université
PARIS-SACLAY

Rapport de projet PCII

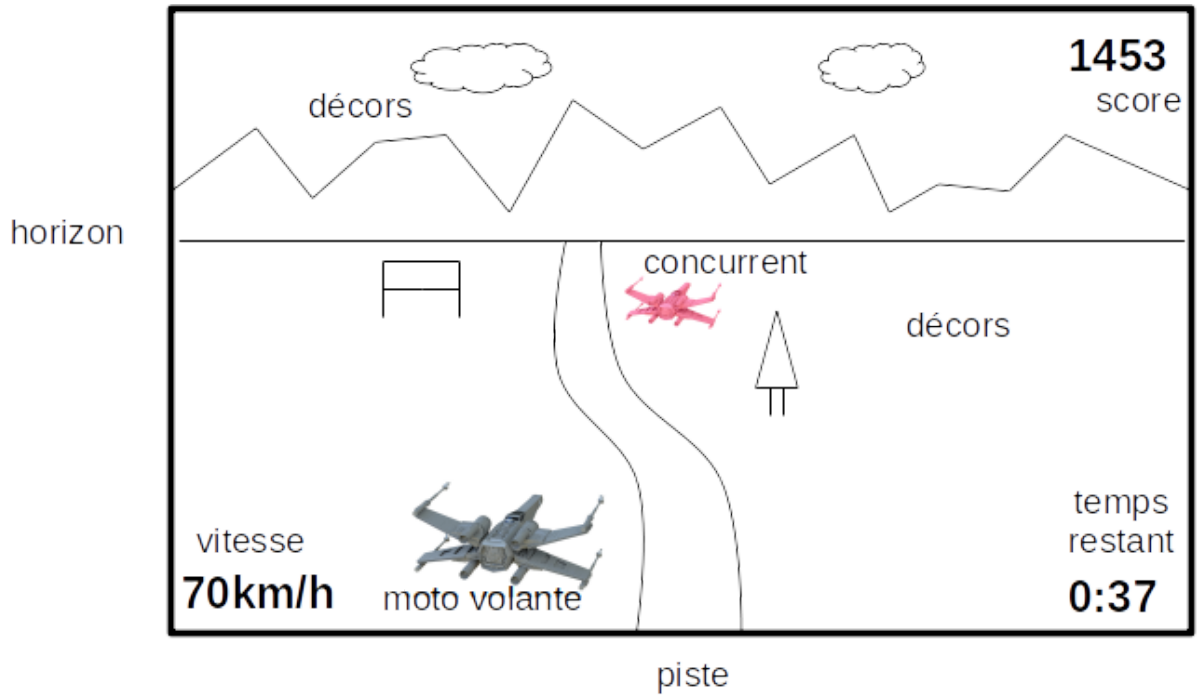
Réalisé par
Hongyu YAN, Shiqing HUANG

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.

Et puis, nous voulons réaliser les fonctionnalités qui sont compliqué :

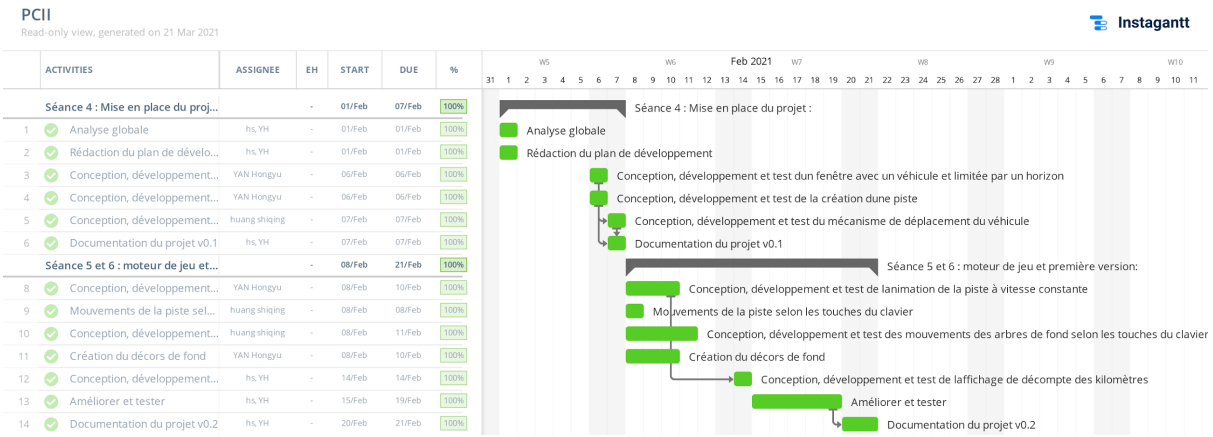
- Ajout d’obstacles et détection de collisions qui ralentissent le véhicule
- Ajout de points de contrôles (représentés par une ligne horizontale sur la piste)
- Calcul de l’accélération du véhicule en fonction de la position par rapport à la piste

Plan de développement

Liste des tâches:

- Séance 4 : Mise en place du projet
    - Analyse globale y,h
    - Rédaction du plan de développement y,h
    - Conception, développement et test d'un fenêtre avec un véhicule et limitée par un horizon y
    - Conception, développement et test de la création d'une piste y
    - Conception, développement et test du mécanisme de déplacement du véhicule h
    - Documentation du projet v0.1 y,h

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



- Séances 7 et 8 : mécanique du jeu
    - Conception, développement et test du changement de l’accélération et la vitesse y
    - Conception, développement et test de la détection de collisions y
    - Conception, développement et test du mécanisme de points de contrôles h
    - Conception, développement et test du mécanisme d’un décompte de temps h
    - Documentation du projet v0.9 y,h

- 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.

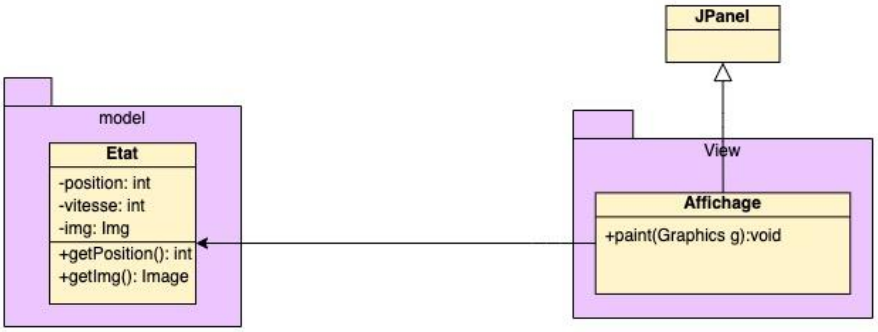
Conception détaillée

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

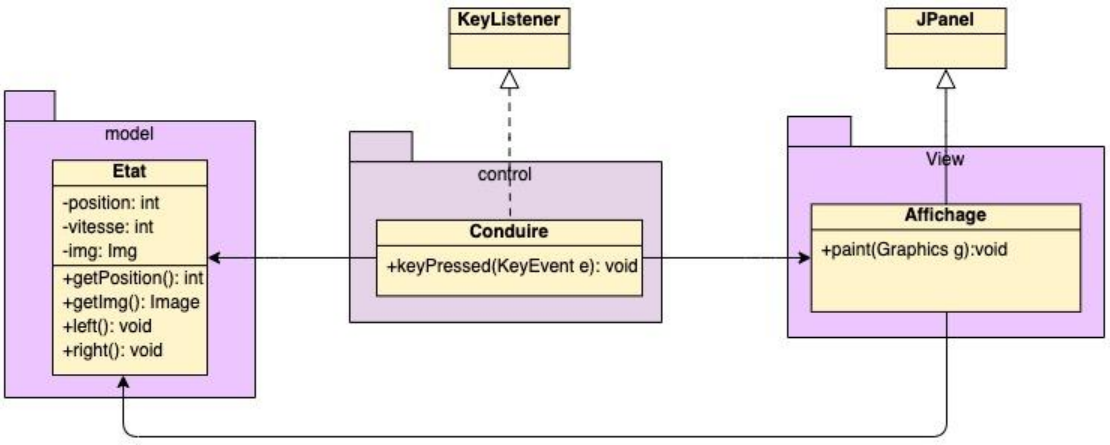
Voici un diagramme de classe pour afficher une fenêtre avec un véhicule dedans.



Pour le déplacement du véhicule, nous utilisons 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

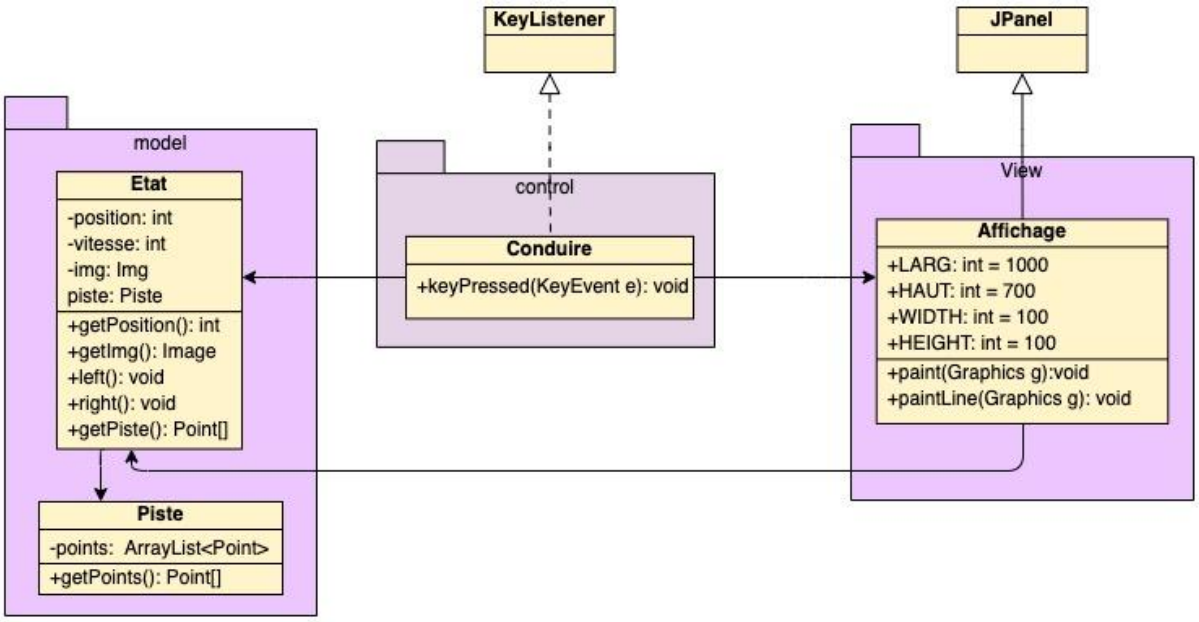
tant que y >= Affichage.***HORIZON*** **faire**

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

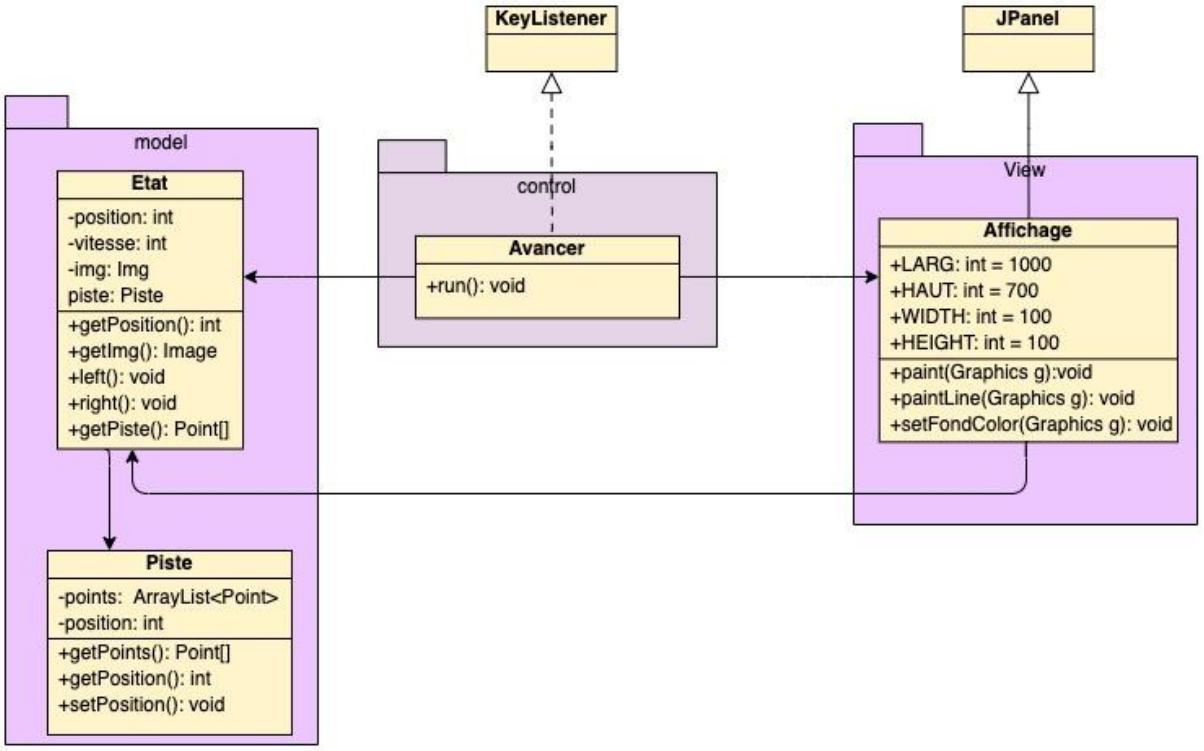
fin tant que



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.

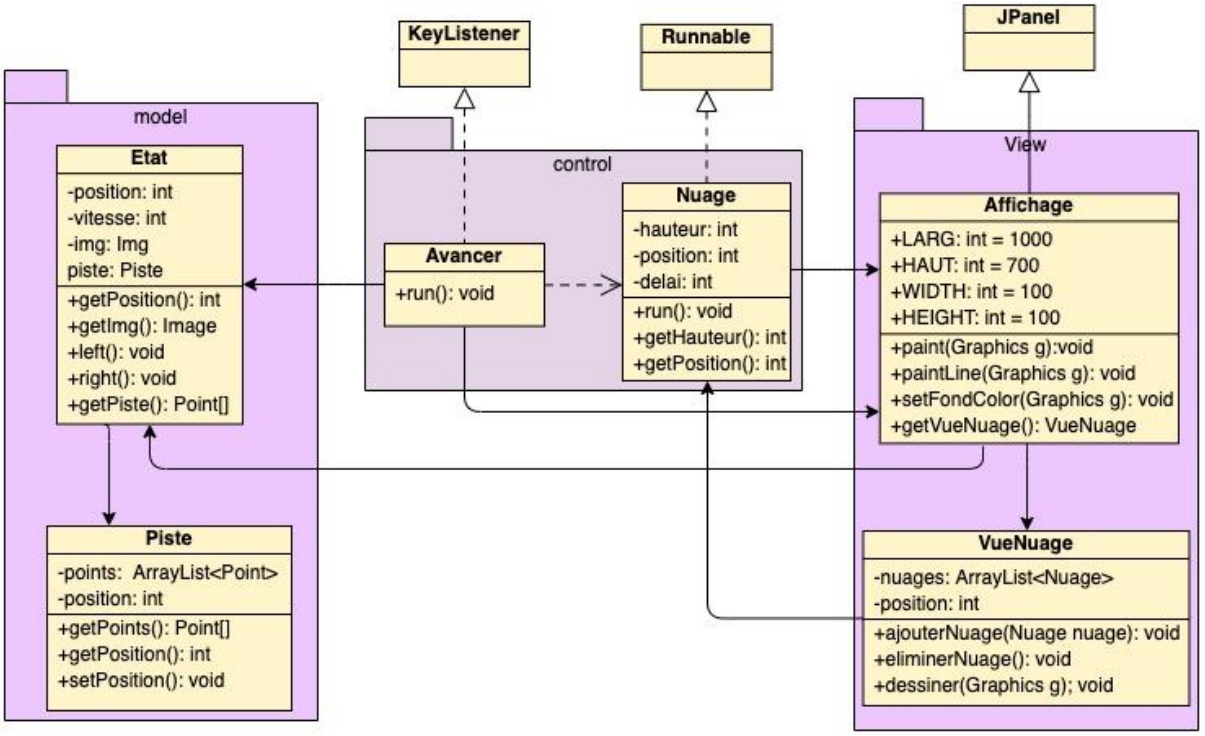
Constantes de la classe Piste :

- 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. Le constructeur de la classe `Nuage` choisit une valeur aléatoire pour le délai et la hauteur pour que les nuages aient des vitesses et des hauteurs différentes. Et munir la classe `Avancer` d'un générateur aléatoire de nuages pour avoir des nuages qui défilent sur l'écran.



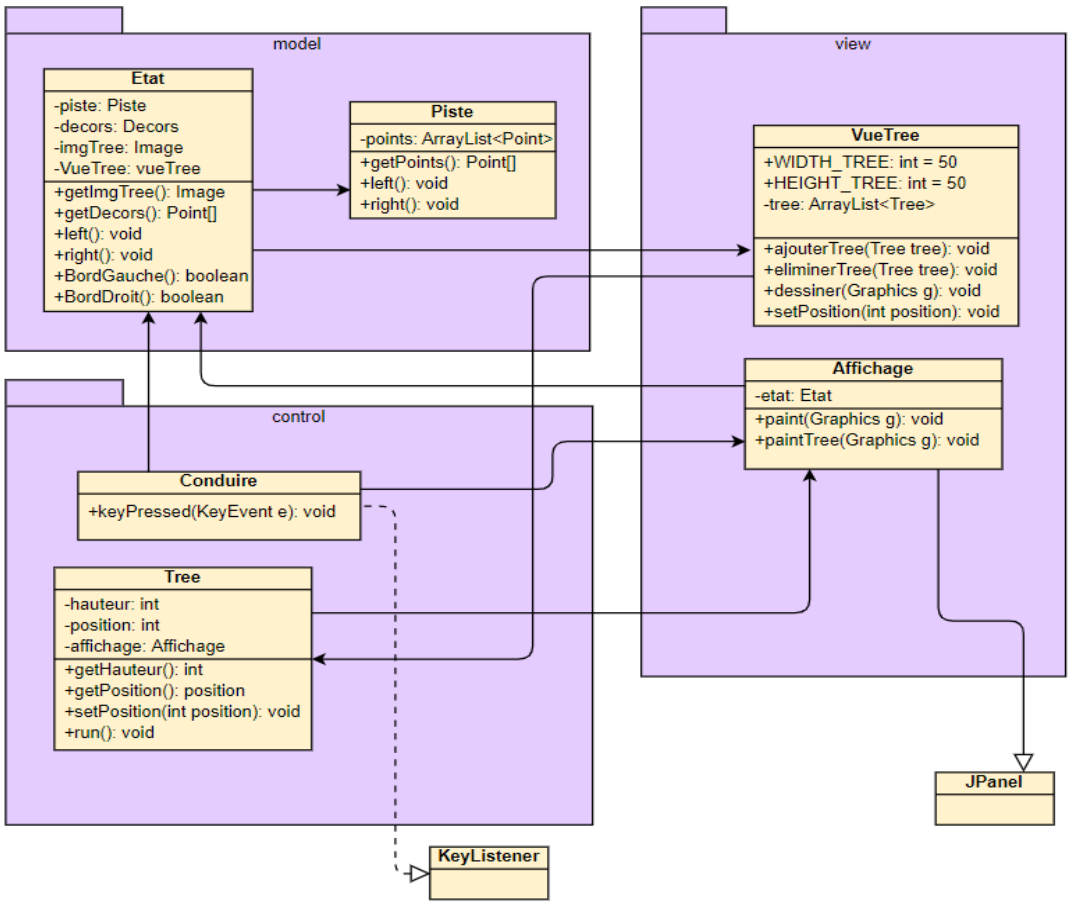
Ensuite, nous voulons ajouter des arbres des deux côtés de la route. Nous créons ensuite une classe `Tree` qui implémente un *thread*. La vue du MVC doit alors afficher les arbres. Dans la classe Tree, la hauteur initiale des arbres est fixée égale à l'horizon, mais la position de l'arbre est aléatoire, afin de faire apparaître l'arbre à partir de différentes positions sur l'horizon. Afin d'éviter que l'arbre n'apparaisse sur la route, il faut limiter la valeur de la position de l'arbre. Et munir la classe `Avancer` d'un générateur aléatoire de arbres pour avoir des arbres qui défilent sur l'écran.

Pour limiter la position de l'arbre:

```
Point[] points <- piste.points;
double x1 <- 0.0, y1 <- 0.0, x2 <- 0.0, y2 <- 0.0, x_piste;
double k, b;
Pour i de 0 à points.length - 1 faire
    Si (points[i].y >= Affichage.HORIZON
       && points[i+1].y <= Affichage.HORIZON) alors
    /*(x1, y1),(x2, y2) sont 2 points sur le côté gauche de la route,
    Affichage.HORIZON est entre ces deux points*/
    x1 <- points[i].x;
```

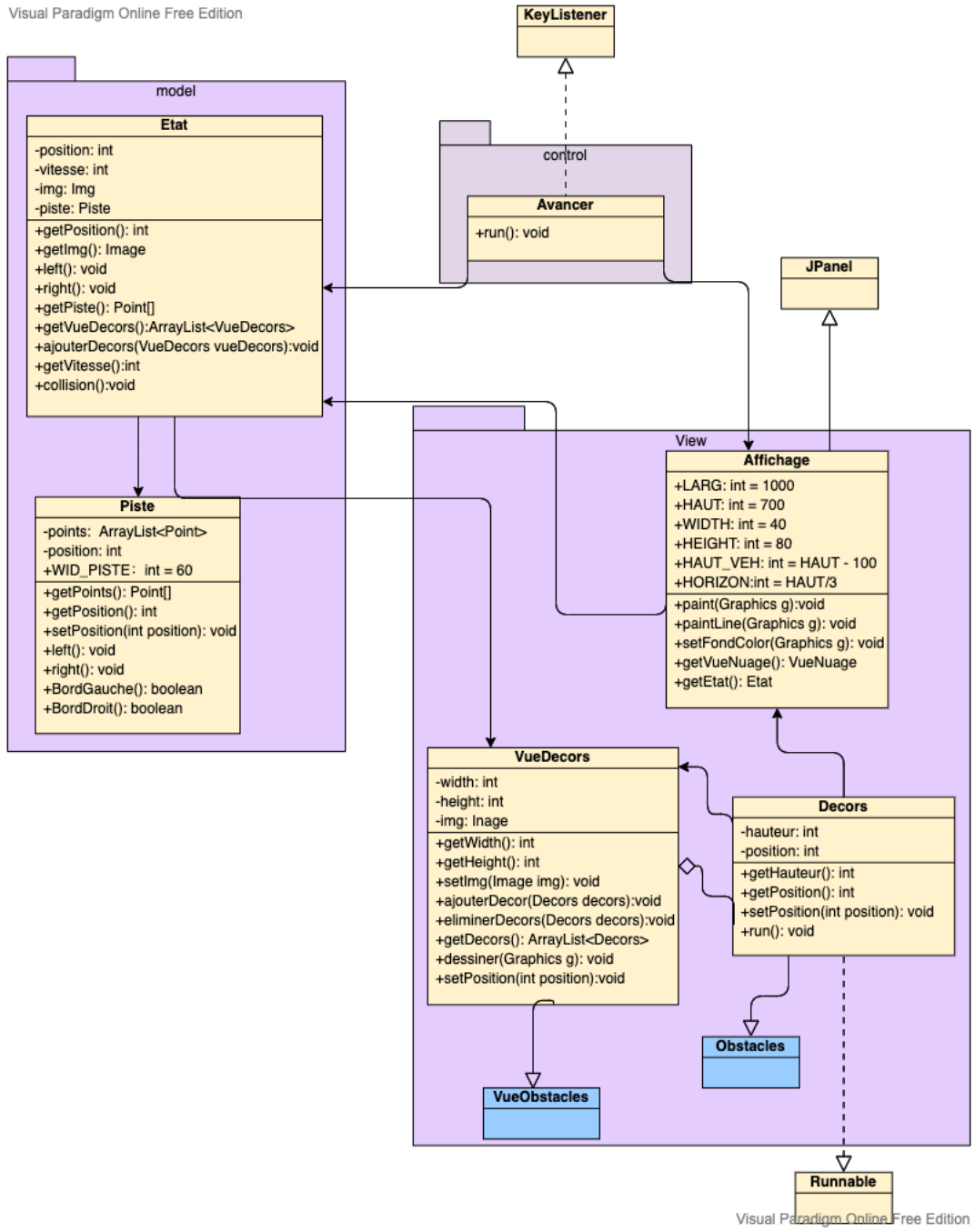
```
            x2 <- points[i+1].x;
            y1 <- points[i].y;
            y2 <- points[i+1].y;
        k <- (y2 - y1) / (x2 -x1);
        b <- (x2*y1 - x1*y2)/(x2 - x1);
        x_piste <- (Affichage.HORIZON - b)/k;
      Faire
          this.position <- new Random().nextInt(300) + Affichage.LARG/2 - 150 ;
    Tantque position + WIDTH_TREE >= x_piste
          && position <= x_piste + WID_PISTE
```

Pour permettre à la piste et aux arbres de suivre le mouvement à gauche ou à droite de la moto et de se déplacer à droite ou à gauche. Pour la piste, nous avons ajouté deux fonctions left () et right () dans la classe Piste. Lorsque KeyPressed est surveillé, ces deux fonctions sont appelées. Pour les arbres, lorsque la position de la route change, l'arbre change avec le changement de route.



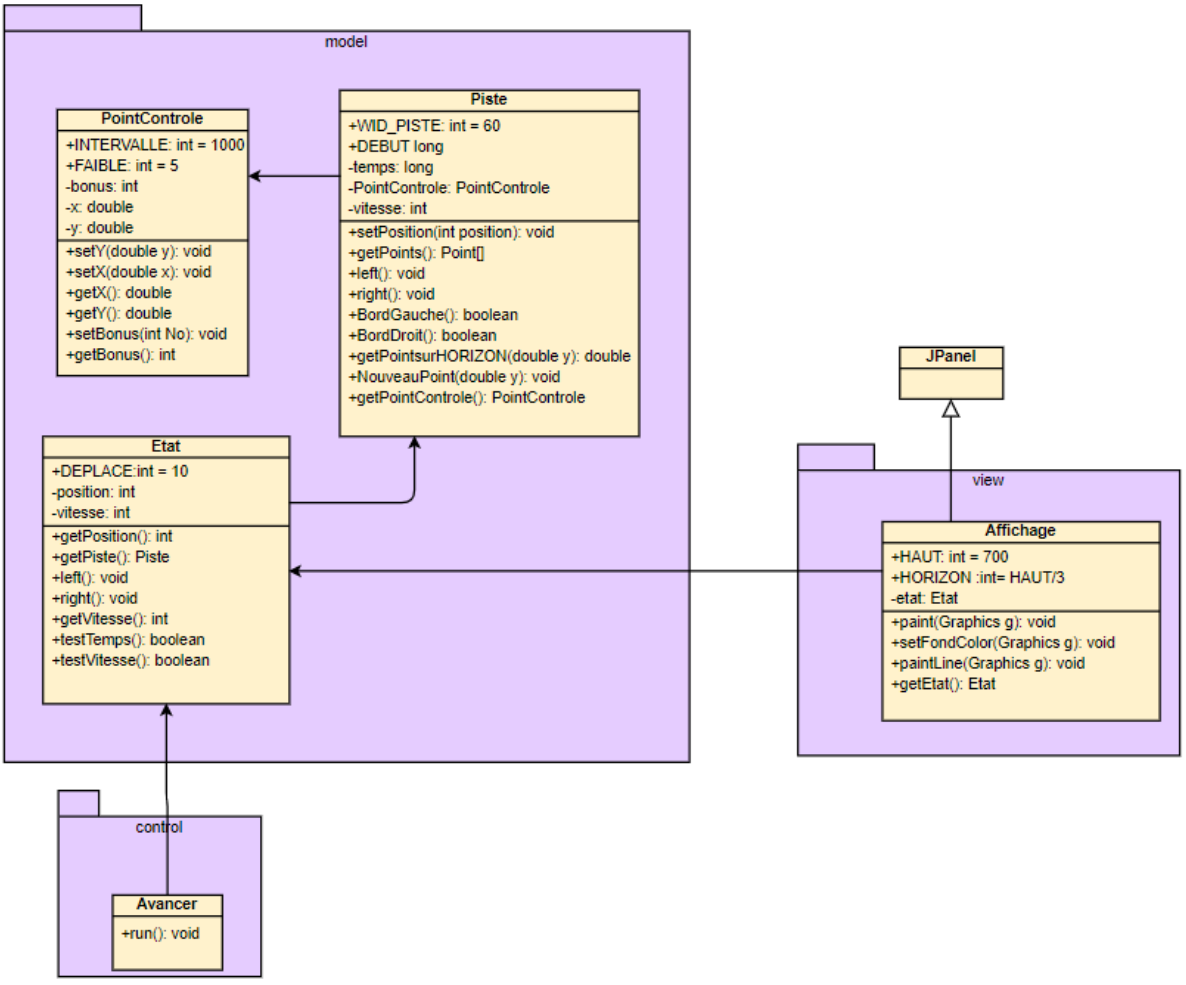
Pour l'ajout d'obstacles, afin de réduire la redondance du code, nous créons une classe Decors qui est père de la classe Arbre et la classe Obstacles et une classe VueDecors avec des fils VueArbre et VueObstacles. Alors dans la classe Etat, nous avons une liste qui contient tous les décors au sol (qui vont avancer avec la piste). Nous pouvons parcourir la liste quand on en a besoin.

C'est l'heure de détecter des collisions permettant de ralentir le véhicule. Nous avons une méthode collision dans la classe. On parcourt la liste des décors et s' il y a un obstacle, nous détectons si notre véhicule est entré en collision avec lui, si oui, on perd une vitesse fixe.



Maintenant, nous voulons ajouter des points de contrôle au jeu, pour cela nous avons ajouté une classe pour définir les états du point de contrôle. Nous jugeons dans la classe Avancer si le point de contrôle doit être dans la plage visible de la fenêtre. Si le point de contrôle est dans la plage visible de la fenêtre, les données du point de contrôle doivent être mises à jour avec le déplacement de la route, et elles doivent être affiliées dans la fenêtre de la classe Affichage.

Nous nous sommes souvenus de l'heure de début du jeu DEBUT lors de la création de Piste. Lorsque l'heure actuelle moins l'heure de début du jeu est égale au temps restant du jeu (ajout continu de bonus), nous arrêterons le jeu.



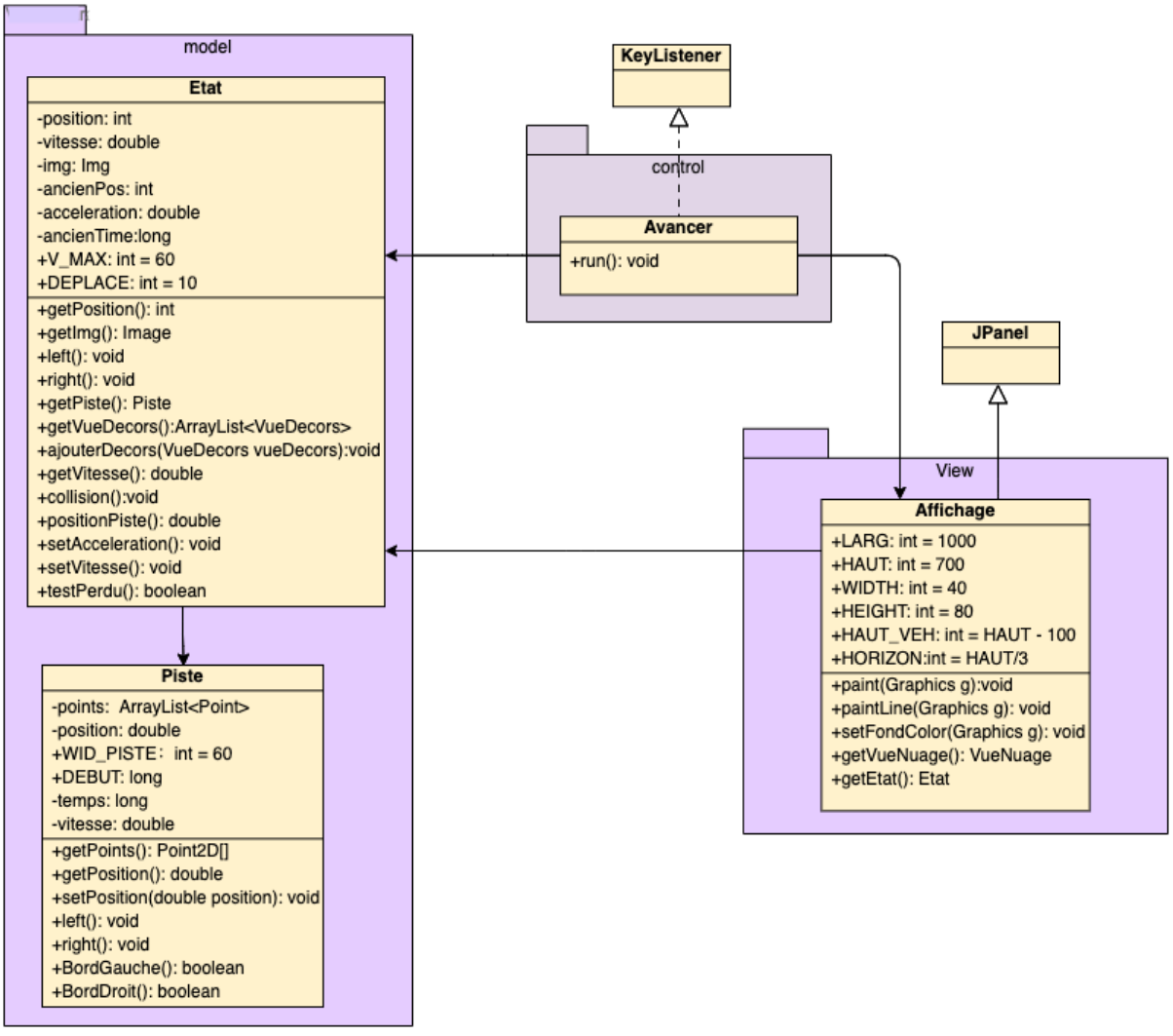
Pour réaliser le mécanisme de calcul de la vitesse du véhicule, nous avons une méthode positionPiste() afin d'obtenir l'abscisse gauche de la piste. Et puis , en utilisant une méthode setAcceleration() pour accomplir un mécanisme de calcul de l'accélération du véhicule en fonction de la position par rapport à la piste. Si le véhicule est sur la piste, l'accélération est 3. Sinon s'il est près de la piste, si

l'accélération est 1 sinon l'accélération est -1. Nous pouvons obtenir du temps en utilisant System.*currentTimeMillis*().

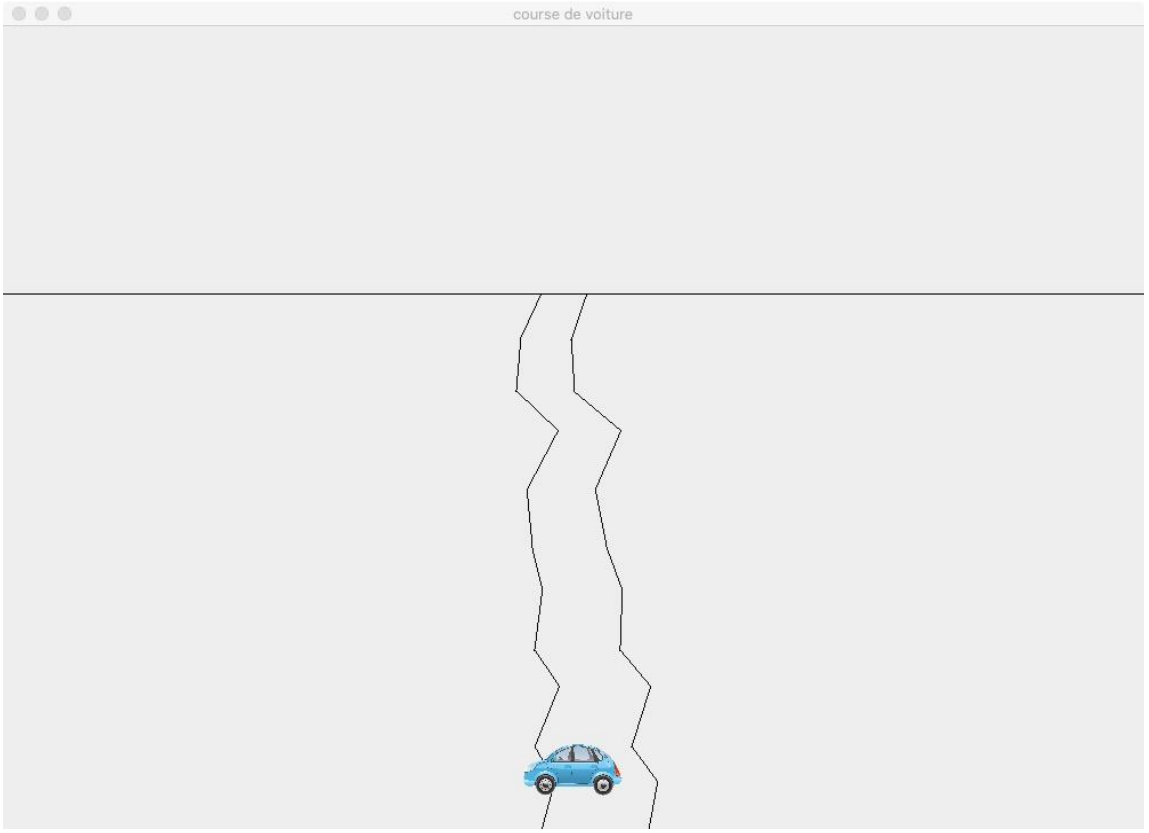
On sait la formule ci-dessous en connaissant l'ancienne vitesse, l'accélération et le temps :

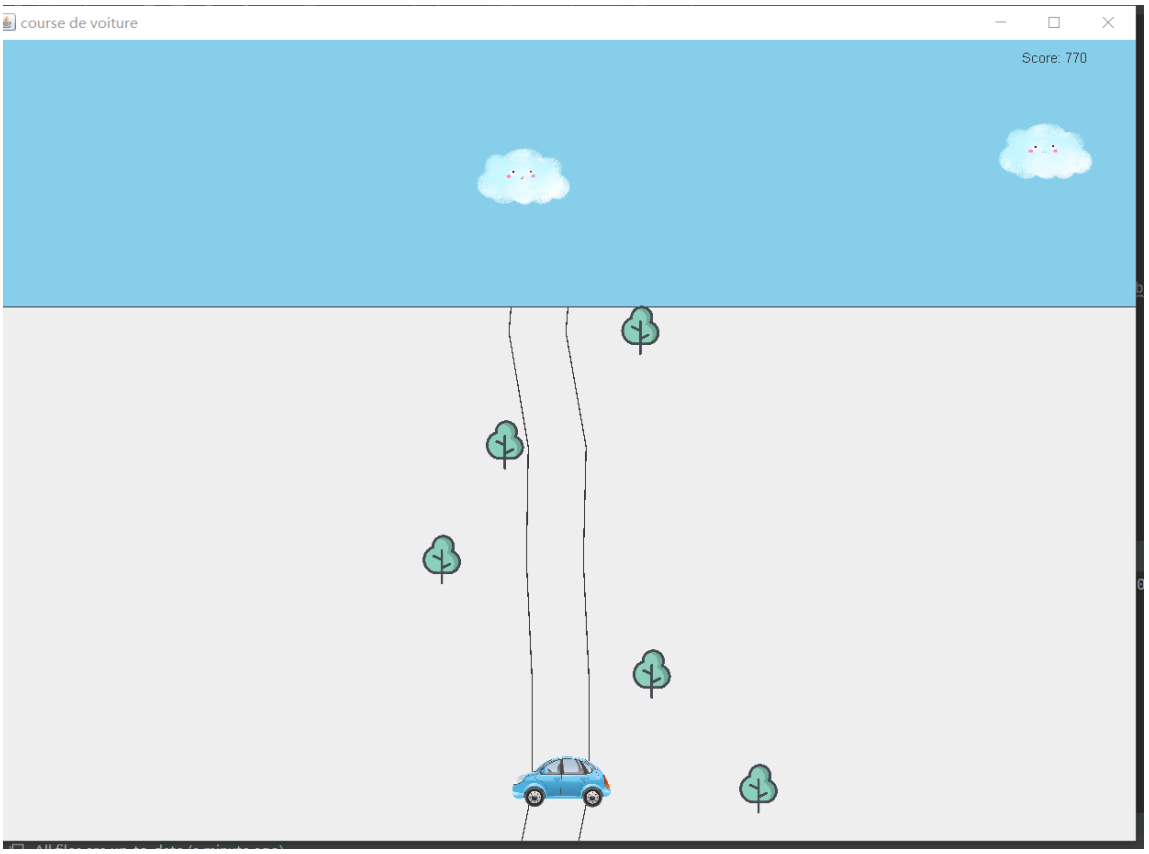
$$v = vi + at$$

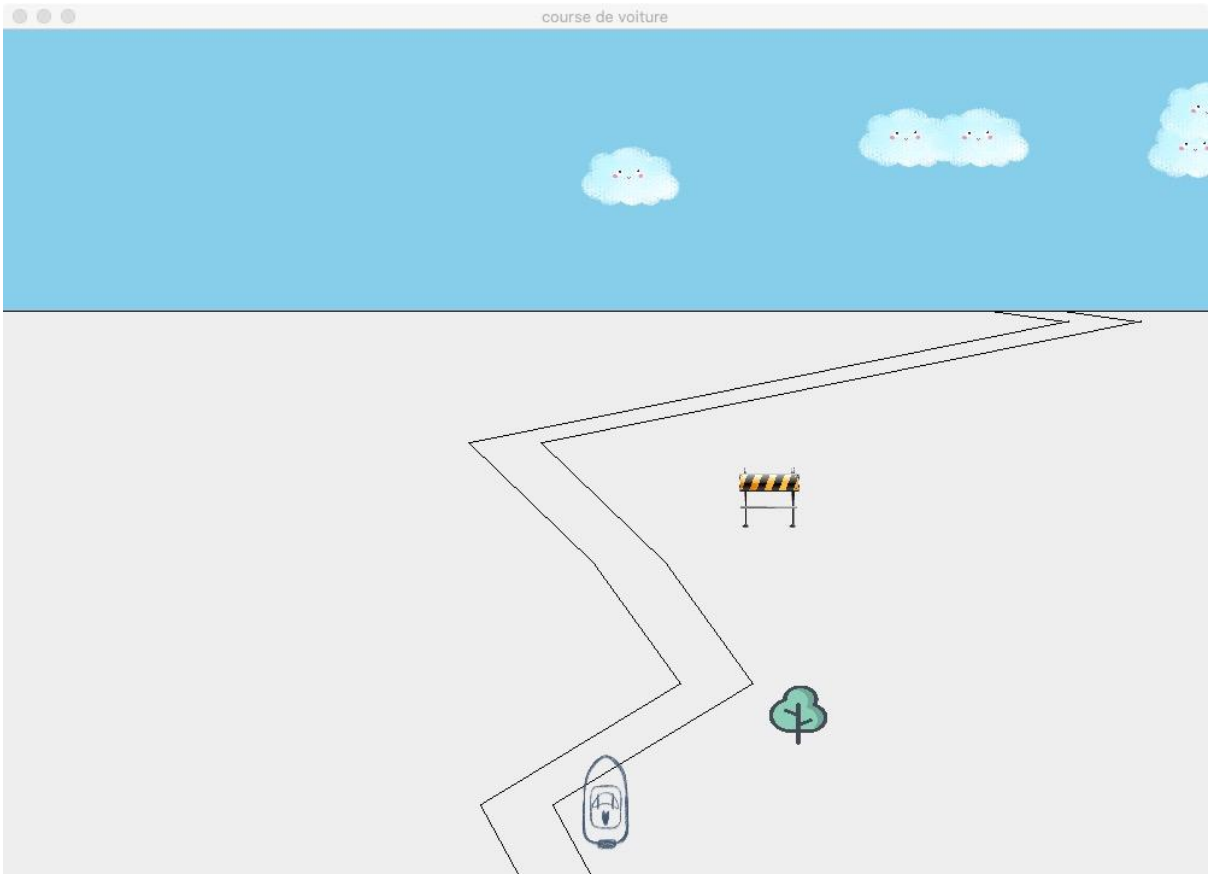
a = accélération, t = temps

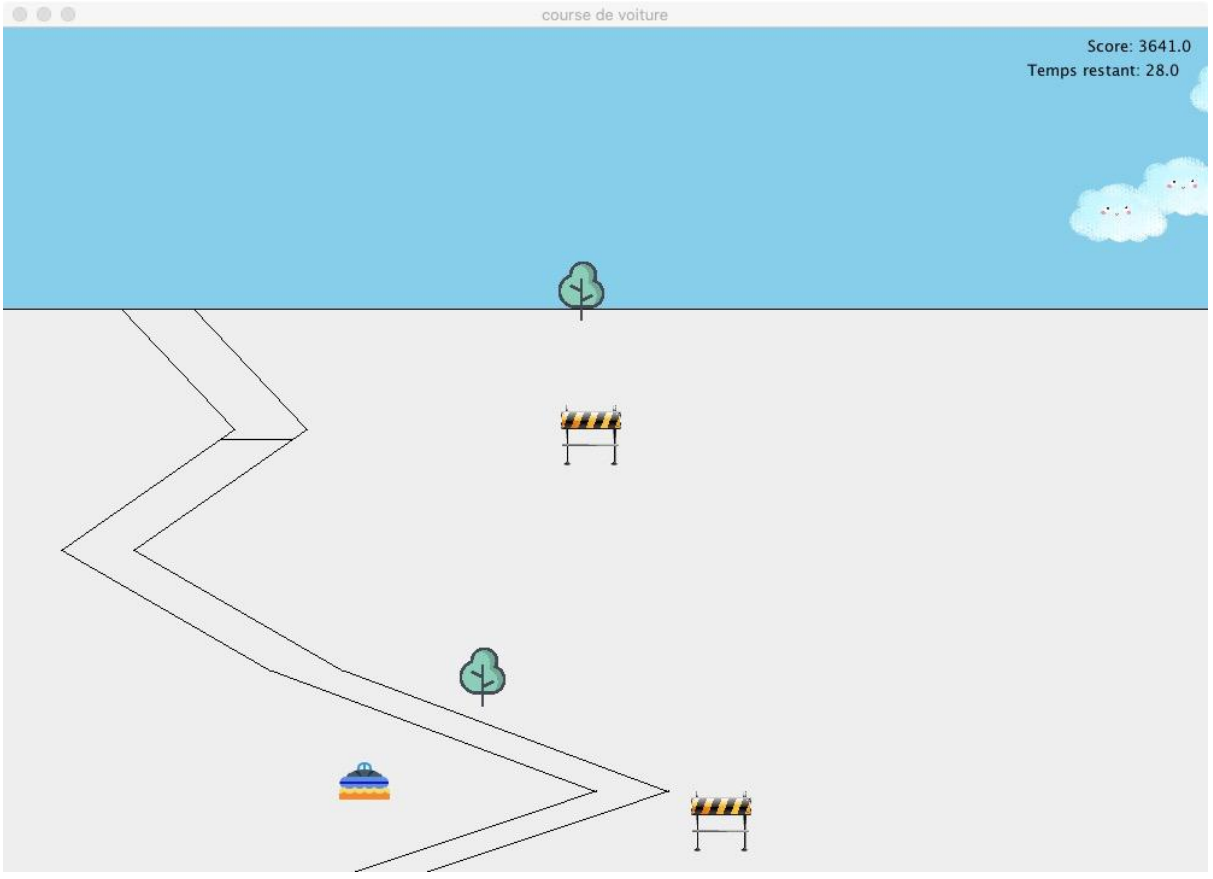


Résultat






course de voiture


course de voiture
Score: 3641.0
Temps restant: 28.0

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 FenetrePrincipal 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. Nous avons une classe père *VueDecors* afin d’afficher les décors qui avancent avec la piste. Elle a des fils *VueArbre* affichant des arbres et *VueObstacles* pour dessiner les obstacles qui peuvent causer la perte de vitesse 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*. Il est possible de changer la largeur de la piste avec la constante ***WID_PISTE***. Nous enregistrons l'heure de *DEBUT* du jeu et le temps restant dans la classe *Piste* pour déterminer si le jeu est terminé.

Le package control vous permet de savoir comment gérer les événements et la manière dont l’état du modèle change. L’animation de la piste et des décors et le mouvement du véhicule sont définis ici.

La prochaine fonctionnalité sera des améliorations, par exemple, courbes de Bézier.

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.

Nous avons constaté que la raison de l'erreur est que nous avons modifié la méthode de génération de piste afin que la limitation de génération d'arbres ne s'applique pas à la situation actuelle. Après ajustement et correction, nous avons résolu ce problème.

Nous faisons bien avancer le projet conformément à notre plan. Nous avons ajouté des obstacles et réduit la vitesse de la moto lorsqu'elle détecte que la moto entre en collision avec l'obstacle. Nous avons également réalisé la fonction de points de contrôle, où des points de contrôle apparaissent à intervalles réguliers. Notre jeu a donc commencé à être limité par le temps. Lorsque le temps restant est égal à zéro, le jeu se termine.