université
PARIS-SACLAY

Rapport de projet PCII

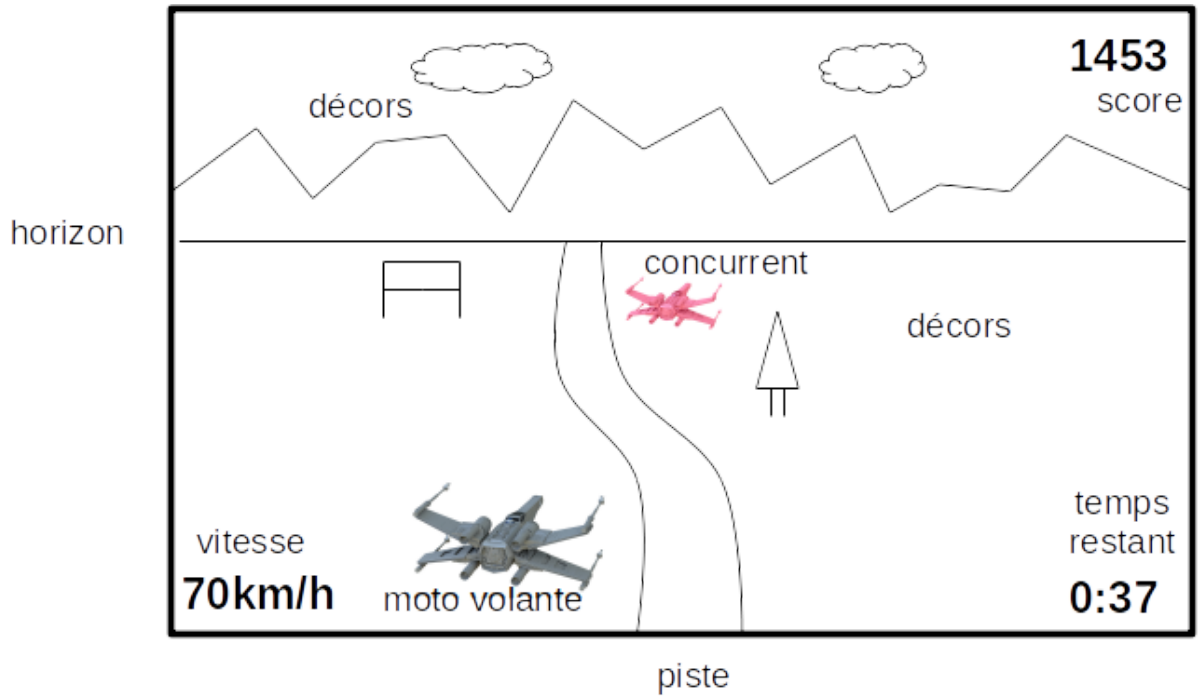
Réalisé par
Hongyu YAN, Shiqing HUANG

Sous la direction de
Thi Thuong Huyen Nguyen

Année universitaire 2020-2021
Licence Informatique

Introduction

L'objectif du projet est de réaliser un jeu vidéo des années 80 de type «course de voiture» en vue à la première personne (le véhicule est vu de derrière). L'originalité de ce jeu est de permettre au joueur de piloter une sorte de moto sur coussin d'air pouvant se déplacer aussi horizontalement pour dépasser ses concurrents. La figure ci-dessous donne une vision schématique du jeu :



Analyse globale

Il y a six fonctionnalités principales:

- l’interface graphique avec le véhicule, l' horizon et la piste,
- le défilement automatique de la piste,
- l’apparition des points de contrôles à intervalles réguliers
- la réaction de le véhicule aux clavier de l’utilisateur,
- le mécanisme de calcul de l’accélération du véhicule en fonction de la position par rapport à la piste ,
- Des données de jeu : temps restant et kilométrage

Nous nous intéressons d’abord uniquement à un sous-ensemble de fonctionnalités qui sont prioritaires et simples à réaliser:

- Création d’une fenêtre dans laquelle est dessiné le véhicule, l' horizon et la piste ;
- Déplacement du véhicule à droite et à gauche lorsqu’on tape les flèches du clavier

Nous nous consacrons ensuite à certaines fonctions :

- Animation de la piste à vitesse constante
- Création du décors
- Mouvements du décors de fond selon les touches du clavier
- Décompte des kilomètres et affichage

Ces fonctions sont plus difficiles et il faut maîtriser les connaissances multi-thread.

Plan de développement

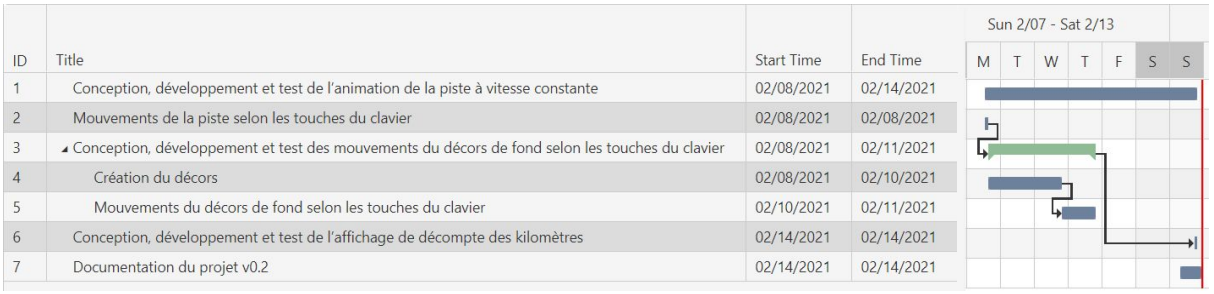
Liste des tâches:

- Séance 4 : Mise en place du projet
    - Analyse globale
    - Rédaction du plan de développement
    - Conception, développement et test d'un fenêtre avec une simple ligne brisée fixe(la piste) et limitée par un horizon, le véhicule
    - Conception, développement et test du mécanisme de déplacement de le véhicule
    - Documentation du projet v0.1

ID	Title	Start Time	End Time
1	Analyse globale	02/01/2021	02/01/2021
2	Rédaction du plan de développement	02/01/2021	02/01/2021
3	Conception, développement et test d'un fenêtre avec la piste et limitée par un horizon, le véhicule	02/06/2021	02/06/2021
4	Conception, développement et test du mécanisme de déplacement de le véhicule	02/06/2021	02/06/2021
5	Documentation du projet	02/07/2021	02/07/2021
6	Introduction + Analyse globale + Plan de développement	02/07/2021	02/07/2021
7	Conception générale + Conception détaillée + Résultat + Documentation utilisateur + Documentation développeur	02/07/2021	02/07/2021
8	Conclusion et perspectives	02/07/2021	02/07/2021

- Séance 5 et 6 : moteur de jeu et première version
    - Conception, développement et test de l'animation de la piste à vitesse constante
    - Conception, développement et test de l'affichage de décompte des kilomètres
    - Conception, développement et test des mouvements du décors de fond selon les touches du clavier
    - Conception, développement et test du calcul de l'accélération et la vitesse
    - Documentation du projet v0.2

    Séance 5



ID	Title	Start Time	End Time
1	Conception, développement et test de l'animation de la piste à vitesse constante	02/08/2021	02/14/2021
2	Mouvements de la piste selon les touches du clavier	02/08/2021	02/08/2021
3	Conception, développement et test des mouvements du décors de fond selon les touches du clavier	02/08/2021	02/11/2021
4	Création du décors	02/08/2021	02/10/2021
5	Mouvements du décors de fond selon les touches du clavier	02/10/2021	02/11/2021
6	Conception, développement et test de l'affichage de décompte des kilomètres	02/14/2021	02/14/2021
7	Documentation du projet v0.2	02/14/2021	02/14/2021

- Séances 7 et 8 : mécanique du jeu
    - Conception, développement et test de la détection de collisions
    - Conception, développement et test du mécanisme de points de contrôles
    - Conception, développement et test du mécanisme d'un décompte de temps
    - Documentation du projet v0.9

- Séance 9 et 10 : Finalisation du projet
    - Conception, développement et test des décors
    - Conception, développement et test de l'animation de la piste
    - Conception, développement et test d'un écran d'accueil
    - Conception, développement et test du mécanisme d'adversaires
    - Conception, développement et test de la création d'une sensation de profondeur
    - Documentation du projet v1
    - Préparation de la soutenance

Conception générale

L’interface graphique s’est construite autour du modèle MVC. Le package *vue* s’occupe de dessiner l’interface, c’est a dire, une fenêtre avec des voitures, une piste et des décors dedans.

Le package *model* définit l’ensemble des données qui caractérisent l’état de l’interface. La modification de ces données correspond à un changement de l’affichage dans l’interface graphique.

Le package *control* effectue les changements dans l'état et informe la *vue* des changements. Il modifie les états des dessins afin de faire déplacer la voiture, avancer la piste et les décors, puis ils informent la vue des changements pour redessiner.

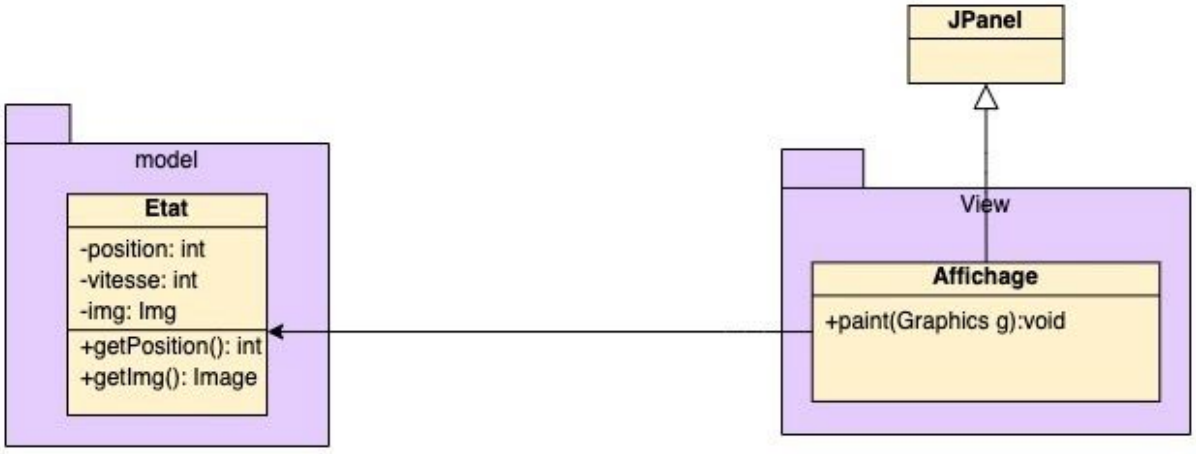
Conception détaillée

Pour la fenêtre avec une voiture, j'utilise l'API Swing et la classe `JPanel`. Je définis les dimensions de la voiture et de la fenêtre dans des constantes.

Constante(s) de la classe Affichage:

- LARG : Largeur de l'interface
- HAUT : Hauteur de l'interface
- WIDTH : Largeur de la voiture
- HEIGHT : Hauteur de la voiture

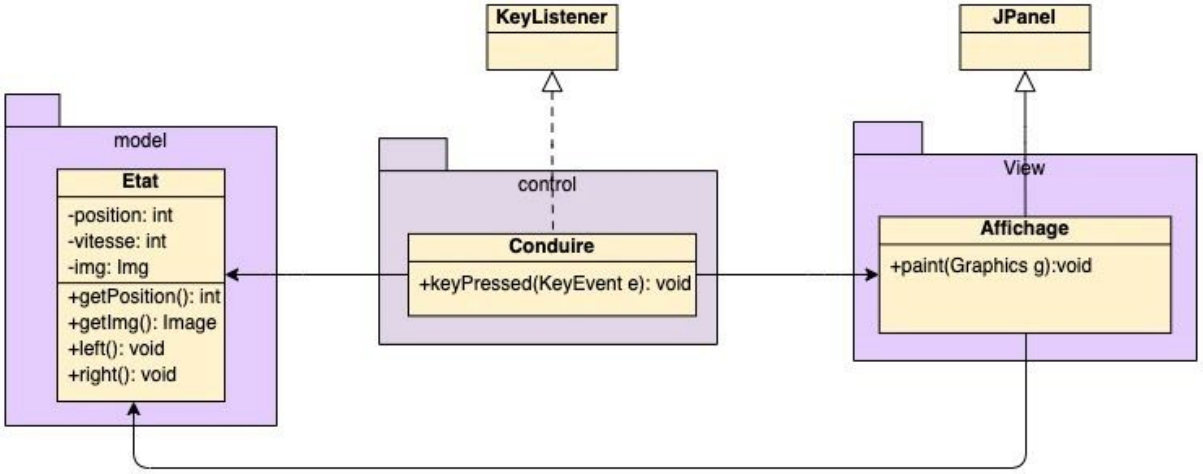
Voici un diagramme de classe pour afficher une fenêtre avec un véhicule dedans.



Pour le déplacement du véhicule, j'utilise la programmation événementielle avec la classe `KeyListener` et la distance est définie dans une constante.

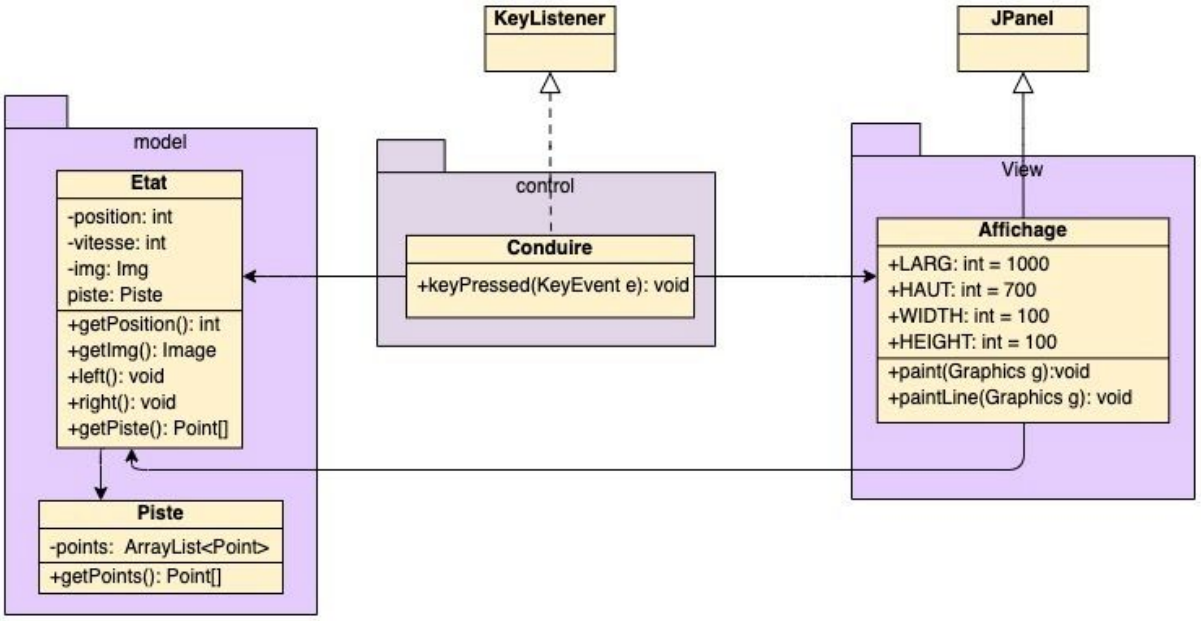
Constante(s) de la classe Etat :

- DEPLACE : La valeur du déplacement de quelques pixels du véhicule



Pour le défilement automatique de la piste, je dois d'abord générer aléatoirement les coordonnées et j'ai proposé un algorithme.

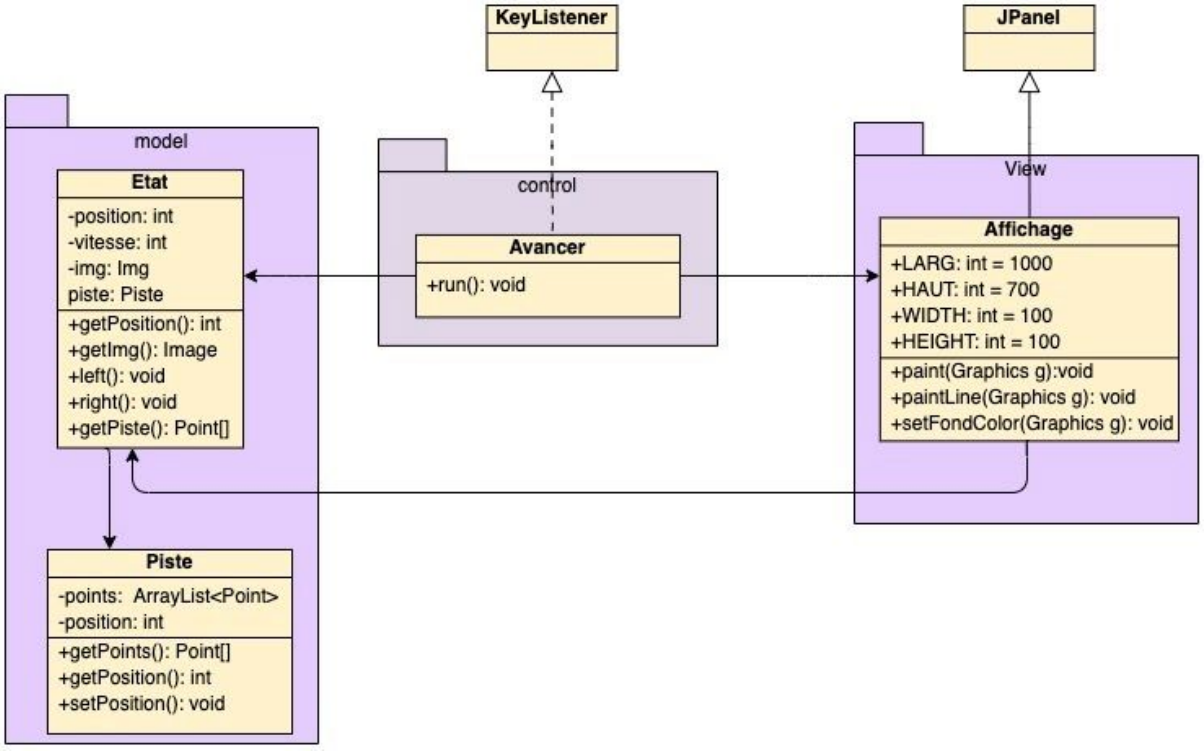
```
piste()  : void

    x <- new Random().nextInt(50) + Affichage.LARG/2 - 50

    y <- Affichage.HAUT

    Ajouter Point(x,y) dans la liste des points

    tant que  y >= Affichage.HORIZON faire

        x <- new Random().nextInt(50) + Affichage.LARG/2 - 50

        y <- y - new Random().nextInt(30) + 30;

        Ajouter Point(x,y) dans la liste des points

    fin tant que
```



Ensuite j'ai besoin d'un autre thread pour avancer la piste, elle devrait être infinie et calculée à partir d'une ligne brisée verticale limitée par l'horizon et générée aléatoirement. Et la valeur du mouvement est définie dans une constante.

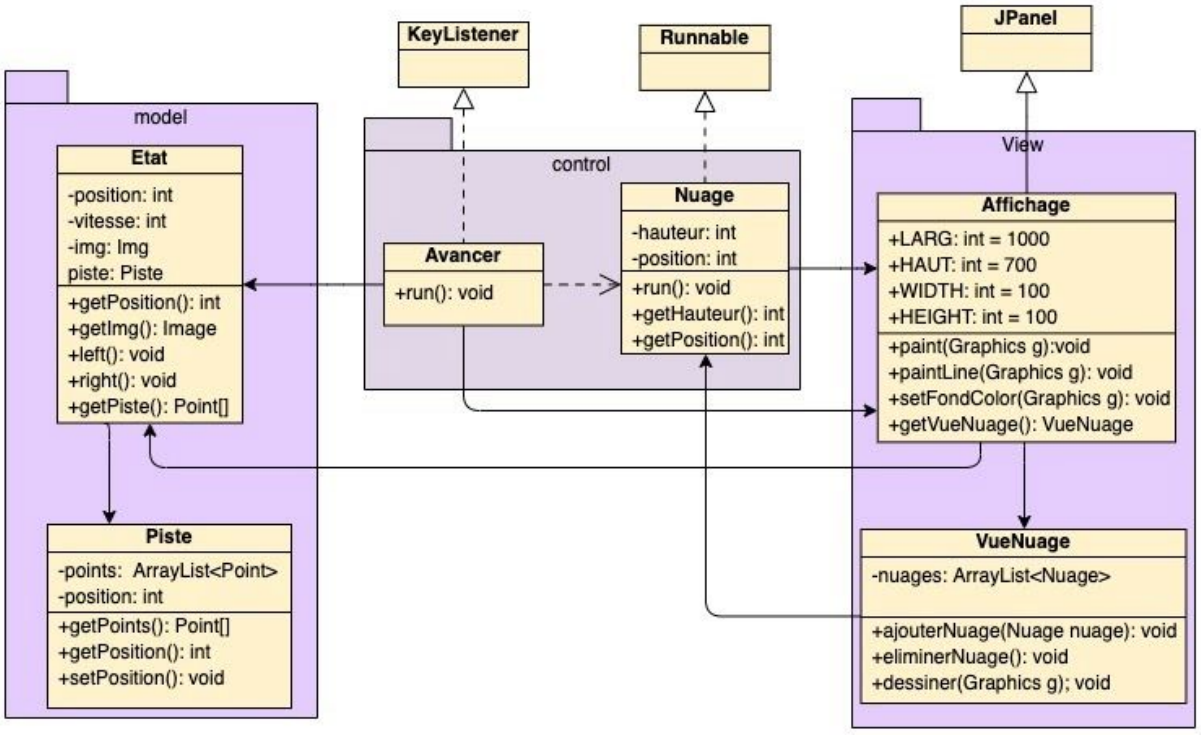
Constantes de la classe Piste :

- AVANCE : la valeur de l'incrément de la position de la piste

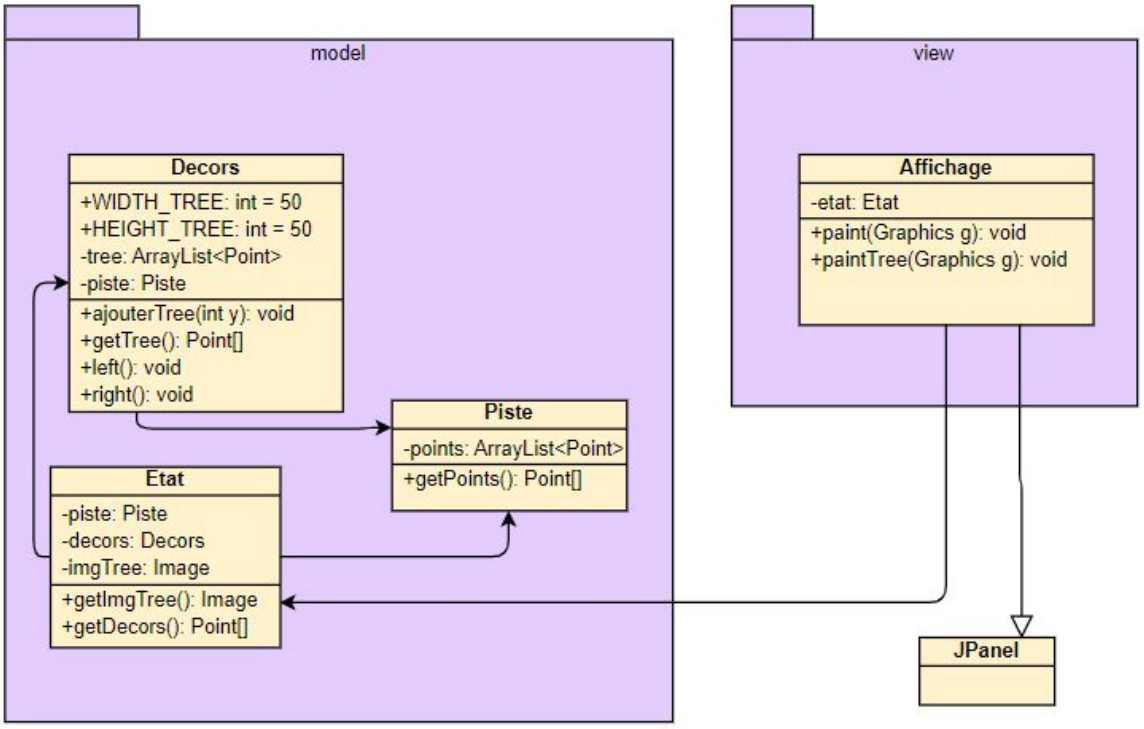


Nous avons un attribut *position* dans la classe Piste afin de savoir la kilomètre et on l'affiche dans la classe Affichage en utilisant la fonction *drawString.*

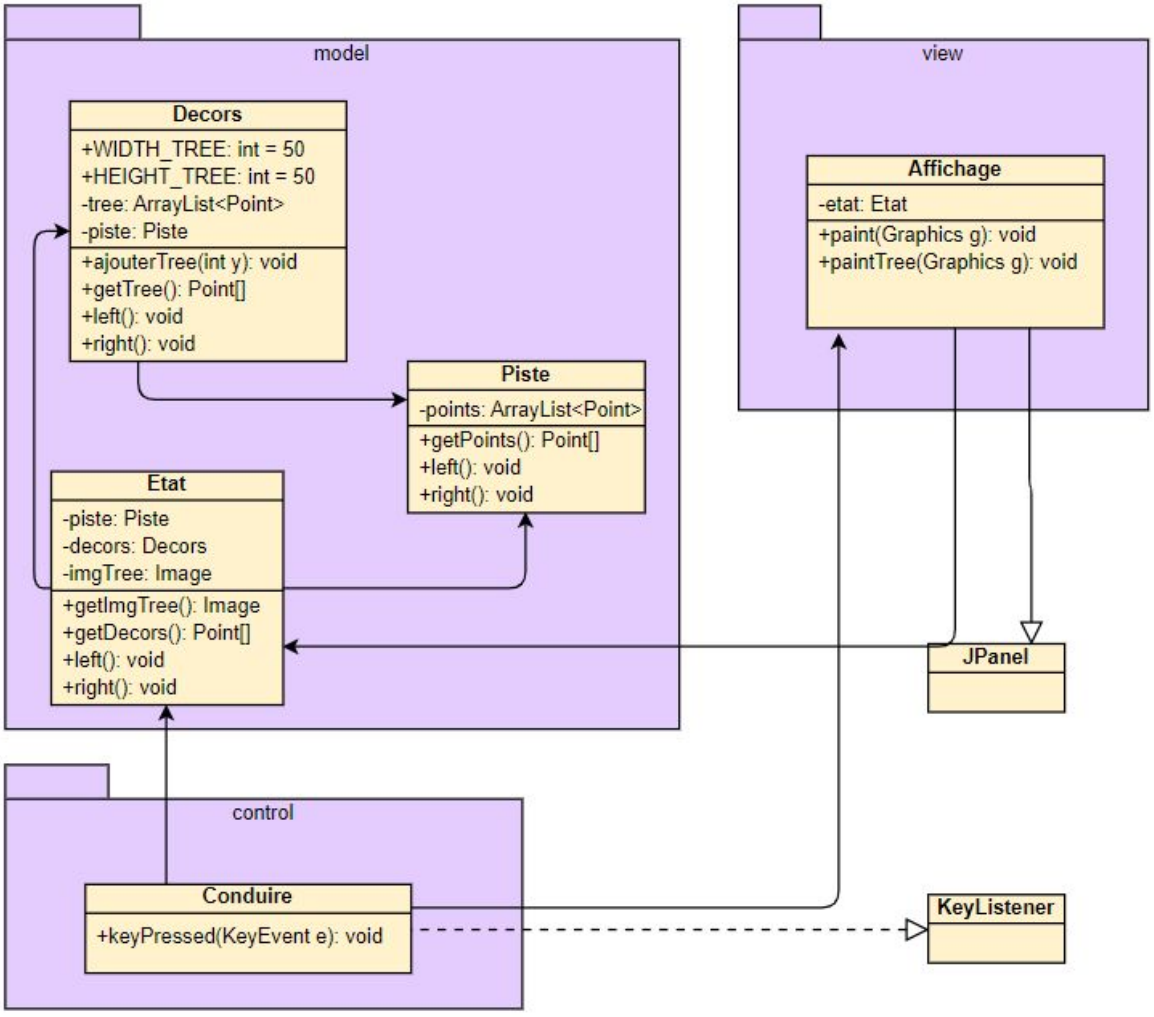
Maintenant, il est temps de rejoindre le décor. Afin d'améliorer la qualité de l'affichage, nous utilisons des images plutôt que des dessins faits à la main. On télécharge une image du nuage dans le projet. Nous créons ensuite une classe `Nuage` qui implémente un *thread*. La vue du MVC doit alors afficher les nuages qui se déplacent dans le ciel, de droite à gauche. Et munir la classe `Avancer` d'un générateur aléatoire de nuages pour avoir des nuages qui défilent sur l'écran.



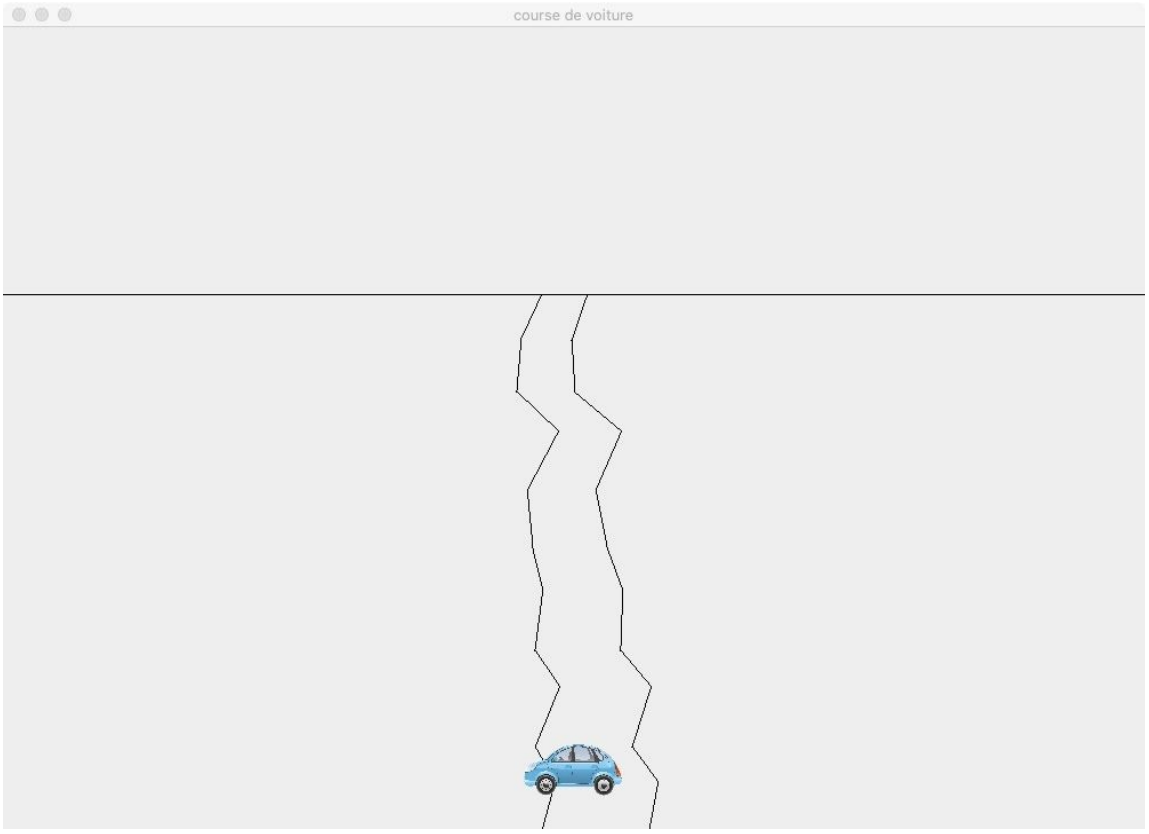
Pour la création du décors(des arbres sont des 2 cotés de la piste), on télécharge une image de l'arbre dans le projet. Les positions des arbres doivent être aléatoires, mais ils ne doivent pas être sur la route. Nous obtenons d'abord l'ordonnée de l'arbre au hasard, puis nous trouvons les points de la route des deux côtés de cette ordonnée. Ensuite, nous calculons l'abscisse de la route correspondant à cette ordonnée. Les abscisses aléatoires doivent être évitées sur la route. Dans la classe Affichage, obtenez la liste des coordonnées des arbres dans la classe decors via la classe etat, puis dessinez les arbres.

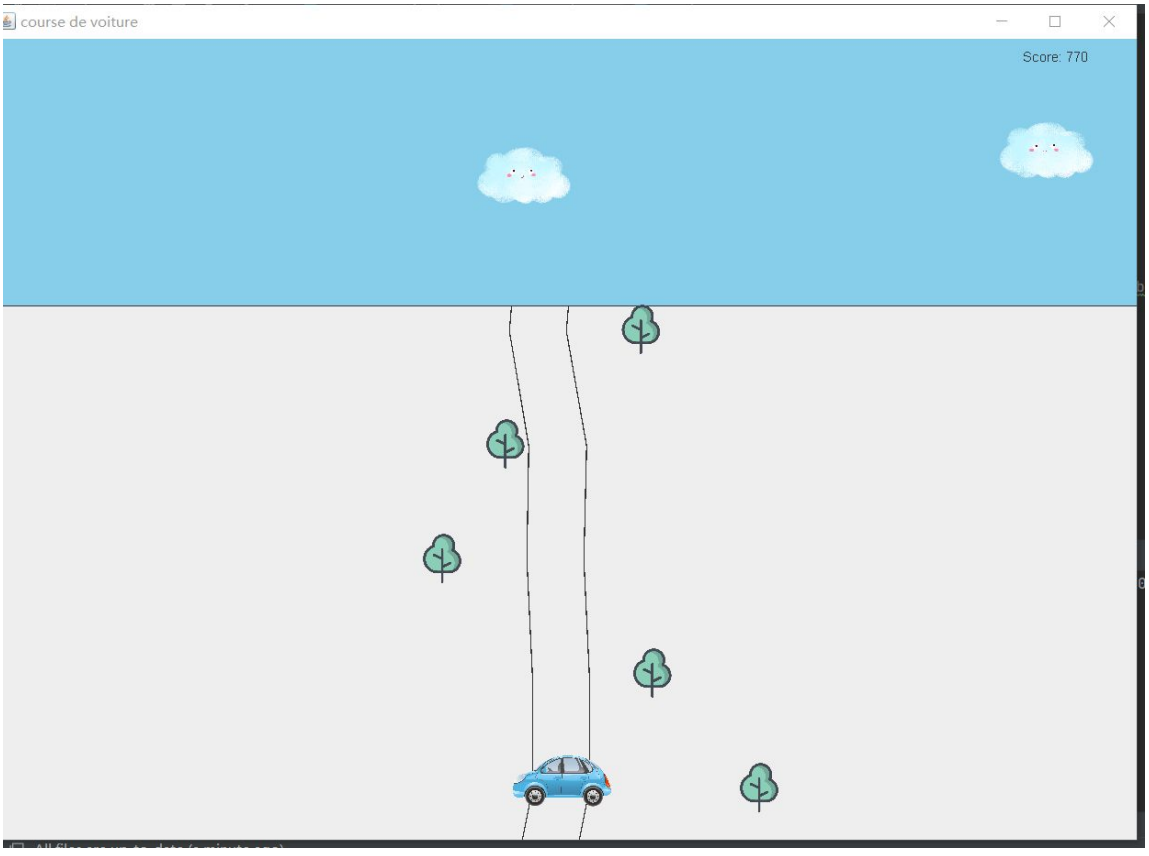


Pour le mouvements du décors de fond selon les touches du clavier, nous écrivons respectivement les fonctions left () et right () dans la classe décors et la classe Piste, et les appelons en classe etat, de sorte que lorsque la voiture se déplace à gauche ou à droite, la piste et les décors se déplacent à droite ou à gauche.


model
Decors
+WIDTH_TREE: int = 50
+HEIGHT_TREE: int = 50
-tree: ArrayList<Point>
-piste: Piste
+ajouterTree(int y): void
+getTree(): Point[]
+left(): void
+right(): void
Piste
-points: ArrayList<Point>
+getPoints(): Point[]
+left(): void
+right(): void
Etat
-piste: Piste
-decors: Decors
-imgTree: Image
+getImgTree(): Image
+getDecors(): Point[]
+left(): void
+right(): void
view
Affichage
-etat: Etat
+paint(Graphics g): void
+paintTree(Graphics g): void
JPanel
control
Conduire
+keyPressed(KeyEvent e): void
KeyListener

Résultat





Documentation utilisateur

- Prérequis : Java avec un IDE (ou Java tout seul si vous avez fait un export en .jar exécutable)
- Mode d'emploi (cas IDE) : Importez le projet dans votre IDE, sélectionnez la classe Main à la racine du projet puis « Run as Java Application ». Appuyez sur les touches gauche et droite du clavier pour contrôler le mouvement du véhicule
- Mode d'emploi (cas .jar exécutable) : double-cliquez sur l'icône du fichier .jar. Appuyez sur les touches gauche et droite du clavier pour contrôler le mouvement du véhicule

Documentation développeur

C'est la classe Main qui contient la méthode main. J'utilise le model MVC pour organiser mes codes.

Allez dans le package view pour voir comment afficher l'interface. Vous pouvez modifier les valeurs des constants définis dans la classe *Affichage* afin de changer la taille de la fenêtre et celle du véhicule.

Le package model définit l'ensemble des données qui caractérisent l'état de l'interface. La modification de ces données correspond à un changement de l'affichage dans l'interface graphique. Vous pouvez contrôler la vitesse du déplacement à gauche ou à droite du véhicule en modifiant la constante DEPLACE de la classe *Etat*. De même, vous pouvez constater le changement de la vitesse du mouvement de la ligne si vous changez le constante AVANCE de la classe *Parcours*.

Le package control vous permet de savoir comment gérer les événements et la manière dont l'état du modèle change.

La prochaine fonctionnalité sera le calcul de l'accélération en fonction de la position par rapport à la piste puis calcul de la vitesse.

Conclusion et perspectives

Nous avons réalisé une analyse globale de ce projet et formulé un plan de développement.

Ensuite, nous avons réalisé les fonctionnalités de la création d'une fenêtre dans laquelle est dessiné le véhicule, l'horizon et la piste et du déplacement du véhicule à droite et à gauche lorsqu'on tape les flèches du clavier.

Ce qui était difficile, c'est que la piste devrait être infinie et calculée à partir d'une ligne brisée verticale limitée par l'horizon, afin de le résoudre, nous avions caché la partie au-delà de l'horizon à l'aide de la fonction *fillRect*.

Nous avons aussi réalisé les fonctionnalités de l'Animation de la piste à vitesse constante, la création du décors, mouvements du décors de fond selon les touches du clavier et décompter des kilomètres et affichage.

Mais nous avons toujours une erreur. Même si nous avons écrit la limite de l'abscisse de l'arbre généré dans le code. Mais il y a encore des arbres qui apparaissent sur la route. C'est peut-être parce que les limites que nous avons écrites ne sont pas assez claires et doivent être améliorées.