



---

# RAPPORT

---

Course de voiture



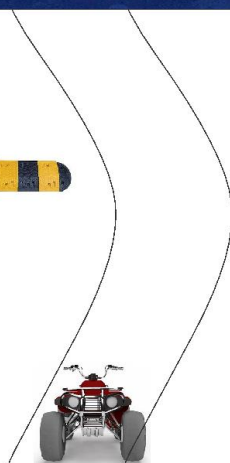
score

**171**



vitesse

**31.6 km/h**



temps restant

**00:29**

03/04/2021

Yajie LIU

Xiaoning MENG

## Table des matières

Introduction .....	2
Analyse globale .....	2
Plan de développement .....	3
Conception générale .....	5
Conception détaillée .....	8
Création d'une fenêtre et ajout les images .....	8
Déplacement de la voiture vers la droite ou gauche lorsqu'on utilise le clavier .....	9
Génération de la piste avec différentes pentes.....	10
La piste est infinie et elle peut avancer automatiquement .....	11
Gestion la vitesse de la voiture .....	12
Gestion du décor de fond.....	14
Transformation la ligne droite de la piste en courbe.....	15
Gestion du temps.....	16
Ajout l'écran d'accueil.....	18
Ajout les obstacles.....	18
Détection de la collision.....	19
Affichage des points de contrôles.....	19
Résultat .....	20
Documentation utilisateur .....	22
Documentation développeur .....	23
Conclusion et perspectives.....	24

## Introduction

Nous travaillons sur ce projet en binôme. L'objectif du ce projet est de réaliser un jeu vidéo des années 80 de type "course de voiture" en vue à la première personne. Les utilisateurs peuvent conduire une moto et utiliser le clavier pour contrôler le mouvement horizontal de la moto. La vitesse de la moto est calculée en fonction de la distance entre lui et la piste. Il y aura aussi des obstacles ou des concurrents sur la piste. Tout ce que les utilisateurs doivent faire est d'éviter les obstacles et de rendre la moto aussi proche que possible du centre de la piste pour obtenir la plus grande accélération possible. Les points de contrôle apparaîtront en fonction de la distance parcourue par la moto selon une certaine règle. Lorsque la moto atteint le point de contrôle, le temps restant sera incrémenté par une valeur constante. Une fois que la vitesse de la moto ou le temps restant passe à 0, le jeu se termine. Le score final dépend de la distance parcourue

## Analyse globale

### Analyse globale

Pour ce projet, il existe six fonctionnalités principales à développer :


- L'interface graphique avec la voiture, le piste, les décors et les labels pour afficher le temps, le score et la vitesse.
- Le défilement automatique de la piste.
- La réaction de la voiture aux contrôles du clavier de l'utilisateur.
- Un mécanisme de calcul de l'accélération et la vitesse de la voiture. (Difficile)
- L'apparition régulière des points de contrôles matérialisés par une bande horizontale sur la piste.
- Un mécanisme pour incrémenter et décrémenter le temps restant. (Difficile)


Il y a aussi des fonctionnalités complémentaires que nous pouvons réaliser :

- L'apparition aléatoire d'obstacles sur la piste.
- Les concurrents.
- La mémorisation des meilleurs scores
- L'inclinaison de la course de voiture et le mouvement du paysage lors des virages.
- L'ajout d'un écran d'accueil. (Facile)
- Le défilement du décor à droite et à gauche en fonction des mouvements du véhicule pour créer une sensation de virage

Dans la suite du développement, nous sélectionnerons certaines de ces fonctionnalités complémentaires à développer en fonction de l'avancement du projet, en partant du principe de veiller à ce que toutes les fonctionnalités nécessaires sont réalisées.

Rappel :  est la partie dont Yajie LIU est responsable.

 est la partie dont Xiaoning MENG est responsable.

 est la partie dont nous sommes responsables ensemble.

1. Découverte du sujet (30mn)
2. Analyse du problème (30mn)
3. Conception, développement et test d'un fenêtre qui affiche les labels et la course de voiture. (45mn)
4. Acquisition de compétence de dessiner une photo dans la fenêtre. (30mn)
5. Acquisition de compétence de keyListener. (45mn)
6. Squelette de l'application. (15mn)
7. Documentation du projet. (45mn)

[illegible]

1. Analyse du problème (30mn)
2. Animation de la piste à vitesse constante (30mn)
3. Modifier le problème sur l'affichage de la course de voiture (30mn)
4. Conception, développement et test le mise à jour l'affichage de vitesse est score (60mn)
5. Conception, développement et test le contrôle la voiture en utilisant clavier (30mn)
6. Gestion de la vitesse (45mn)
7. Documentation du projet (60mn)

255mn	270mn	285mn	300mn	315mn	330mn	345mn	360mn	375mn	390mn	405mn	420mn														
		Analyse du problème																							
						Animation de la piste																			
												Modifier le problème sur l'affichage de la piste													
												mise à jour l'affichage de vitesse est score													
																		contrôle la voiture en utilisant clavier							
435mn	450mn	465mn	480mn	495mn	510mn	525mn																			
			Gestion de la vitesse																						
															rapport du projet										

## Liste des tâches (Séance 3) :

1. Analyse du problème (30mn)
2. Mouvements du décor de fond selon les touches du clavier (45mn)
3. Transforme la ligne droite de la piste en courbe (60mn)
4. Résoudre le problème sur la vitesse devient soudainement 0 (30mn)
5. Gestion du temps restant (90mn)
6. Documentation du projet (60mn)

540mn	555mn	570mn	585mn	600mn	615mn	630mn	645mn	660mn	675mn	690mn													
		Analyse du problème				Mouvements du décor de fond selon les touches du clavier				Transforme la ligne droite de la piste en courbe		Résoudre le problème sur la vitesse devient soudainement 0											
705mn	720mn	735mn	750mn	765mn	780mn	795mn	810mn	825mn	840mn														
						Gestion du temps restant				Documentation du projet													

## Liste des tâches (Séance 4) :

1. Analyse du problème (30mn)
2. Ajouter des obstacles et laissez-le apparaître dans l'interface (60mn)
3. Détection de la collision (90mn)
4. Documentation du projet (45mn)

855mn	870mn	885mn	900mn	915mn	930mn	945mn	960mn	975mn	990mn	1005mn	1020mn						
		Analyse du problème				Ajouter des obstacles				détection de la collision							
1035mn	1050mn	1065mn															
			Documentation du projet														

## Liste des tâches (Séance 5) :

1. Analyse du problème (30mn)
2. Ajouter d'un écran d'accueil (45mn)
3. Ajouter de points de contrôles (représentés par une ligne horizontale sur la piste) (60mn)
4. Documentation du projet (60mn)

1080mn	1095mn	1110mn	1125mn	1140mn	1155mn	1170mn	1185mn	1200mn	1215mn	1230mn	1245mn	1260mn					
		Analyse du problème				Ajouter d'un écran d'accueil				Ajouter de points de contrôles		Documentation du projet					

Liste des tâches (Séances 6) :

1. Analyse du problème (45mn)
2. Modifier que la vitesse passe à 0 et le jeu est terminé (30mn)
3. Ajouter des images dans les décors (60mn)
4. Optimiser le code pour réduire l'erreur de génération d'accélération (60mn)
5. Documentation du projet (60mn)

1275mn	1290mn	1305mn	1320mn	1335mn	1350mn	1365mn	1380mn	1395mn			
			Analyse du problème								
			Modifier que la vitesse à 0 et le jeu est terminé								
						Ajouter l'image dans le décor					
1410mn	1425mn	1440mn	1455mn	1470mn	1485mn	1500mn	1515mn	1530mn			
			Optimiser le code pour réduire l'erreur de génération d'accélération								
						Documentation du projet					

## Conception générale

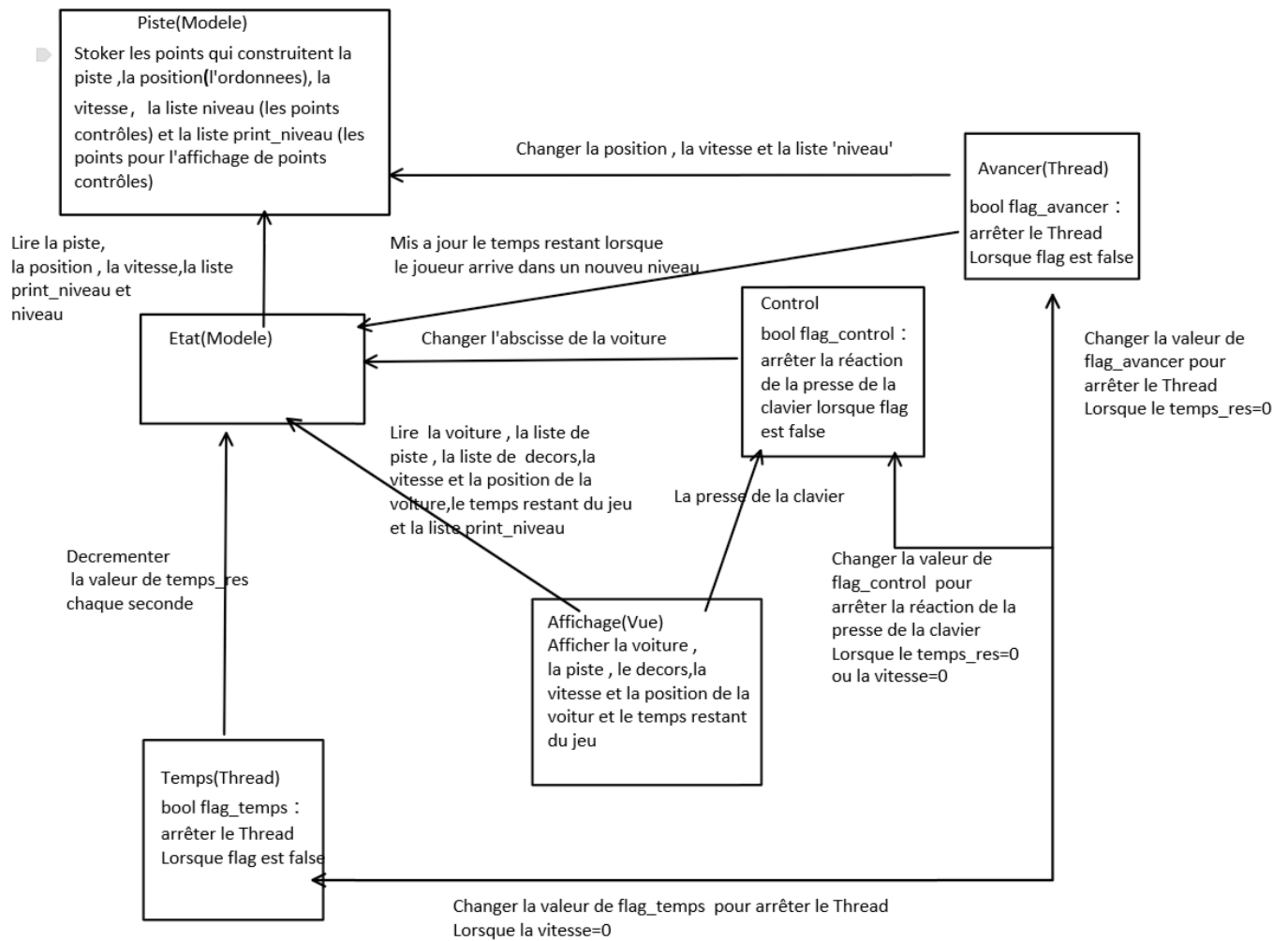
Nous avons adopté le motif MVC pour le développement de notre interface graphique.

Il y a trois parties : Modèle, Vue, Control.

1. Création d'une fenêtre dans laquelle est dessiné les éléments et ajout les images.  
Nous avons mis cette partie dans le Vue. Car la fenêtre et les éléments doivent être visibles par l'utilisateur.
2. Déplacement de la voiture vers la droite ou gauche lorsqu'on utilise le clavier.  
Nous avons mis cette partie dans le Contrôle. Cette fenêtre est interactive, donc les événements doivent être dans le contrôle pour réaliser.
3. Génération de la piste avec différentes pentes.  
Nous avons mis cette partie dans le Vue et Modèle. La partie de générer une piste est dans le Vue en utilisant Graphics. Pour les différentes pentes, les données doivent être modifiées, j'ai le mis dans le Modèle.
4. La piste est infinie et elle peut avancer automatiquement.  
Nous avons mis cette partie dans le Modèle. Pour la piste est infinie, il doit ajouter toujours les nouveaux points et supprimer les points inutiles, et pour avancer automatiquement, il est comme le déplacement de l'ovale (en tutoriel), il doit modifier des données.
5. Gestion la vitesse de la voiture.

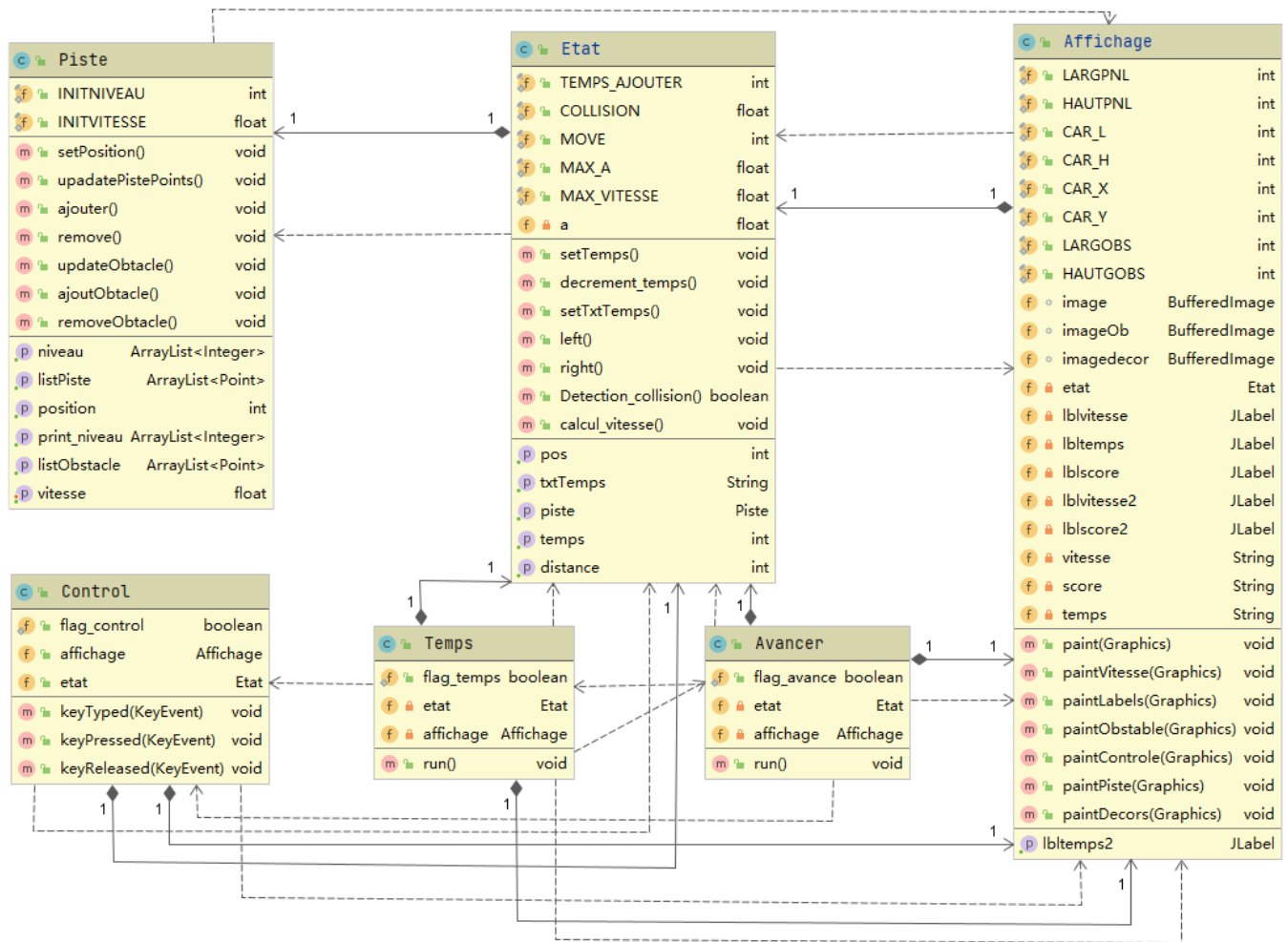
Nous avons mis cette partie dans le Modèle. Car nous devons modifier la vitesse de la piste en fonction de la distance entre la piste et la voiture. Les données doivent être modifiées toujours.

6. Mouvements du décor de fond selon les touches du clavier.  
Cette partie est comme la partie d'avancer la piste, mais il ne bouge pas automatiquement et est contrôlé par le clavier. Donc nous avons la mise dans le Control et Vue. Dans le Modèle pour changer la distance de chaque mouvement.
7. Transformation la ligne droite de la piste en courbe.  
Cette partie pour modifier la piste nous avons déjà dessiné qui dans le Vue.
8. Gestion du temps.  
Nous avons mis cette partie dans le Modèle. Car le temps change toujours.
9. Ajout l'écran d'accueil.  
Nous avons ajouté un autre écran dans le Vue.
10. Ajout des obstacles.  
Nous avons mis cette partie dans le Modèle. Car il doit ajouter toujours les nouveaux points pour déterminer la location d'obstacle et supprimer les points inutiles.
11. Détection de la collision.  
Nous avons mis cette partie dans le Modèle. Parce que nous devons vérifier de la progression du jeu en fonction de l'évolution des données
12. Affichage des points de contrôles  
Nous avons mis cette partie dans le Modèle.





## Conception détaillée



### 1. Création d'une fenêtre et ajout les images

Pour la fenêtre qui affiche les éléments nécessaires, nous utilisons l'API Swing et la classe JPanel. Nous définissons les dimensions de la fenêtre de la constante.

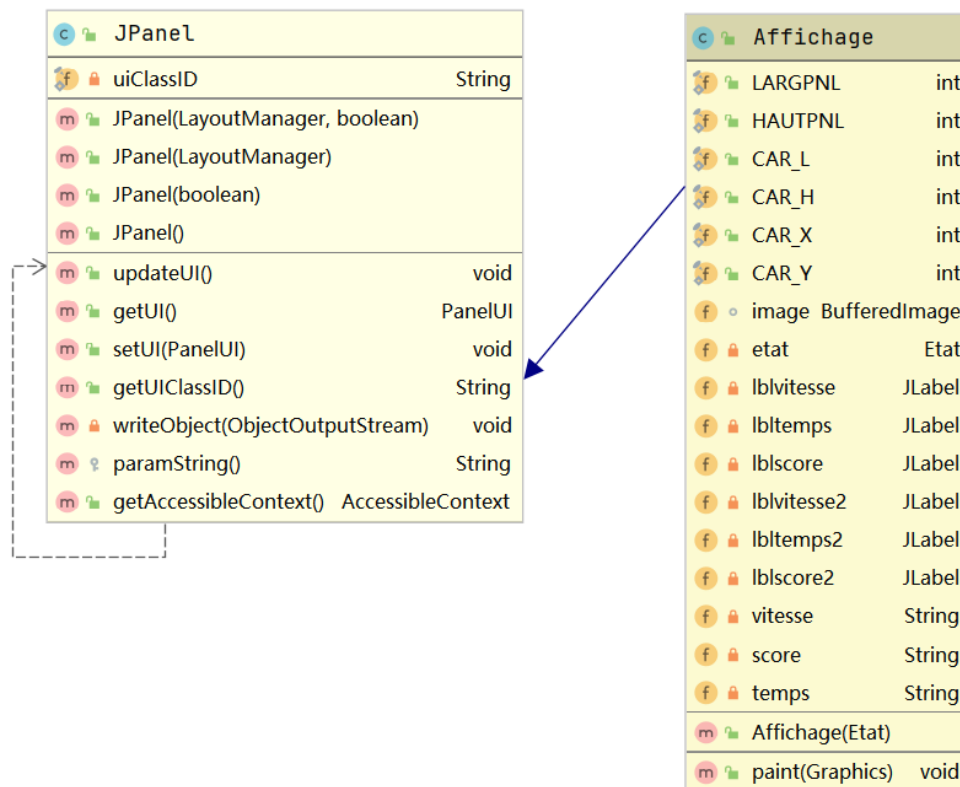
```
public static final int LARGPNL = 800;
public static final int HAUTPNL = 600;
```

- Ajouter une image de voiture dans l'interface

Pour la photo de la voiture, d'abord, nous utilisons "ImageIO.read" pour obtenir la photo et puis utilisons la méthode paint(Graphics g) dans la classe "Affichage" qui est dans le Vue et la fonction drawimage() pour dessiner la photo dans la

fenêtre. Les dimensions de la photo dans des constantes :

```
public static final int CAR_L = 100;
public static final int CAR_H = 100;
public static final int CAR_X = 250;
public static final int CAR_Y = 475;
```



## 2. Déplacement de la voiture vers la droite ou gauche lorsqu'on utilise le clavier

Pour la fonctionnalité de déplacer la voiture en utilisant le clavier. Nous utilisons la programmation événementielle avec la classe `KeyListener` et la distance est définie dans une constante :

```
public static final int move = 30;
```

Nous utilisons la fonction `getKeyCode()` pour déterminer sur quelle touche fléchée l'utilisateur a appuyé :

Gouche : `getKeyCode()=37`

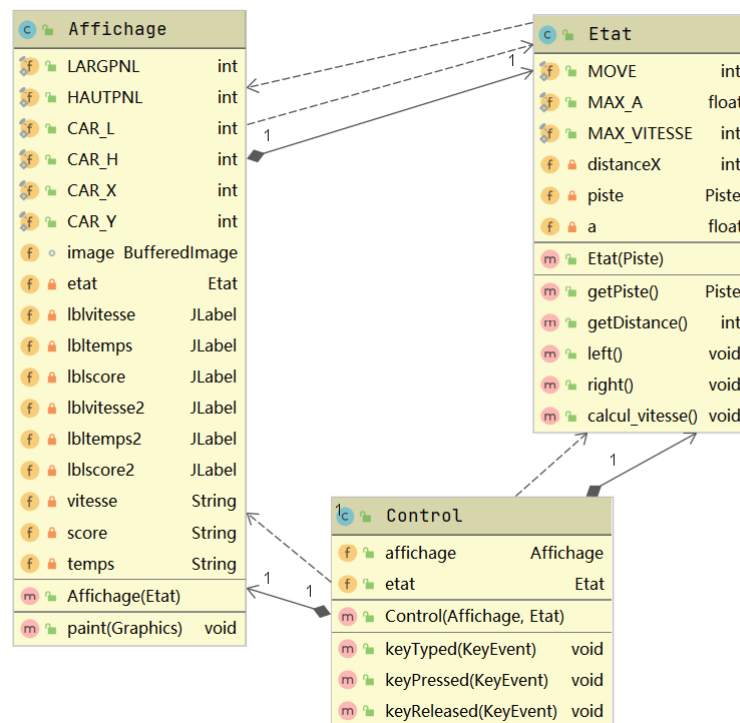
Haut : `getKeyCode()=38`

Droite : `getKeyCode()=39`

Bas : `getKeyCode()=40`

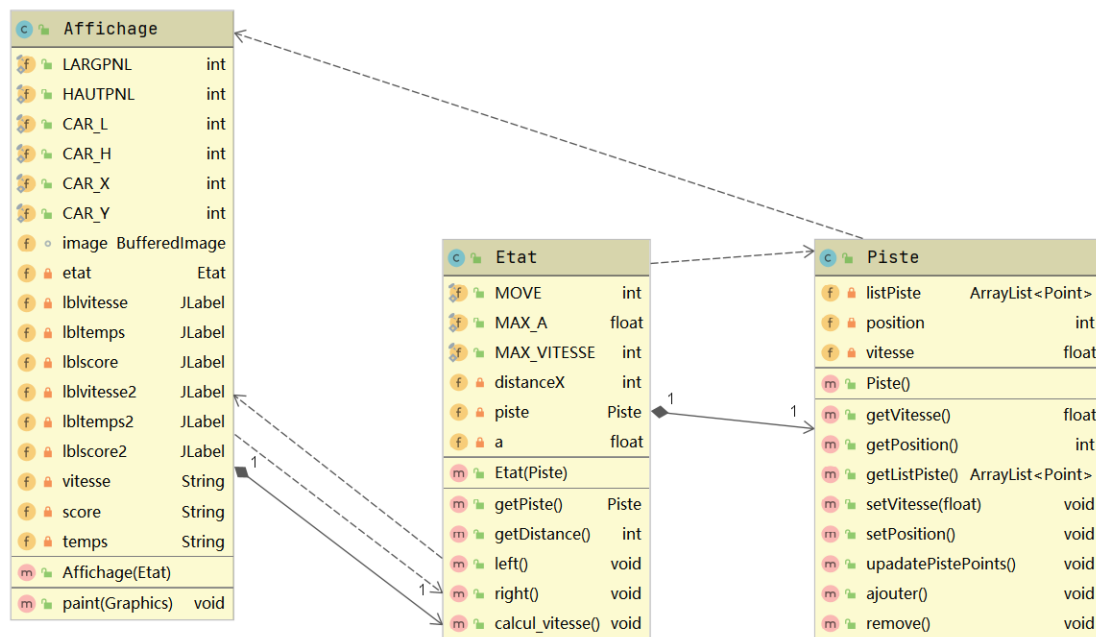
Les méthodes pour déplacer la voiture sont dans le Modèle. Dans les deux méthodes, nous augmentons ou diminuons la coordonnée x de la voiture pour

réaliser le mouvement gauche ou droite de la voiture.



### 3. Génération de la piste avec différentes pentes.

Pour réaliser la fonctionnalité de générer la piste avec différentes pentes, la class "Piste" peut générer les points avec la fonction "random" dans le même intervalle. Et puis, écrire une méthode 'updatePistePoints()' dans la classe Piste pour obtenir les points qui sont utilisés de dessiner la piste. Ces points sont générés aléatoirement, donc les lignes entre les deux points peut construire une piste.



#### 4. La piste est infinie et elle peut avancer automatiquement

Nous voulons la piste peut avancer automatiquement. Nous devons utiliser thread pour le réaliser car nous voulons évitez les conflits avec le mouvement de la voiture. Nous écrivons une nouvelle classe Avancer pour implémenter cette fonctionnalité. Nous aussi changeons la méthode getPiste() pour modifier la coordonnées des points par retirer la valeur de position à leur valeur d'abscisse.

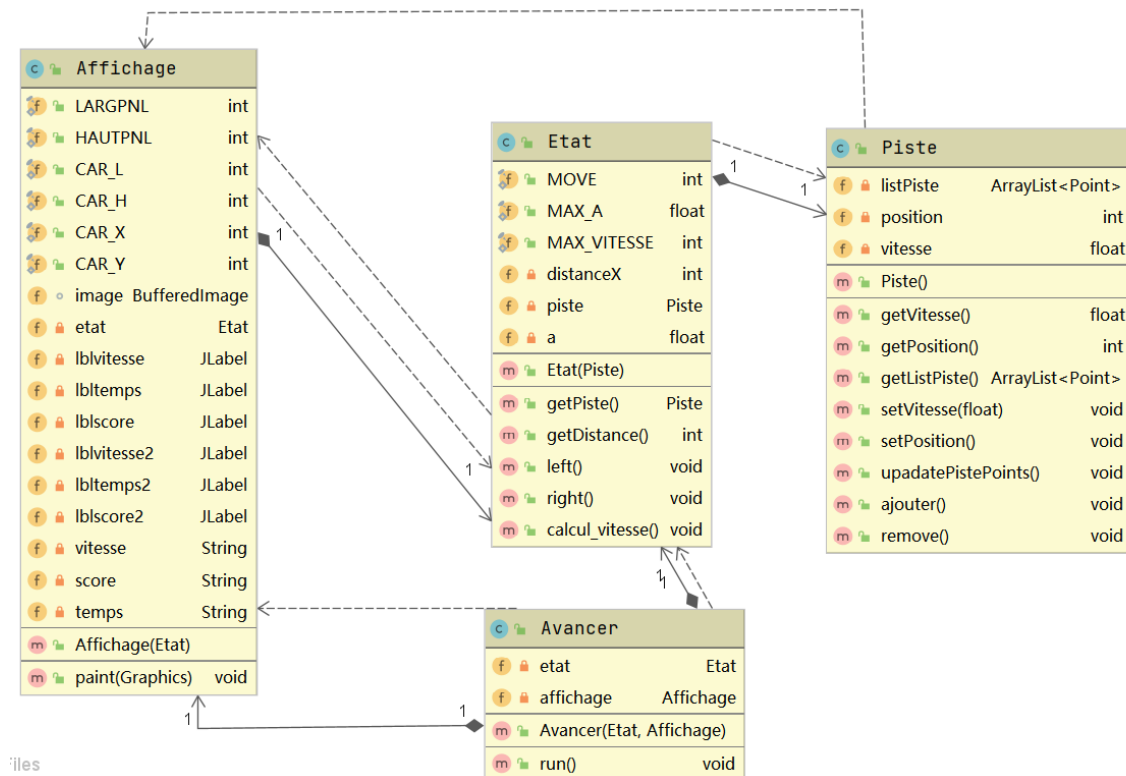
Pour réaliser la piste est infinie, il y a deux conditions nécessaires.

Lorsque le dernier point entre dans la Ligne horizontale, il faut générer un point supplémentaire pour que la piste ne s'interrompe pas.

Lorsque le deuxième point est sorti de l'interface de la fenêtre, il faut retirer le premier point de la liste : il ne sera plus utilisé.

J'ai ajouté deux fonctionnalités dans la classe Piste : ajouter() et remove(). Ces deux méthodes implémentent respectivement l'ajout de nouveaux points et la suppression de points invisibles.

Il y a un problème sur affichage la piste. Car lorsque le dernier point est au-dessus de la ligne horizontale, la ligne générée au-dessus de la ligne horizontale doit être masquée, nous utilisons `clearRect()` pour effacer la ligne qui doit être masquée.



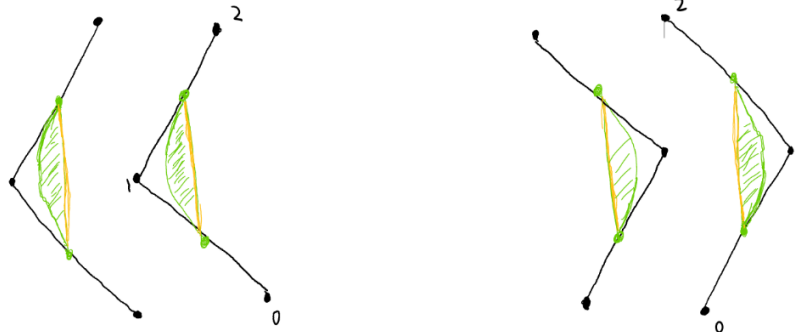
## 5. Gestion la vitesse de la voiture

Nous devons modifier la vitesse de la piste en fonction de la distance entre la piste et la voiture. Nous définissons l'accélération maximale et la vitesse maximale pour nous assurer que le jeu peut fonctionner normalement :

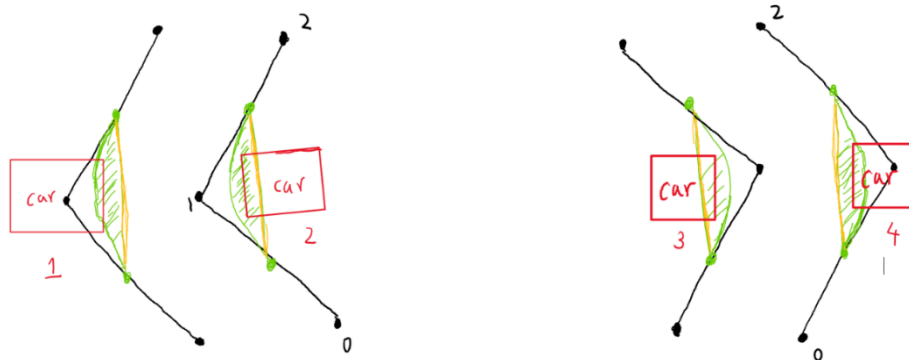
```
public static final float MAX_A = 2;
public static final int MAX_VITESSE = 50;
```

Ensuite, nous avons sélectionné le côté droit de la piste et le côté droit de la voiture comme les systèmes de références, nous avons calculerons la distance entre la voiture et le côté droit de la piste par unité de temps et attribué l'accélération correspondante en fonction de cette distance.

Lors du calcul de la distance, nous devons calculer la pente et l'expression de la ligne droite entre deux points, afin que nous puissions obtenir les coordonnées exactes du point sur le côté droit de la piste qui doit être comparé avec le côté droit de la voiture en ce moment.

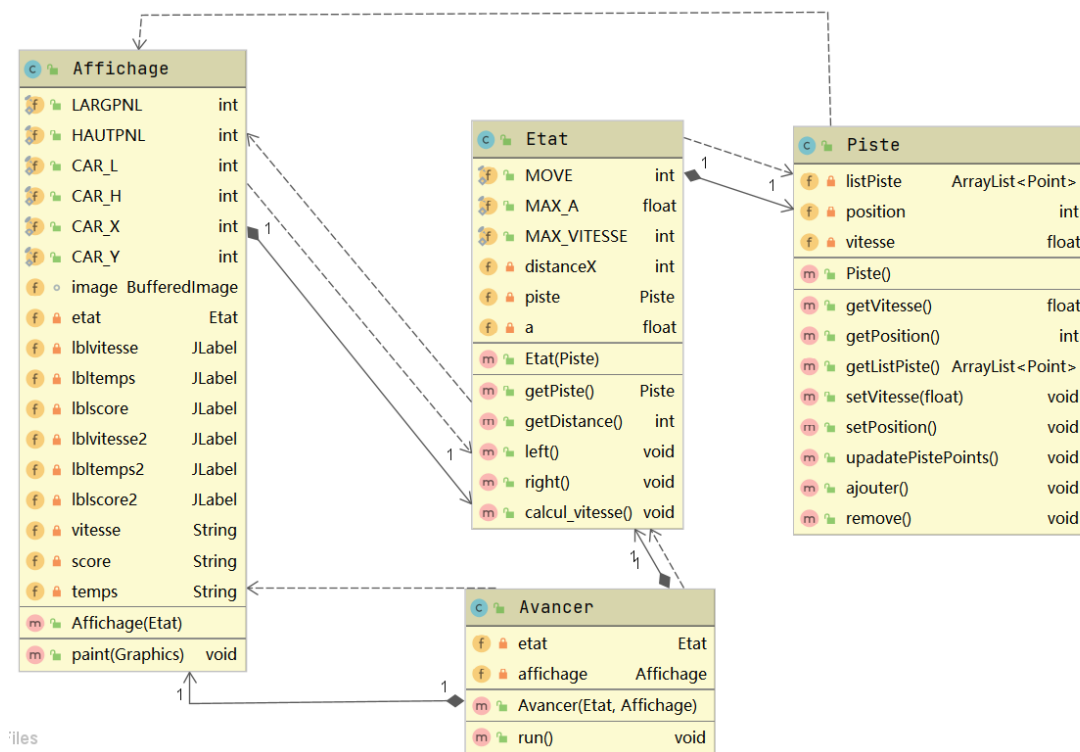


- Réduisez l'erreur entre la ligne brisée et la courbe  
Comme la figure ci-dessus, les points noirs sont les points dans le ArrayList 'ListPiste', les points verts sont les points qui servent à construire la piste dans le modèle afin de calculer l'accélération, et la courbe verte est la piste dans l'affichage. On peut voir qu'il existe une certaine zone d'erreur entre la ligne brisée et la courbe. Dans ces zones vertes, il y a des erreurs dans le système de calcul d'accélération précédent.



Donc, on a ajouté un système de la réduction l'erreur dans la fonction `public void calcul_vitesse()`. Dans les zones critiques, on lui donne une accélération constante. Comme la figure ci-dessus, Le carré rouge représente la voiture. On peut voir qu'il existe quatre cas. Pour le cas 1, on met l'accélération à 1, pour le cas 2, on met l'accélération à 0, pour le cas 3, on met l'accélération à 0 et pour le cas4, on met l'accélération à 1.

Lorsque la voiture est complètement sur la piste, l'accélération est la plus élevée. Si la voiture est complètement hors de la piste, l'accélération est négative, la valeur est -1. La vitesse est lentement réduite jusqu'à 0.



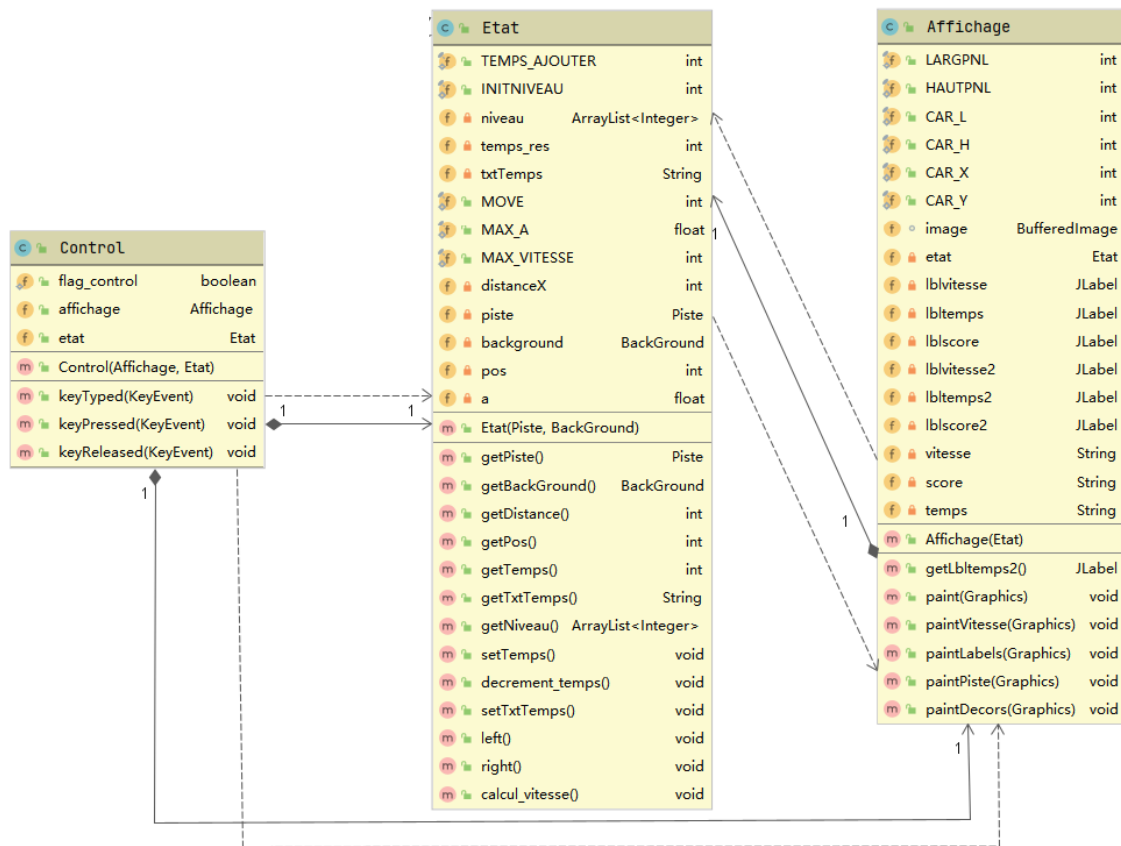
## 6. Gestion du décor de fond

Pour dessiner du décor de fond, nous avons ajouté une image pour le décor. Dans la classe "Affichage", nous avons les ajoutés comme ajouter l'image de voiture.

Pour de l'utilisation du clavier pour les contrôler, nous avons le mis dans la même méthode que le contrôle du mouvement de la voiture. Définissez une nouvelle variable "pos" pour déterminer les coordonnées des décors par rapport à l'interface après chaque mouvement. La distance de chaque mouvement est la même que celle de la voiture et le mouvement invariant est MOVE.

```
public static final int move = 30;
```

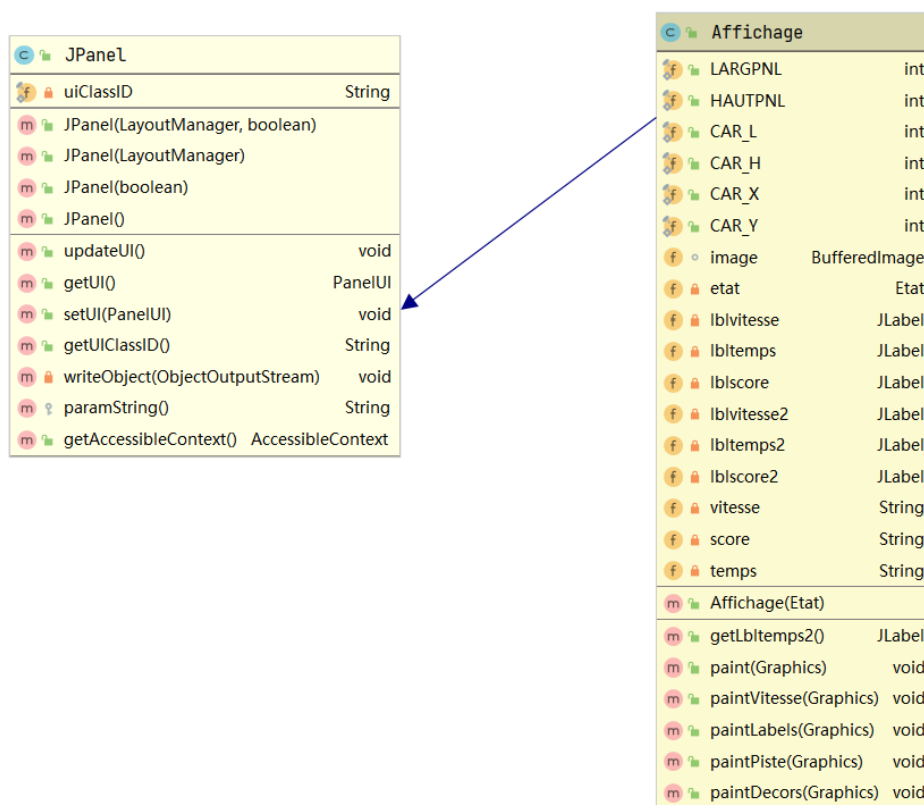
Nous avons défini la plage dans laquelle les décors et la voiture peuvent se déplacer, afin d'empêcher la voiture de sortir de l'interface.



## 7. Transformation la ligne droite de la piste en courbe

Pour transformer la ligne droite de la piste en courbe, nous avons utilisé la courbe de Bézier, nous avons choisi 3 points (courbe quadratique). Il y a trois points au lieu de 2 points, les 3 points sont : le point central du premier point et du deuxième point, le deuxième point et le point central du deuxième point et du troisième point. Donc les méthodes pour ajouter le point et supprimer le point ont été modifiées. Nous avons utilisé `QuadCurve2D` et `Graphics 2D` pour dessiner les courbes.





## 8. Gestion du temps

Pour la gestion du temps, nous avons le séparé à trois parties.

Le temps restant est fixé à 30 secondes au début. Si la position de la voiture (score) atteint à un nouveau niveau, il augmentera de 20 secondes. Si le temps restant diminue à 0, le jeu est terminé, une fenêtre affichant le score final apparaîtra.

- Afficher le temps restant et le décrémenter

Nous avons utilisé 3 int et un nouveau thread 'temps' pour réaliser cette fonctionnalité.

```
int temps_res; // le temps restant
int mm = temps_res / 60 % 60;
int ss = temps_res % 60;
```

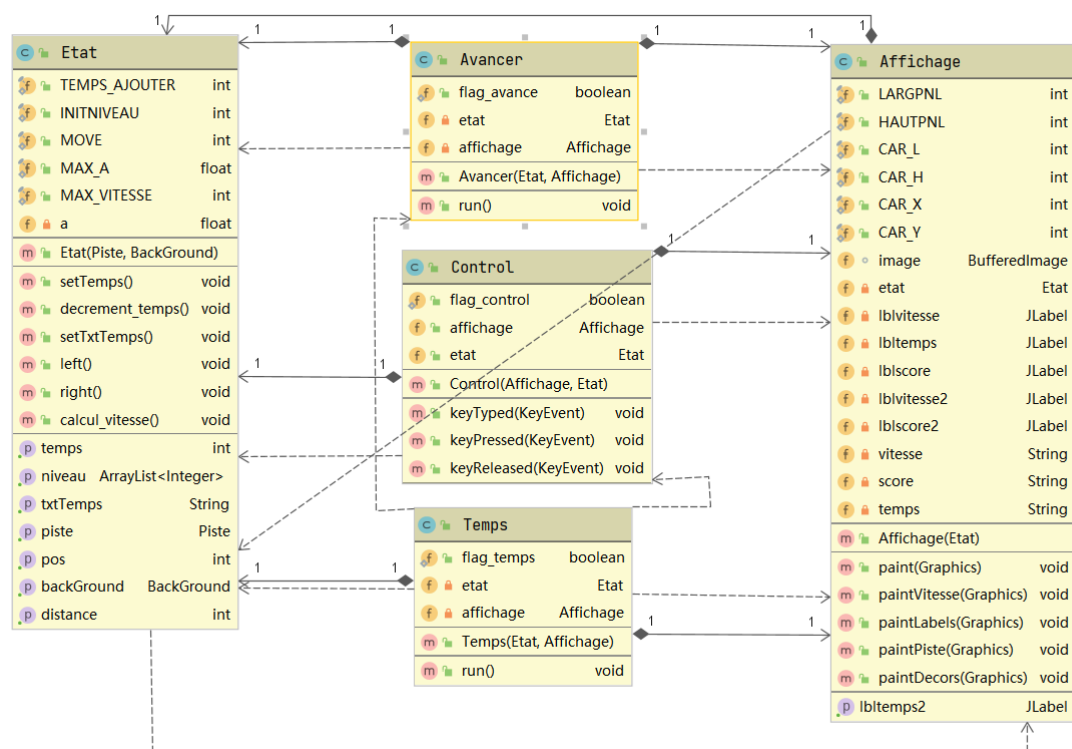
Nous avons défini trois variables, temps\_res, mm et ss. Le temps\_res sert à stocker le temps restant total en secondes, le mm et ss servent à stocker la partie minute et la partie seconde du temps restant. Dans l'affichage, le temps restant sera affiché sous la forme mm : ss.

Afin de réduire le temps restant d'une par seconde, dans la fonction run() de la Class Temps, dans chaque itération, temps\_res est décrémenté, et puis, le mm,

ss seront recalculer, et le thread fait une pause de 1 seconde entre chaque itération.

```
public static final int TEMPS_AJOUTER=20;
public static final int INITNIVEAU=4000;
```

- Lorsque le score actuel atteinre un nouveau niveau, le temps augmente  
Nous avons mis en place un total de dix niveaux et construit une arraylist pour stocker les scores nécessaires pour passer chaque niveau. Le premier niveau est défini dans la constante `INITNIVEAU`, Chaque niveau arrière est supérieur à l'avant. Dans la classe avancer, une fois que le score atteinre un nouveau niveau, le temps restant sera ajouté à la valeur constante `TEMPS_AJOUTER`
- Le temps restant diminue à 0, le jeu est terminé.  
Nous avons défini trois flags : `flagavancer`, `flagcontrol` et `flagtemps`. Ces trois drapeaux correspondent au mouvement de la piste, au contrôle du clavier et au décrémentation du temps. Dans le thread "Temps", nous avons vérifié si nous avons obtenu le score correspondant avant le temps restant diminue à 0. Si le temps restant diminue à 0, tous les flags deviennent faux, donc il peut terminer le jeu. Lorsque le jeu est terminé, une boîte de dialogue apparaîtra pour vous indiquer le score final.

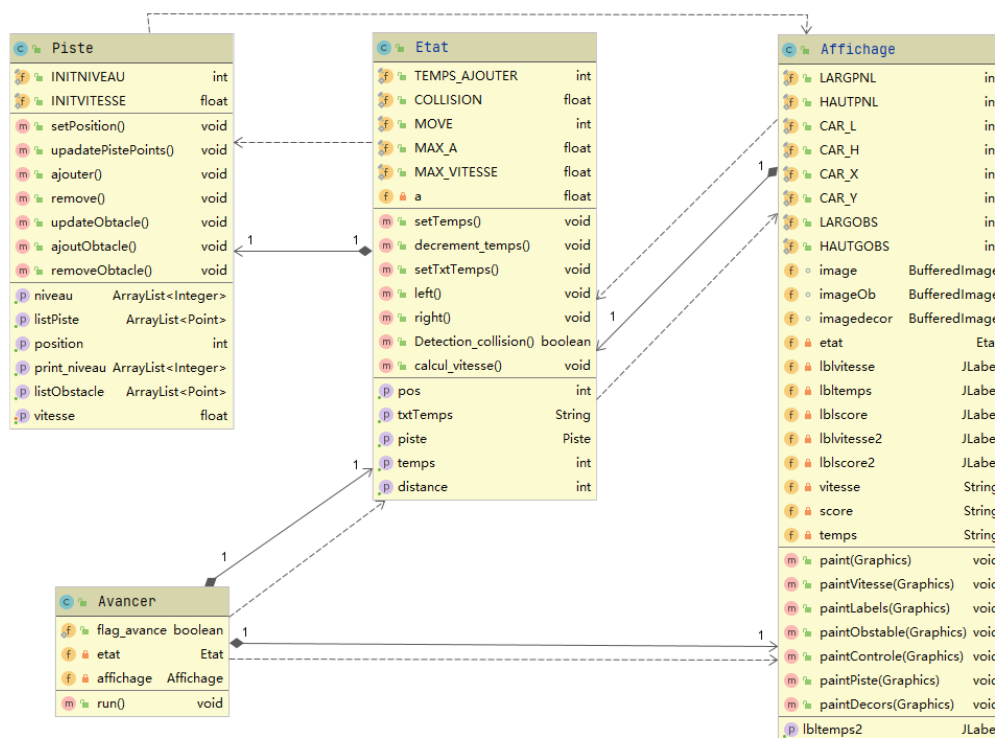


## 9. Ajout l'écran d'accueil

Pour ajouter un autre écran, nous avons créé une nouvelle fenêtre dans la classe Main. Nous avons ajouté le bouton et le titre de ce jeu, masqué la fenêtre d'affichage l'interface du jeu. Pour afficher l'interface du jeu, nous avons utilisé "addActionListener" pour contrôler le bouton. Si nous cliquons le bouton, la fenêtre d'accueil est masquée et l'interface du jeu est affichée.

## 10. Ajout les obstacles

Pour ajouter les obstacles, nous avons utilisé la même façon comme générer la piste. D'abord, nous avons créé une liste pour stocker les points pour générer les obstacles. Le mécanisme d'ajout et de suppression de points est également le même que le mécanisme de la piste. Lorsque le troisième point sorti de l'interface de la fenêtre, il faut générer un point supplémentaire pour générer un nouvel obstacle. Lorsque le premier point est sorti de l'interface de la fenêtre, il faut retirer le premier point de la liste. De cette manière, à mesure que la vitesse de la voiture augmente, la fréquence des obstacles augmente également, ce qui augmente également la difficulté du jeu.



## 11. Détection de la collision

La fonctionnalité « Détection de la collision » est réalisée par la fonction `Detection_collision()` dans la classe `Etat`. C'est une fonction de type boolean, elle renvoie `true` s'il y a un obstacle dans la liste qui entre en collision avec la voiture, sinon elle renvoie `false`. Dans cette fonction, on utilise 5 constantes :

```
/**La largeur de la voiture et La largeur de la piste*/
public static final int CAR_L = 100;
/** La hauteur de la voiture */
public static final int CAR_H = 100;
/** L'ordonne de la voiture */
public static final int CAR_Y = 500;
// La largeur de l'Obstacle
public static final int LARGOBS = 100 ;
// La hauteur de l'Obstacle
public static final int HAUTGOBS = 80 ;
```

Et nous réalisons la vérification de la collision par la façon suivante : Nous parcourons la liste d'obstacle (Représenté dans le modèle par une liste de points, chaque point représente les coordonnées du coin supérieur gauche de l'obstacle).

Pour chaque point dans la liste, nous vérifions d'abord si son ordonnée est comprise entre  $Et$  nous réalisons la vérification de la collision par la façon suivante : Nous parcourons la liste d'obstacle (Représenté dans le modèle par une liste de points, chaque point représente les coordonnées du coin supérieur gauche de l'obstacle).

Pour chaque point dans la liste, nous vérifions d'abord si son ordonnée est comprise entre  $CAR\_Y - HAUTGOBS$  et  $CAR\_Y + CAR\_H$ . Si Oui, cela signifie que l'obstacle représenté par ce point peut entrer en collision avec la voiture. On continue de vérifier si son abscisse est entre  $distanceX - LARGOBS$  et  $distanceX + CAR\_L$  ( $distanceX$  est l'abscisse de la voiture), si c'est vrai, cela veut dire qu'il y a une collision, la fonction renvoie `true`, sinon elle renvoie `false`.

## 12. Affichage des points de contrôles

Dans cette fonction, on utilise les constantes :

```
/** L'ordonne de la voiture */    public static final int CAR_Y = 500;
```

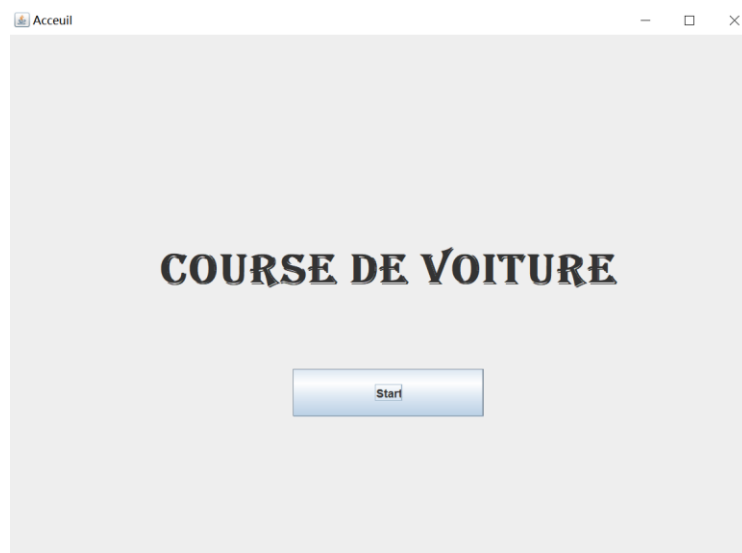
L'affichage des points de contrôles est matérialisé par une bande horizontale sur

la piste. Les points de contrôles sont stockés dans `ArrayList<Integer> niveau`, et afin de les afficher dans la vue du jeu, on génère un autre `ArrayList<Integer> print_niveau`. `print_niveau` est généré à partir de 'niveau'.

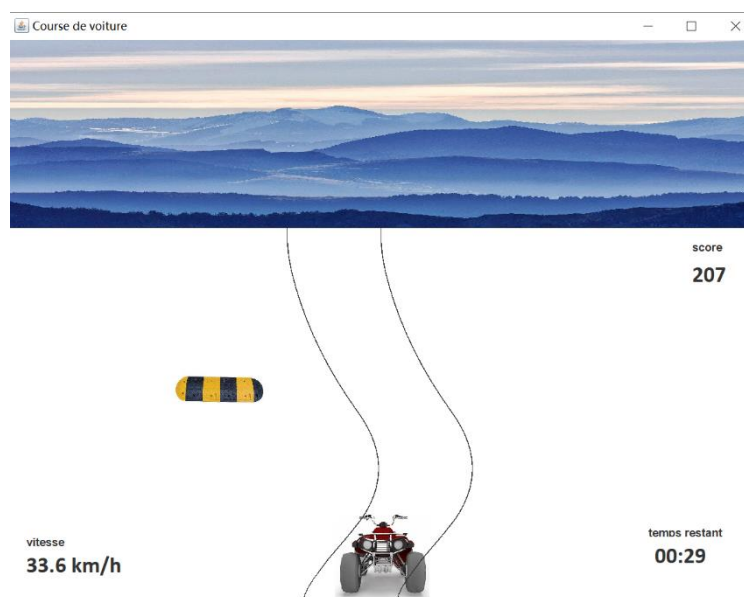
Le principe de la génération est le suivant : Pour chaque entier  $i$  dans la liste 'niveau', on crée un nouvel entier  $y = \text{CAR\_Y} - i$  et l'ajoute dans la liste `print_niveau`. La règle de mise à jour de la liste `print_niveau` est la même que celle de la piste : Pour chaque entier dans la liste, modifier sa valeur par retirer la valeur de position. Elle est également implémentée dans la fonction `updatePistePoints()` de la classe `Piste`.

## Résultat

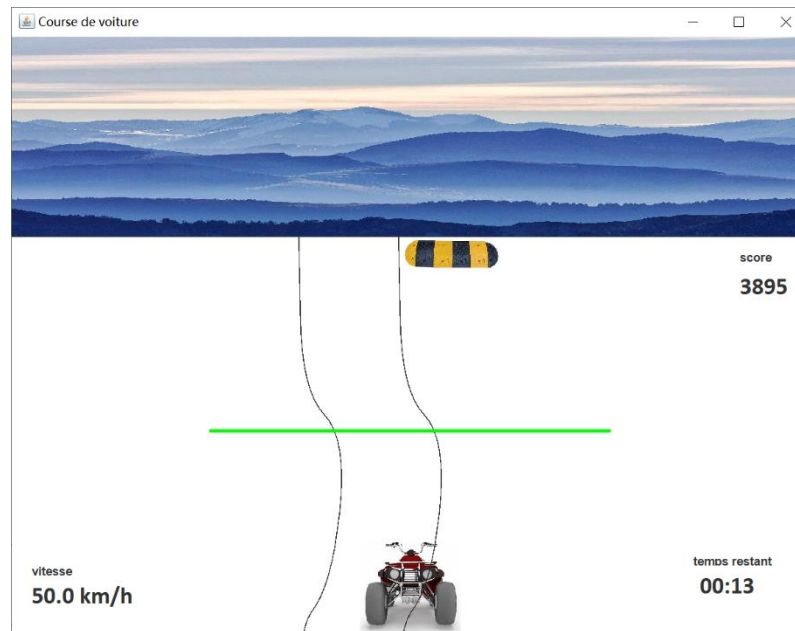
1. Cliquez le bouton pour commencer le jeu.



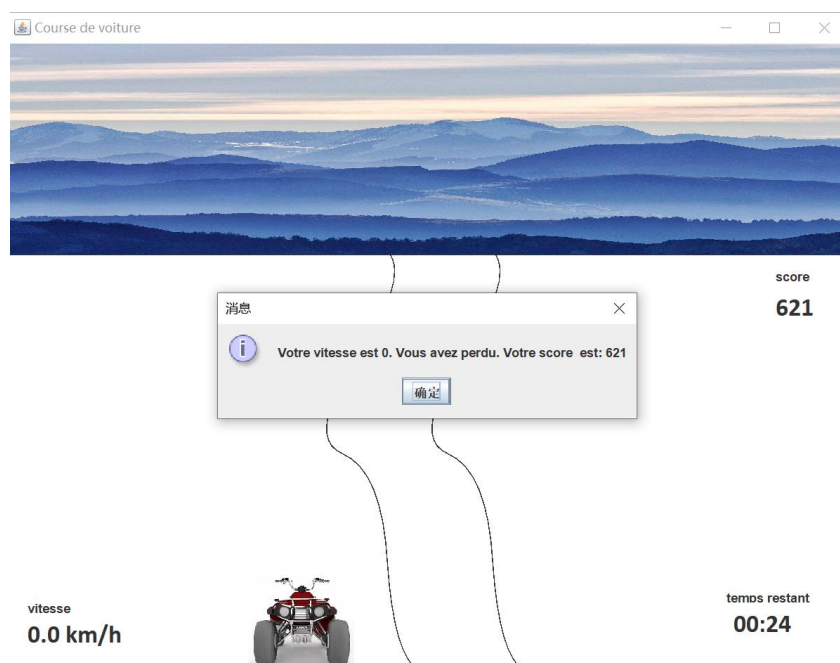
2. Utilisez le clavier pour contrôler la voiture, il y a des obstacles que vous devez les éviter.



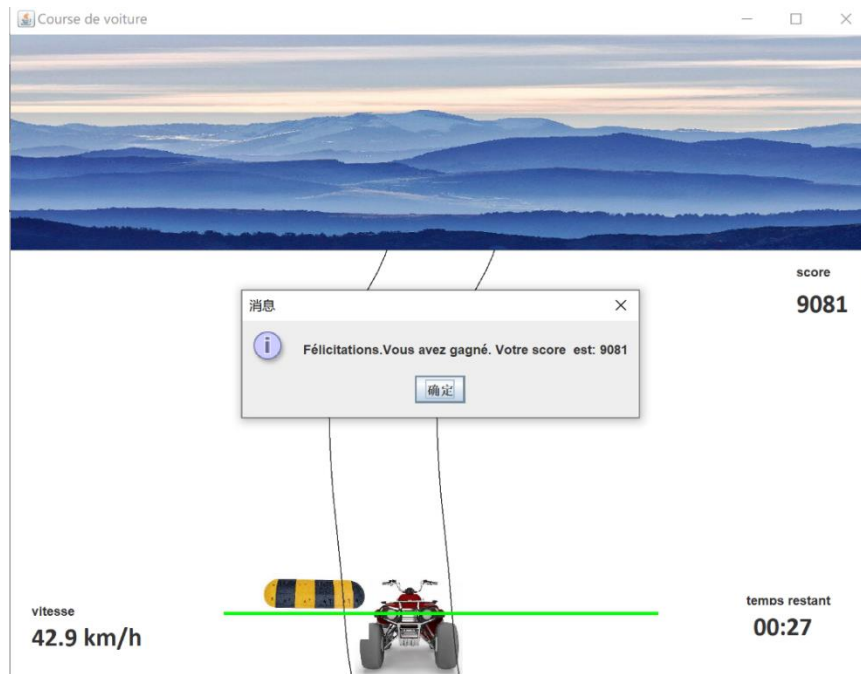
3. Vous pouvez regarder une ligne verte qui indique que vous entrez le prochain niveau.



4. Si la vitesse de la voiture est 0 ou le temps est 0, il affiche un message pour indiquer l'utilisateur le jeu est terminé et le score que l'utilisateur a obtenu.



5. Si vous réussissez tous les niveaux, il affiche un message comme ça.



## Documentation utilisateur

Prérequis : Java avec un IDE, la version  $\geq 1.8$  (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 ». Cliquez le bouton "Start" pour commencer le jeu. Utilisez les touches fléchées du clavier pour contrôler la voiture pour se déplacer à gauche et à droite. La voiture peut obtenir l'accélération maximale sur la piste, et elle sera décélérée lorsqu'elle heurtera des obstacles.

Mode d'emploi (cas .jar exécutable) : double-cliquez sur l'icône du fichier .jar. Cliquez le bouton "Start" pour commencer le jeu. Utilisez les touches fléchées du clavier pour contrôler la voiture pour se déplacer à gauche et à droite. La voiture peut obtenir l'accélération maximale sur la piste, et elle sera décélérée lorsqu'elle heurtera des obstacles.

## Documentation développeur

Cette version simple du jeu est développée en utilisant le mode MVC. Regardez d'abord les trois classes Control, Etat et Affichage pour comprendre brièvement la structure de l'ensemble du ce projet. La méthode main () se trouve dans la classe Main. Toutes les instances d'autre classes sont créé dans la classe Main. Les principales méthodes d'implémentation des fonctionnalités sont dans Etat, Piste, Avancer et Temps, Les deux dernières classes héritent de la classe thread, elles servent à réaliser le mouvement de la piste et la modification du temps restant. Les méthodes pour afficher les choses dans l'interface sont dans Affichage.

Les constances que vous pouvez modifier sont suivantes :

1. La taille de la fenêtre :

```
public static final int LARGPNL = 800;  
public static final int HAUTPNL = 600;
```

2. La taille de la voiture :

```
public static final int CAR_L = 100;  
public static final int CAR_H = 100;
```

3. Les coordonnées de l'ovale dans l'interface :

```
public static final int CAR_X = 350;  
public static final int CAR_Y = 500;
```

4. Le mouvement de la voiture à chaque fois que vous utilisez le clavier :

```
public static final int MOVE = 30;
```

5. L'accélération maximale

```
public static final float MAX_A = 2;
```

6. La Vitesse maximale

```
public static final float MAX_VITESSE = 50;
```

7. Le temps augmente lorsque le joueur atteint un nouveau niveau

```
public static final int TEMPS_AJOUTER=20;
```

8. La vitesse à diminuer lorsqu'il a une collision

```
public static final float COLLISION=4;
```

9. La vitesse initiale

```
public static final float INITVITESSE=20;
```



## 10. La taille des obstacles

```
public static final int LARGOBS = 100;  
public static final int HAUTGOBS = 80;
```

## 11. Le nombre de points de contrôles

```
public static final int NB_NIVEAU=15;
```

Pour l'instant, les principales fonctionnalités sont terminées, mais il y a encore les fonctionnalités complémentaires pour réaliser. Par exemple, ajouter les concurrences, Le dessin de la piste se rétrécit au bout pour créer une sensation de profondeur, le dessin du véhicule change en fonction des actions du joueur pour suggérer les mouvements à droite et à gauche etc. Ces fonctionnalités complémentaires peuvent rendre le jeu plus visuel.

## Conclusion et perspectives

Finalement, nous avons réussi à faire fonctionner ce jeu. Nous avons implémenté toutes les fonctions principales et certaines de ces fonctionnalités complémentaires. Les fonctionnalités complémentaires que nous avons réalisées sont : Le défilement du décor à droite et à gauche en fonction des mouvements du véhicule. L'apparition aléatoire d'obstacles sur la piste. L'ajout d'un écran d'accueil.

Au cours du développement de ce projet, nous avons non seulement révisé les connaissances que nous avons acquises sur swing, l'utilisation de multithread mais nous avons également appris l'utilisation de Class QuadCurve2D, l'interface KeyListener et ActionListener.

Pendant le développement, nous avons rencontré plusieurs difficultés. L'un des problèmes les plus difficiles est : lorsque nous jouons au jeu, la vitesse devient soudainement 0 de temps en temps et la voiture ne peut plus être accélérée. Afin de résoudre ce problème, nous avons effectué de nombreux tests. Tout d'abord, nous avons suspecté le problème du type de données, car notre type de données n'est pas unifié, la vitesse est de type float, mais l'accélération définie est de type int, nous ne l'avons pas résolu ce problème après avoir unifié tous les types de données. Enfin, nous avons découvert que le problème venait du calcul de la distance entre la voiture et la piste. Nous avons utilisé cette expression :  $\text{pente} = (p2y - p1y) / (p2x - p1x)$  au cours du calcul,  $p1$  et  $p2$  sont les points pour générer la piste. Mais il y a une situation que  $p1$  et  $p2$  ont les mêmes des coordonnées sur l'axe des x, donc  $p2x - p1x = 0$ , cela cause le problème. Pour résoudre ce problème, nous avons utilisé deux méthodes pour calculer la distance, nous avons extrait cette situation et calculer la distance sans utilisant la pente.

Pour finir, nous voudrions dire que nous sommes contents de pouvoir faire ce projet, il nous a permis d'accumuler une expérience précieuse dans l'analyse de projets complexes et peut nous aider dans le futur.