# 1.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. Notre version du jeu sera implémentée en Java, le joueur utilisera le clavier pour faire déplacer le véhicule qui accélère sur la piste pour éviter des obstacles et réussir à franchir le « checkpoint » avant la fin d'un compteur.

# 2.Analyse globale

Notre projet sera implémenté suivant le modèle MVC (model/vue/control), et contiendra une vue dans laquelle sont dessinés les éléments suivants :

— Un véhicule, représenté par une image, il reste toujours sur place alors que les autres éléments de la vue bougeront pour simuler l'avancement sur la route. Le véhicule bougera à l'aide du clavier et pourra se déplacer à droite et à gauche de la piste avec des lois physiques qui s'appliqueront lorsque notre véhicule sort de la piste ou heurte un autre objet ce qui affectera sa vitesse et son accélération.

— Une piste infinie qui est généré automatiquement, aléatoirement et qui défile lorsque le véhicule avance. La piste contiendra aussi des virages et la vue les prendra en compte pour être sûr de garder le véhicule au centre de la fenêtre et que le reste du décor bouge pour donner l'impression d'avancer tout au long de la piste.

— Un horizon et un décor qui donne un sens de perspective et d'un monde 3D. Le décor bougera selon la position du joueur sur la piste et augmentera de taille lorsqu'elle s'approche du véhicule

— Un menu pour contrôler les paramètres et qui permet de lancer et quitter le jeu.

## 2.1-Le véhicule

Le véhicule et son mouvement sont primordiaux pour notre projet, le véhicule sera représenté par une image de vaisseau spatial inspiré du film Star Wars. Lorsque le joueur tourne à gauche ou à droite les images seront mises à jour pour donner l'impression de changement de position sur la piste. Le véhicule pourra aussi se déplacer verticalement, il quittera la piste et perd ainsi de la vitesse ceci sera visible avec un changement de l'image qui sera coloré lorsque le véhicule est sur la piste. La vitesse sera calculée automatiquement et diminuera si le véhicule sort de la piste ou bien heurte d'autre objet sur son chemin.

## 2.2-La piste

La piste sera représentée par des sections et contiendra des checkpoints qui attribue des points bonus si le joueur les franchi. La piste défilera automatiquement si la vitesse du joueur est différente de zéro, et contiendra des virages à gauche et à droite ainsi que des parties droites qui seront attribué aux sections. La piste contiendra aussi des objets immobiles qui au contact avec le joueur le pénalise en le freinant.

## 2.3-L'Horizon

L'horizon constituera la moitié supérieure de notre affichage, trois éléments seront dessinés dessus :

- L'arrière-plan : Une image d'une nuit étoilée pour simuler un monde dans l'espace.

- Des étoiles : Ces étoiles partent du centre de l'horizon et en s'approchant du joueur leur taille augmentera. Ceci créera l'illusion de traverser un champ d'étoiles.
- Des planètes : Les planètes se déplaceront en suivant les virages de la piste est donc logiquement le déplacement du joueur.

### 2.4-Le menu

Le joueur aura accès au menu au début du jeu et aura le choix entre jouer et quitter. Le menu sera aussi accessible en utilisant la touche p durant la partie. La navigation dans le menu sera gérer avec la souris et le joueur cliquera sur les boutons.

## 3. **Plan de développement**

### 3.1-Le véhicule

- Analyse du problème : temps de travail estimé 2h
- Conception, développement et test du mouvement sur la piste : temps de travail estimé 3h

Pour le mouvement sur la piste il sera nécessaire d'implémenter une solution qui permet de gérer les touches de clavier en continu pour éviter des mouvements saccadés et avoir un bon control sur le véhicule. Pour cela on aurait besoin de l'interface KeyListener de Java Swing.

- Conception, développement et test du mécanisme de vitesse et d'accélération : temps de travail estimé 2h

La gestion de la vitesse du véhicule n'est pas une tâche difficile à réaliser, mais plusieurs tests sont nécessaires pour affiner le résultat et avoir un mouvement fluide sur la piste.

- Recherche et modification d'images : temps de travail estimé 2h -3h

La recherche d'image open Source peut s'avérer un peu difficile puisque c'est rare de trouver de trouver des images liées à notre sujet.

- Documentation du code : temps de travail estimé 30min.

### 3.2-La piste

La piste est sans doute la pièce maitresse dans notre jeu et est l'élément le plus difficile à implémenter et tester.

- Analyse du problème : temps de travail estimé 2jours
- Conception, développement et test : 7 jours
- Documentation du code : temps de travail estimé 1h

La recherche d'algorithme pour afficher la piste et donner un sens de perspective au joueur est une tache difficile pour cela il faudra rechercher la projection en pseudo3d et étudier la géométrie nécessaire pour implémenter ce problème. Pour la suite il faudra rechercher les algorithmes pour la création des virages.

### 3.3-L'Horizon

- Analyse du problème : temps de travail estimé 1h
- Conception, développement et test :1 jour
- Recherche et modification d'images : temps de travail estimé 2h -3h
- Documentation 30min

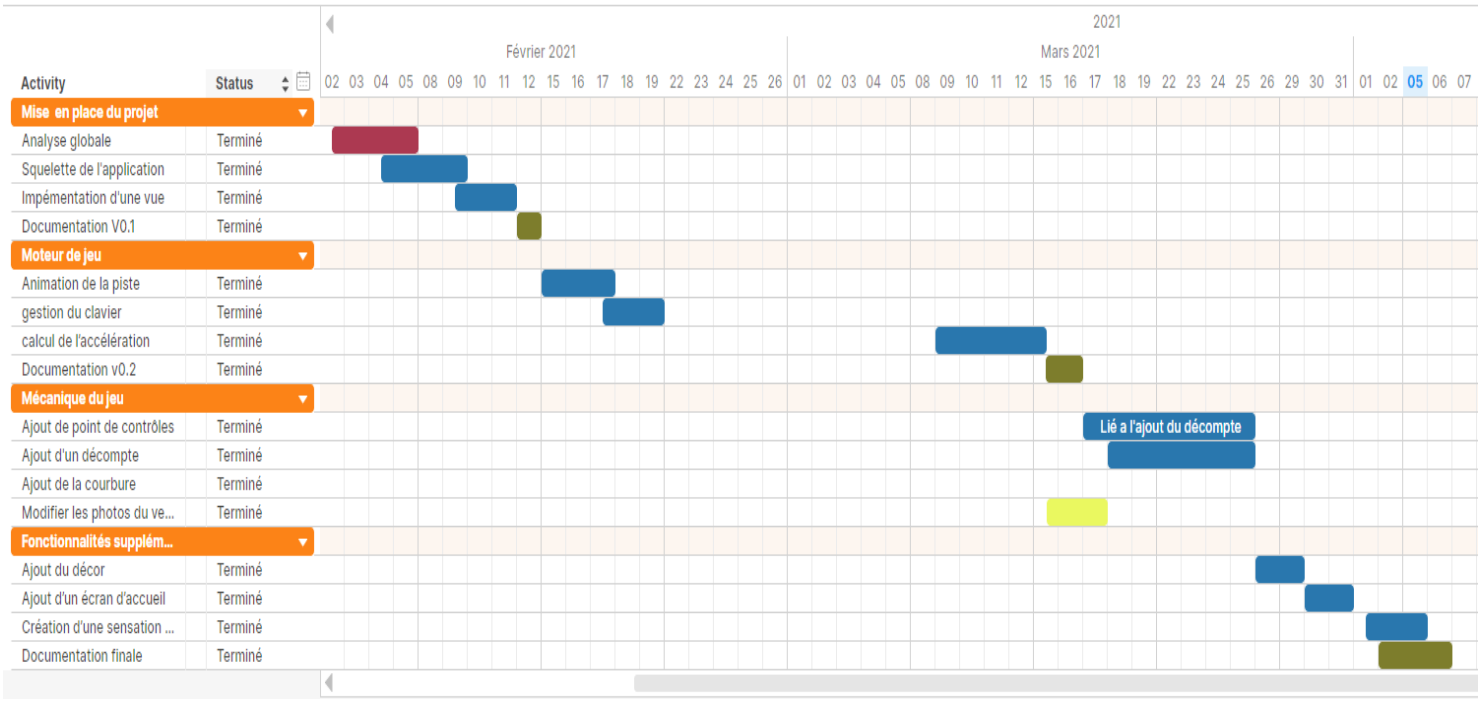
Cette tache est relativement facile en comparaison avec la précédente cependant on aura besoin d'une recherche approfondie d'images qui conviennent à nos besoins. Le champ d'étoiles ne sera pas très compliqué à implémenter néanmoins cette fonctionnalité nécessite beaucoup de test pour ne pas avoir un décor qui perturbe le joueur.

### 3.4-Le menu

- Analyse du problème : temps de travail estimé 1h
- Conception, développement et test :2h
- Documentation 30min

Le menu est la fonctionnalité la plus simple à implémenter, on aura besoin de l'interface MouseListener.

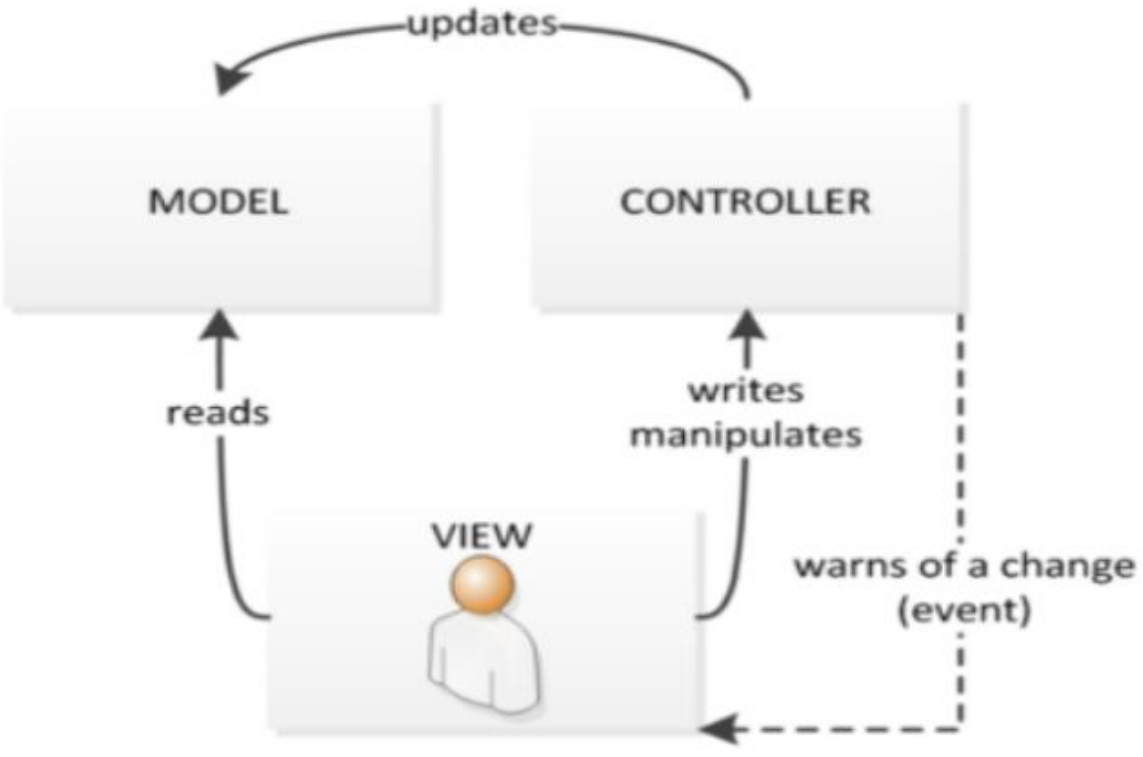
## Diagramme de Gannt




## 4. Conception général

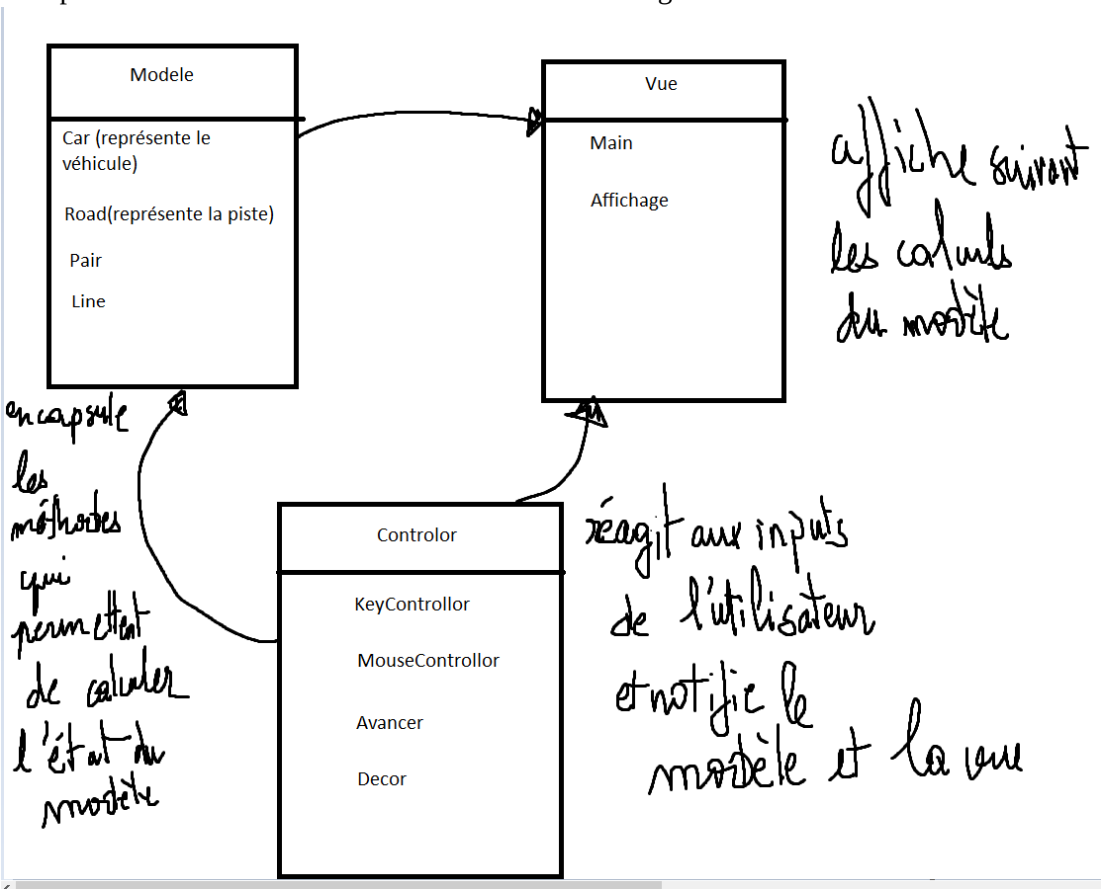
Puisque notre programme gère une interface graphique, nous avons adopté le design pattern MVC. Notre fenêtre sera gérée par la vue qui s'occupera de l'affichage des éléments à dessiner sur l'écran le véhicule compris. Tandis que le mouvement de la voiture suite à des clics sur le clavier sera gérer par le control qui implémente un listener et notifie la vue des changements qui ont eu lieu tout en modifiant les variables du modèle. Voici un diagramme qui illustre le modèle MVC :

Rapport PCII
Ben Naceur Fedy




Le motif MVC (*source : Wikimedia*)

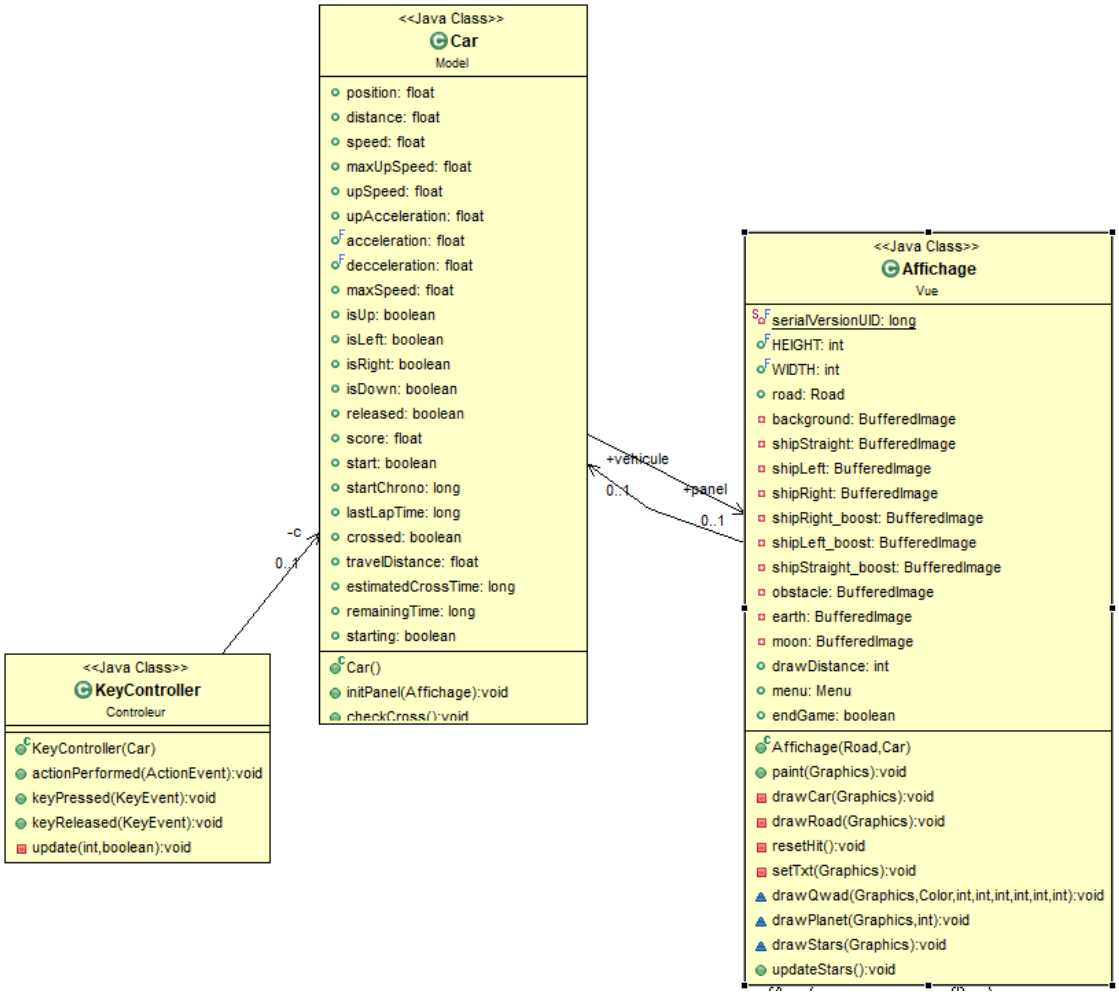
Plus précisément notre architecture ressemblera à ce diagramme :



# 5. Conception détaillée

## 5.1-Le véhicule

La classe Car du package Model ainsi que KeyController du package coderont le comportement du véhicule. Voici un diagramme illustrant la classe Car et celles qui interagissent avec :



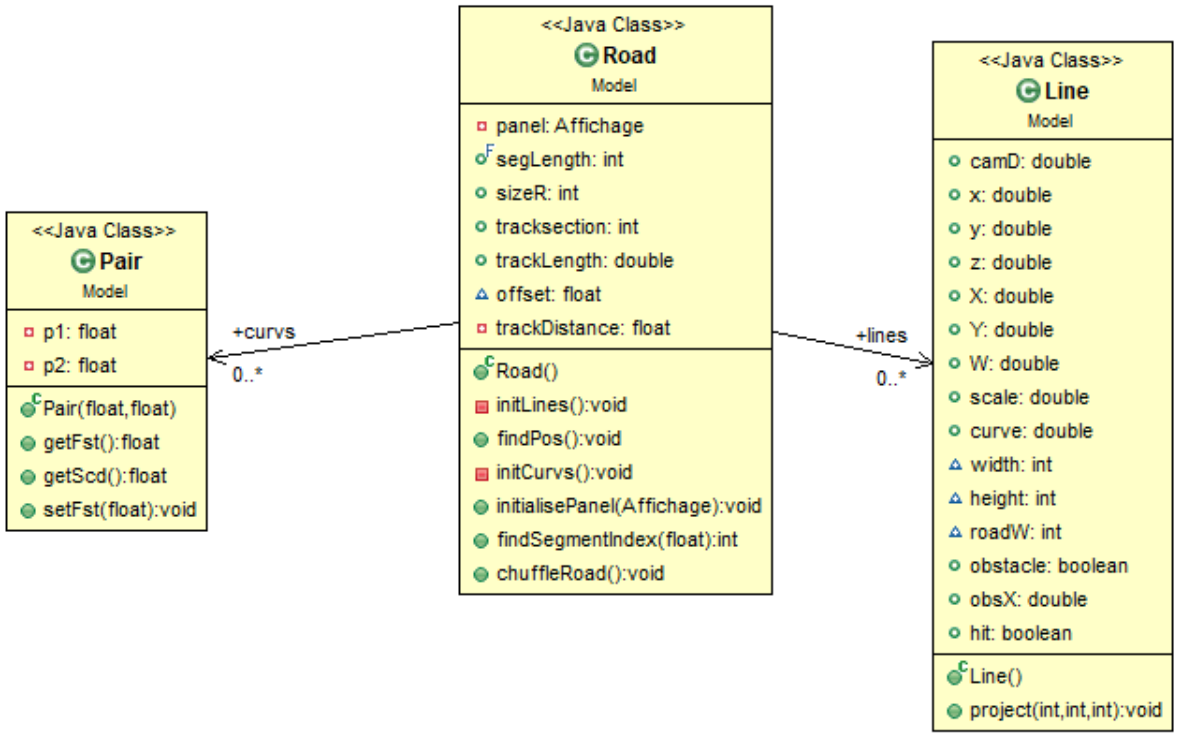

Les attributs importants de la classe Car :

- position : position du véhicule sur la piste
- distance : la distance parcourue par rapport au circuit
- speed : la vitesse du véhicule
- maxUpSpeed : la vitesse maximale en hauteur
- upSpeed : la vitesse en hauteur
- upAcceleration :l'accélération en hauteur
- acceleration :accélération sur la piste
- maxSpeed :la vitesse maximale du véhicule
- score : le score du joueur
- lastLapTime : le temps du dernier tour
- estimatedCrossTime : le temps estimé pour chaque tour
- remainingTime : le temps qui reste pour effectuer le tour s'il vaut 0 la partie est perdue

La classe KeyController implémente le comportement du véhicule en fonction des « inputs » de l'utilisateur sa méthode actionPerformed permet de mettre à jour les attributs de la classe Car en fonction du déplacement sur la piste elle permet aussi la gestion en continu du clavier.

## 5.2-La piste

La classe Road, Pair et Line contiennent la logique derrière la route, pour bien expliquer comment les structures interagissent entre eux on commence d'abord par donner un diagramme qui regroupe les trois classes concernées :




Avant d'expliquer le rôle des attributs une explication du choix de cette structure nous parait nécessaire, en effet suite au dernier Tuto qu'on a effectué l'implémentation de la ligne brisée semblait donner moins de flexibilité et de liberté qu'on voulait pour un jeu de course. De plus plusieurs taches semblent être considérablement plus difficile à réaliser en utilisant des lignes brisées notamment les virages et le sens de perspective. Après de nombreux tests on a choisi d'utiliser l'implémentation suivante :

Notre route sera divisée en section avec deux caractéristiques : la longueur et le degré de courbure (0 sera une ligne droite ,4 un virage droit et -4 un virage gauche) ces sections seront stockées dans l'attribut curvs en utilisant la classe Pair qui est une petite version de celle trouvée dans JavaFx comme on n'avait pas accès, on l'a implémenté. Pour générer une route aléatoire on utilise la fonction shuffleRoad qui attribut à chaque section une courbure aléatoire après chaque tour.

Ci-dessous les attributs de la classe Road :

- lines : cet attribut représente l'ensemble des lignes qui constituent l'intégralité de la piste et qui seront nécessaire pour l'affichage avec la méthode project de la classe Line. Cette collection sera initialisée au moment de la création de la route et ne changera pas. Chaque ligne contiendra un attribut curve qui représentera la courbure attribuée à cette partie de la route.
- curvs : Curvs représentera chaque section de la route et l'idée derrière le fonctionnement a été expliqué ci-dessus.

Voici un extrait de code qui permet d'attribuer de manière aléatoire les courbures dans la route :

Procédure chuffleRoad

Pour chaque paire dans curv :

On choisit de manière aléatoire un niveau de courbure

j = 0;

c = curvs.get(j).getFst() //représente la courbure pour une section

d = curvs.get(j).getScd()//représente la longueur de la section

x = 0 //représente la distance traversé dans la route

Pour chaque ligne dans lines :

Si on se trouve dans la bonne section {

j++;

d = curvs.get(j).getScd();

c = curvs.get(j).getFst();

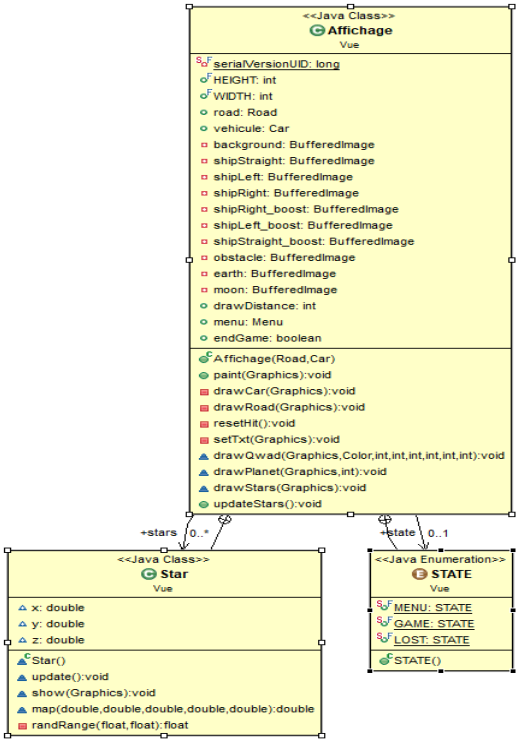
x = i * segLength;

/*On attribue les courbures aux lignes

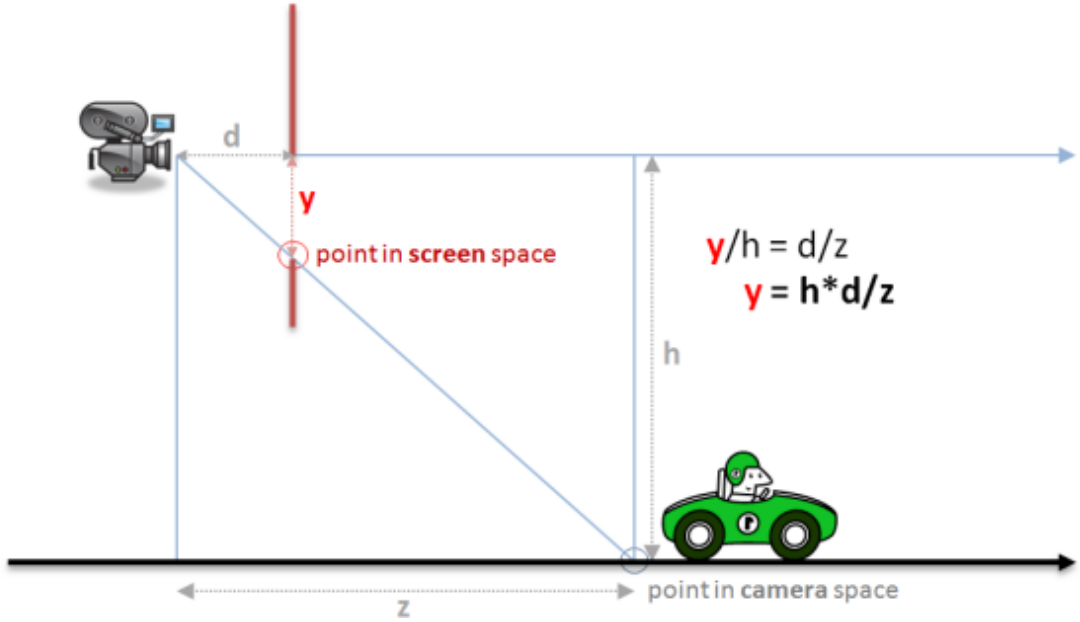
}

## 5.3-L'Affichage

Cette partie était sans doute la plus difficile à réaliser, la classe Affichage contient la majorité du code liée au rendu graphique, on commencera par donner un diagramme qui représente cette classe et on s'intéressera principalement à la méthode drawRoad qui s'occupe de la majorité de ce qui est dessiné sur l'écran :



Plutôt que s'intéresser au code lui-même, une bonne compréhension de la projection pseudo3D sera plus importante. Le cœur de cette projection est la loi de Thalès voici un diagramme qui illustre ces propos




H représente la hauteur de la caméra, z la distance entre le véhicule et la camera et d la distance entre la camera et l'écran (le plan rouge) en utilisant la loi de Thalès on peut conclure y = h*d/z.

Si on représente ce diagramme avec une vue de dessus et en appliquant encore une fois la loi de Thalès on obtient le résultat suivant : x = w * d /z.

Maintenant on aura besoin de projeter ces valeurs pour obtenir les valeurs à afficher sur l'écran la fonction project de la classe Line le fait :

```
public void project(int camX, int camY, int camZ) {
        scale = camD / (z - camZ);
        X = (1 + scale * (x - camX)) * width / 2;
        Y = (1 - scale * (y - camY)) * height / 2;
        W = scale * roadW * width / 2;
}
```

Cette fonction fusionne trois étapes de la projection, la première étant la transformation des coordonnées du monde 3d en coordonnées sur le plan de la caméra ensuite la normalisation des données et enfin la mise à l'échelle des coordonnées pour les afficher sur l'écran.

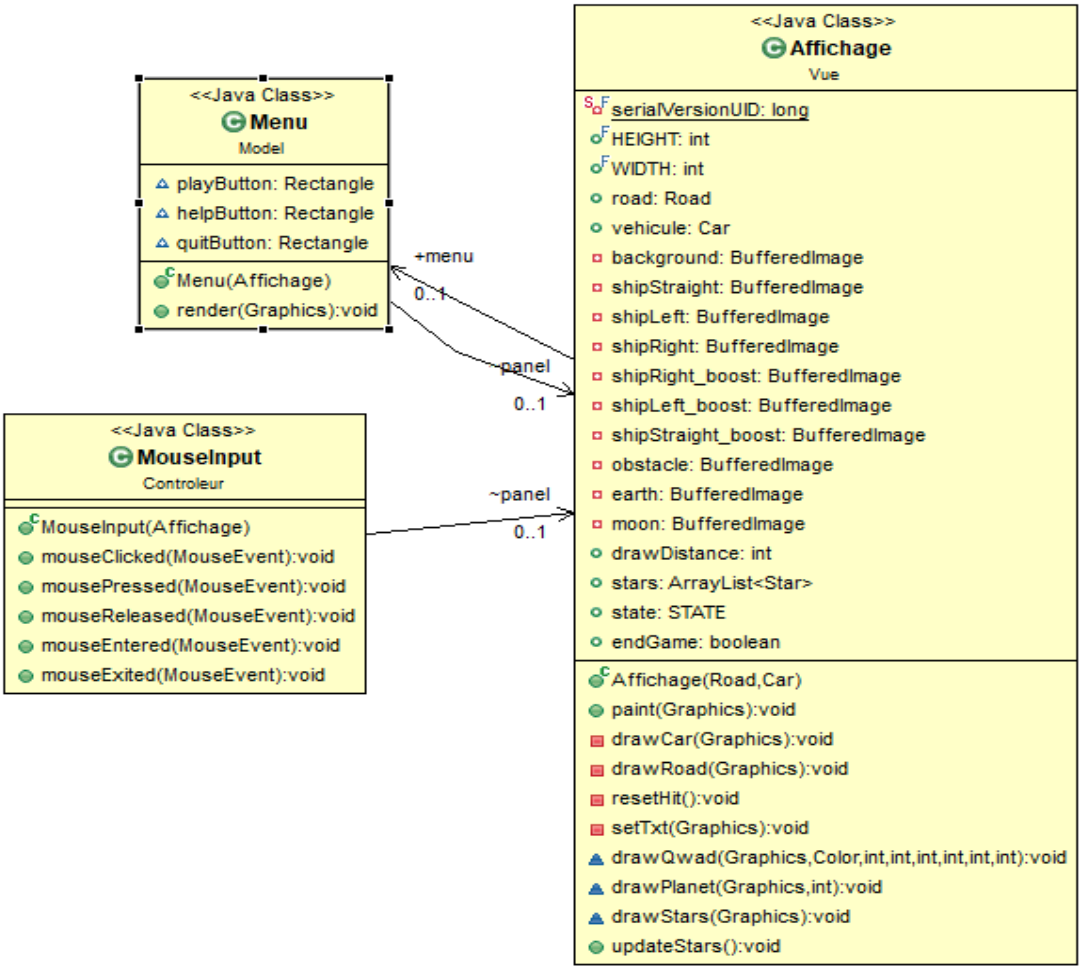
## 5.4-Le décor

Le décor était simple à implémenter, pour le champ d'étoiles on utilise un algorithme naïf qui génèrent des étoiles de manière aléatoire ensuite à l'aide d'un thread décor on met à jour leurs position. Lorsque les étoiles atteignent une certaine distance on reset le processus pour recréer l'animation. Au moment de l'affichage on teste la distance au joueur et en fonction de cette valeur en augmente la taille de l'étoile représenté par un ovale.

Pour les planètes rien de plus simple, on stocke la valeur de l'abscisse du dernier point de la route et on met à jour la position des images en fonction de cette valeur.

## 5.5-Le menu

La classe Menu et MouseInput s'occupe du fonctionnement du menu voici un petit diagramme montrant la relation :

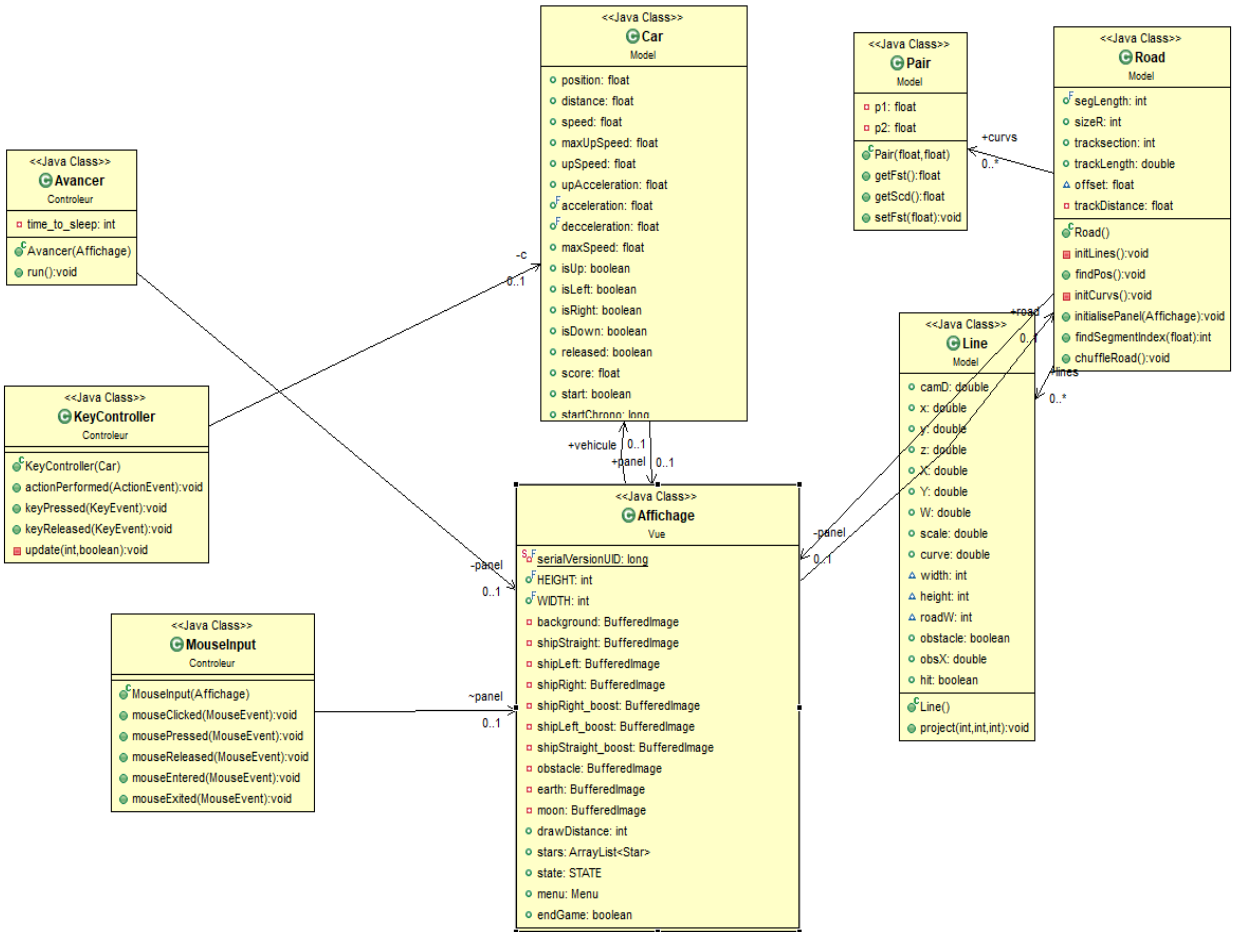



La classe affiche possède un attribut de type State qui permet d'identifier l'état du jeu si cet attribut vaut Menu le MouseInput réagit aux cliques de la souris et si jamais l'utilisateur clique sur l'un des boutons le MouseInput déclenchera les actions associées.

Les boutons du menu sont des simples rectangles et pour simuler les cliques du bouton on teste les coordonnées de la clique de la souris si jamais il se trouve dans un rectangle on déclenche l'action associée.

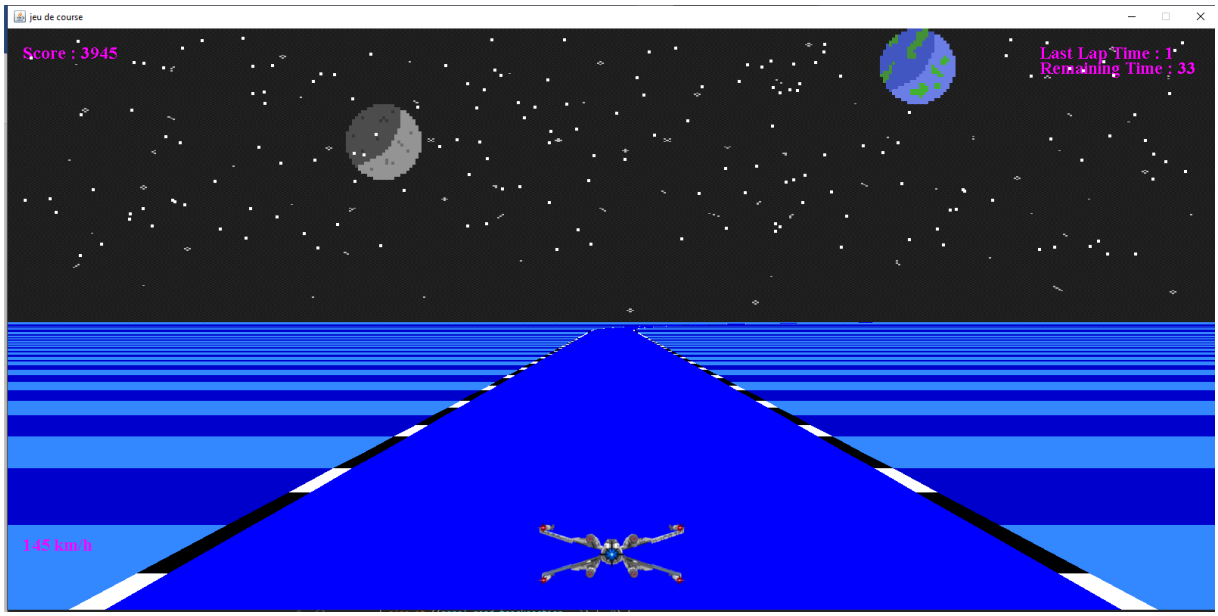
```
public void mousePressed(MouseEvent e) {
    int mx = e.getX();
    int my = e.getY();
    //Teste si on clique sur le bouton play
    if (mx >= panel.WIDTH/2-50 && mx<=panel.WIDTH/2-50+100) {
        if (my>=350 && my<=400) {
            panel.state = STATE.GAME;
        }
    }
    //teste si on clique sur le bouton quit.
    if (mx >= panel.WIDTH/2-50 && mx<=panel.WIDTH/2-50+100) {
        if (my>=550 && my<=600) {
            System.exit(1);
        }
    }

}
```

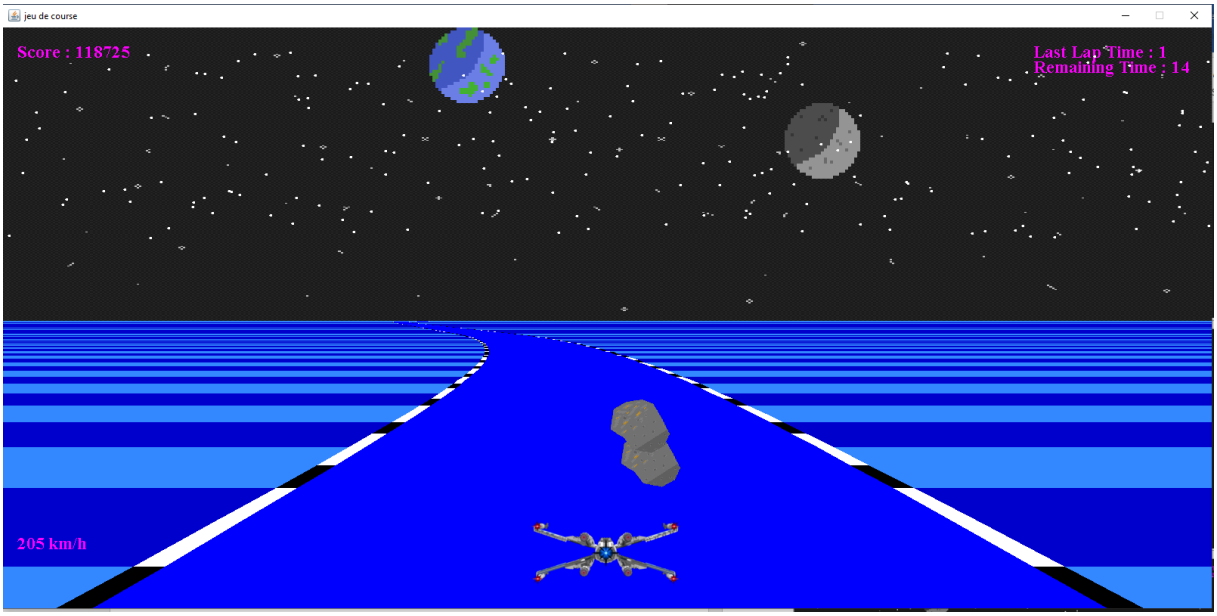
## 5.6-Diagramme de classe




## 6. **Résultat**

Rapport PCII
Ben Naceur Fedy




# 7.Documentation utilisateur

Prérequis : Java.

Mode d'emploi : double-cliquez sur l'icône du fichier .jar puis cliquez sur « Play » pour jouer.

Pour le fonctionnement de l'application le joueur sera accueilli par un menu ou il choisira « Play », à l'aide de la souris, le jeu se lance.

Pour naviguer dans le monde on utilisera les flèches gauche et droite du clavier pour de déplacer horizontalement sur la piste, on pourra aussi se déplacer verticalement à l'aide de la flèche haut en faisant cela le véhicule perdra de la vitesse progressivement jusqu'à atteindre zéro.

Durant la phase de jeu on pourra mettre le jeu en pause en cliquant sur la touche p, on recommence à jouer en cliquant sur « Play » avec la souris.

Le but du jeu est d'atteindre les checkpoints dans le temps alloué qui diminuera en fonction de votre avancement et score. Le joueur devra donc éviter les obstacles trouver sur son champ pour éviter la perte de vitesse.

# 8.Documentation développeur

## Structure du code

La classe main de ce projet se trouve dans la classe Main du package affichage. Pour avoir plus de control sur les paramètres du jeu les classes Car, Line, Affichage et Road seront les plus utiles.

Les constantes principales à modifier sont les suivantes :

- camD de Line : cette constante représente la distance entre la caméra et le joueur.
- roadW de Line : cette constante représente la largeur de la route sur l'écran.

- speedMax de Car :cette constante représente la vitesse  maximale du véhicule.
- Acceleration de Car : cette constante représente l'accélération du véhicule.
- upAcceleration de Car : cette constante représente l'accélération verticale du véhicule.
- upSpeedMax de Car : cette constante représente la vitesse verticale maximale.
- estimatedCrossTime : représente le timer pour atteindre le prochain checkpoint.
- Pour la classe affichage on pourra modifier le nombre d'étoiles présentes sur l'écran mais se limiter à 500 est la meilleure solution puisque ça devient perturbant pour le joueur.

### Fonctionnalités non développées

Obstacles « mobiles » sur la piste : Par manque de temps cette fonctionnalité n'a pas été implémenter. Cette fonctionnalité sera difficile à implémenter vu le choix de structure pour représenter la route. On aura besoin d'associer chaque véhicule adversaire à une section après créer une fonction qui simulera le mouvement de l'adversaire en prenant en compte la position des autres autour de lui.

Ce choix a été fait pour privilégier l'amélioration du rendu graphique qui pour ce type de jeu est important.

De plus un système qui enregistre les meilleures scores du joueur pourra être implémenté, pour le faire on enregistrera dans un fichier les scores et on y accèdera si on veut les afficher ou bien pour vérifier si on a nouveau meilleur score ou pas.

## 9.Conclusion et perspectives

On a donc réussi à implémenter un jeu de course similaires à ceux des années 80 en vue à la première personne ainsi qu'un menu qui permet d'accéder au jeu ou bien de le mettre en pause. L'originalité de notre projet et que notre véhicule peut se déplacer verticalement et donne la sensation de voler dans un cadre spatial.

Durant cette épreuve l'implémentation de la vue était un vrai « challenge » qui nécessitait une connaissance de la projection en espace pseudo3D, ainsi que plusieurs tests pour affinées les valeurs finalement choisies. Ces « challenges » nous ont permis d'avoir une très bonne connaissance sur l'utilisation des Threads, de l'API swing de Java ainsi que plusieurs techniques et astuces sur la projection en 2D.

Pour ce projet l'ajout des adversaires sera la prochaine étape puisque cela donnera une bonne immersion et un bon défi pour les joueurs. Le décor aussi pourra s'améliorer en ajoutant des images sur les bords de la route.