La classe Déplacer qui permet un déplacement fluide de l'utilisateur.
    - La classe Control qui gère les entrées de l'utilisateur, elle permet à la classe Deplacer de fonctionner quand l'utilisateur utilise touches de direction.
    - La classe Avancer qui gère le défilement de la piste.
    - Les classes Accélérer qui gère l'accélération.
    - Timer qui gère le défilement du temps.

- La classe Affichage s'occupe d'afficher l'état du jeu. Elle crée la fenêtre, dessine l'ovale et affiche la piste, le décor et les informations du joueur. Cette classe est seule dans son package.

- La classe main est aussi seule dans son package. C'est elle que l'on exécute pour jouer au jeu.

Conception détaillée :

Nous allons dans cette partie parler de façon plus détaillée de l'implémentation des fonctionnalités.

D'abord l'affichage de la fenêtre et de l'ovale. Pour cette fonctionnalité on utilise l'API Swing. On a aussi besoin de la classe Affichage qui hérite de JPanel, et de la classe Etat. Les dimensions de la fenêtre que l'on veut créer sont stockées dans les constantes xS et yS de la classe Affichage. Les dimensions de l'ovale sont, elles, dans les constantes xC, yC, w et h de la classe Etat. Pour afficher la fenêtre on crée directement dans la classe Main une JFrame auquel on ajoute l'affichage. Grâce à l'attribut etat de la classe, on a relié l'état du jeu à notre affichage.

Nous avons aussi implémenté le déplacement de l'ovale sur l'axe horizontale dans la fenêtre. La longueur d'un déplacement est définie dans la classe Etat par la constante decalX. La fonction moveR() et moveL() qui fait monter notre ovale est appelé dans la classe Control qui implémente la classe KeyListener, lorsque l'utilisateur appuie sur Q ou D (on utilise pour le moment Q et D, mais on changera plus tard pour les flèches directionnelles). Après cela, le contrôleur demande à la classe Affichage de mettre à jour la fenêtre avec la fonction repaint(). Dans les fonctions moveR() et moveL(), on vérifie si l'ovale ne sortirait pas du cadre après le saut, si c'est le cas, on le colle simplement à la bordure de la fenêtre, sinon on le fait monter de la valeur de decalX.

Nous avons ensuite implémenté le défilement de la piste. Pour cela on a simplement réutilisé à classe Parcours utilisé plus tôt dans l'année. La seule différence est que la piste sera à la verticale, on doit donc utiliser Random sur la coordonnée x des points et non y comme avant. On doit aussi prendre en compte l'horizon. Avec la classe Avancer, on a ensuite géré le défilement avec la bibliothèque Thread, comme dans le projet précédent. On a aussi coloré le ciel, le sol et la piste.

On a ensuite implémenté des changements sur les déplacements de l'ovale. Désormais, les fonctions de déplacements de l'ovale modifient un attribut de la classe Etat qui correspond à la coordonnée x par rapport au centre de la piste. Ainsi, en ajoutant cette constante aux coordonnées des points de la piste, on a l'impression que l'on se déplace. On a fluidifié ces mouvements grâce à une extension de la classe Thread en la classe Déplacer, qui lorsque l'un des deux attributs depD ou depG est à true (pas les deux !) on se déplace du coté concerné. Ces attributs sont modifiés par la classe Control. Ainsi l'ovale ne s'arrête pas quand on maintient enfoncé une touche de déplacement.

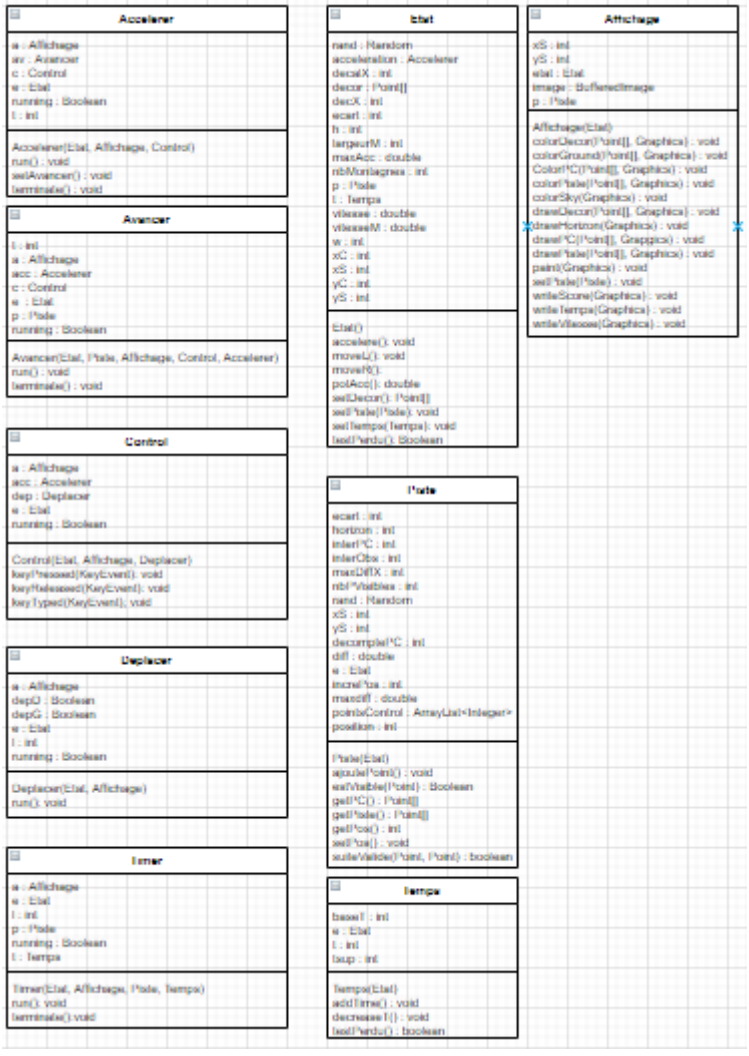
Nous avons implémenté le défilement et l'affichage du temps. Pour cela nous avons utilisé les classes Timer et Temps créées en avance lors de la séance précédente. Temps gèrera les fonctions et Timer appellera a intervalle réguliers les fonctions nécessaires. En l'occurrence, toutes les 1000 ms, la méthode decreaseT(), qui fait décroitre le temps de 1 seconde. Pour l'affichage, nous avons simplement créé une nouvelle méthode dans le classe Affichage qui écrit le temps restant. Pour le débogage, lorsque le temps arrive a 0, on appelle addTime (que l'on appelle normalement quand on passe dans un checkpoint) puisque les checkpoints ne sont pas encore implémentés.

Nous avons aussi implémenté l'affichage de la vitesse et du score de l'utilisateur. Pour cela nous avons utilisé la même technique que précédemment, c'est-à-dire que nous avons créées des méthodes dans la classe Affichage.

Nous avons commencé a gérer le remplacement de l'ovale par un véhicule, mais pour l'instant ce n'est pas concluant. (Décommenter la ligne pour afficher l'image actuelle) Nous continuerons de chercher une meilleure image, quitte à faire des retouches.

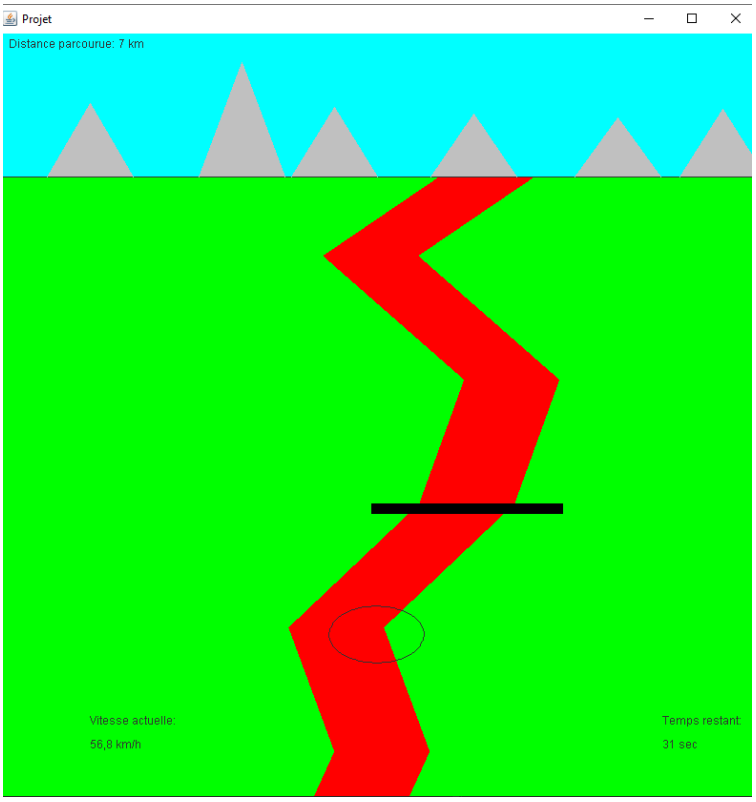
Lors de cette séance nous avons aussi commencé à gérer l'accélération du véhicule, mais cela sera détaillé dans le prochain rapport !

Ci-dessous le diagramme des classes implémentées de mon projet :



Résultat :

Voici ce que donne notre projet dans l'état actuel. L'ovale est au centre de la fenêtre et la piste se déplace, ainsi que les montagnes en fond. La piste se déplace à l'infini. L'ovale ne s'arrête pas quand on maintient les touches de déplacements appuyées. On voit la vitesse, le temps et la distance parcourue (score). On a matérialiser les points de contrôle par des bandes noires.



Documentation utilisateur :

Pour exécuter le code, il vous faudra un IDE avec Java pour l'exécuter à partir du code, ou bien Java pour l'exécuter à partir d'un fichier .jar.

À partir d'un IDE, importez simplement le projet, et à partir de la classe Main faites « Run as Java Application ». La fenêtre avec l'ovale s'ouvrira alors, vous pourrez déplacer ce dernier en utilisant les touches « <- » et « -> ».

À partir du fichier .jar, double cliquez dessus pour l'exécuter et ouvrir la fenêtre. Vous pourrez faire monter l'ovale de la même façon en cliquant dans la fenêtre.

Documentation développeur :

Pour continuer ce projet, il nous faudrait ajouter les textures pour le décor et une image animée pour l'ovale et le transformer en autre chose (un véhicule par exemple). On pourrait aussi arrondir les angles afin de les rendre moins abruptes et plus facile à manœuvrer, ça serait aussi sûrement plus esthétique. Il faudra faire en sorte que tous les points soient atteignables à partir du précédent. Il manque les conditions de défaites. Il faut vérifier la largeur de la route qui change.

Conclusion et perspectives :

Jusqu'ici, nous avons codé la fenêtre, l'ovale et le décor qui se déplace de manière fluide. Nous avons utilisé le format MVC, les bibliothèques Swing, Thread et KeyListener. Il manque encore beaucoup de fonctions fondamentales, mais le projet s'approche de son apparence finale.