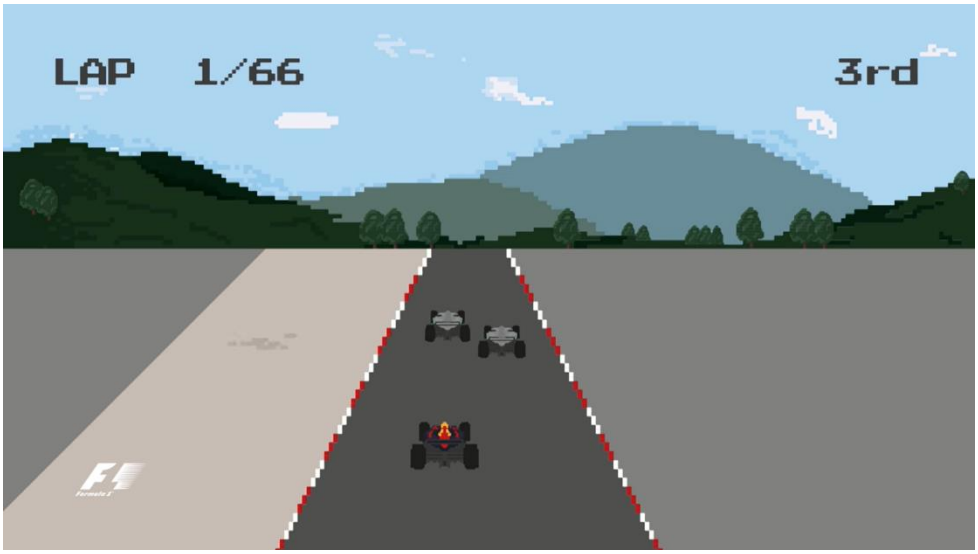
université
PARIS-SACLAY

Rapport du Projet PCII

UN JEU VIDEO DE TYPE « COURSE DE VOITURE »



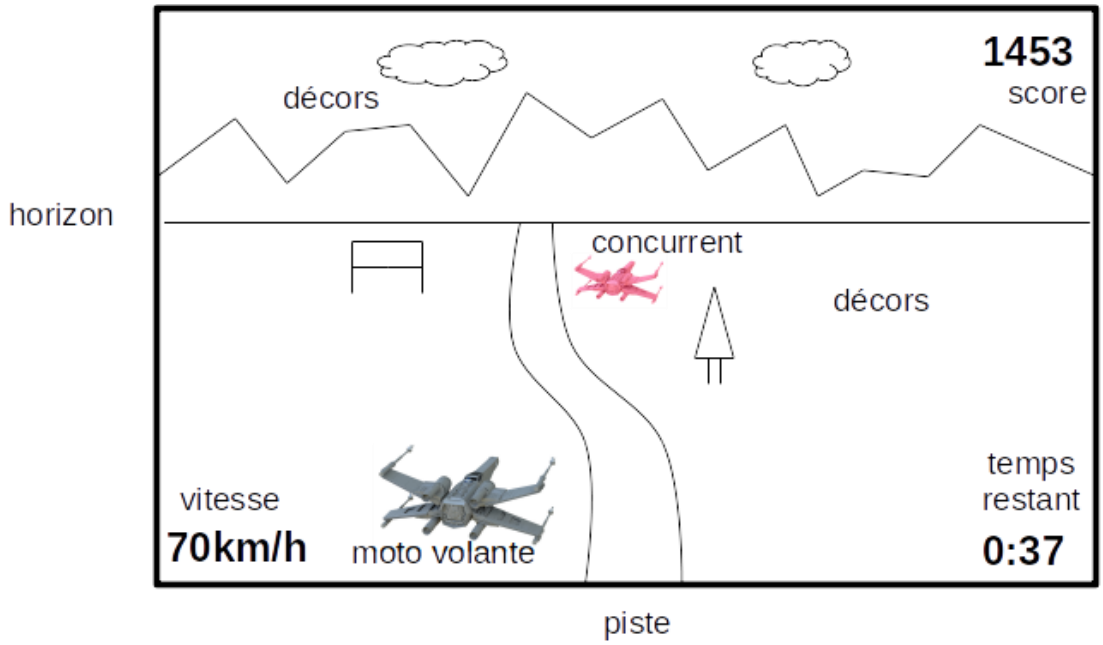
Référent : Thi-Thuong-Huyen NGUYEN

Jing ZHANG | Liuyi CHEN
GROUPE 2, L3 INFORMATIQUE

Table des matières

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

Pour réaliser le jeu, plusieurs spécifications principales sont attendues :

I. Une vue dans laquelle sont dessinés les éléments principaux.
    a. Un véhicule, représenté par un dessin ou par un ensemble d'images
    b. Un horizon (représenté par un trait) surmonté d'un décor.
    c. Une piste infinie, calculée à partir d'une ligne brisée verticale limitée par l'horizon et générée aléatoirement
    d. À intervalles réguliers apparaissent des points de contrôles matérialisés par une bande horizontale sur la piste.
    e. Affichage du score lorsque le jeu se termine.

II. Un mécanisme de contrôle au clavier
    a. Pour l'animation du déplacement des éléments dans la fenêtre
    b. Pour parcourir le menu lorsque la suspension du jeu

III. Un modèle de jeu comprenant des données et des implémentations des fonctionnalités
    a. L'état du véhicule : position et vitesse
    b. Mécanisme de calcul de l'accélération du véhicule en fonction de la position par rapport à la piste
    c. Mécanisme de calcul de la vitesse en fonction de l'accélération
    d. Des données de jeu : temps restant et kilométrage
    e. Gestion du temps supplémentaire alloué à joueur.
    f. Mécanisme et contrôle sur la suspension du jeu.

En outre, si le développement se déroule bien, il y aura des fonctionnalités plus complémentaires afin d'améliorer les performances du jeu et l'expérience de l'utilisateur :

IV. Fonctionnalités complémentaires
    a. Implémentation d'un thread d'affichage qui s'exécute tous les 1/24 seconde
    b. Apparition aléatoire d'obstacles sur la piste (générés hors du champ de vision et apparaissant au fur et à mesure).
    c. Le dessin du véhicule change en fonction des actions du joueur pour suggérer les mouvements à droite et à gauche.
    d. Le dessin de la piste se rétrécit au bout pour créer une sensation de profondeur.

Plan de développement

I. Liste des tâches

Séance	Tâches
Séance 4 **Mise en place du projet**	Découverte, Compréhension du sujet
	Analyse globale
	Mise en œuvre de la structure MVC (Squelette de l'application)
	Rédaction du plan de développement
	Création des données du véhicule
	Recherche d'images pour le véhicule
	Implémentation d'une vue minimaliste
	Gestion du clavier en continu
	Documentation du projet v0.1
Jalon : la piste est représentée par une simple ligne brisée fixe et limitée par un horizon, le véhicule est une simple croix qui se déplace d'une valeur fixe lorsqu'on appuie sur les touches	
Séance 5 et 6	Animation de la piste à vitesse constante
	Décompte des kilomètres et affichage
	Mouvements du décor de fond selon les touches du clavier
	Amélioration du décor
	Implémentation de la courbe Bézier
	Calcul de l'accélération en fonction de la position par rapport à la piste
	Gestion de la vitesse
	Détection de collisions qui ralentissent le véhicule
	Mécanisme de contrôle sur la suspension du jeu
	Documentation du projet v0.2
Jalon : la piste est animée, le véhicule se déplace naturellement, il est représenté par une image : il ne manque plus que la mécanique du jeu	
Séances 7 et 8 **Mécanique du jeu**	Ajout d'obstacles et détection de collisions qui ralentissent le véhicule
	Ajout de points de contrôles (représentés par une ligne horizontale sur la piste)
	Ajout d'un décompte de temps et détermination de la fin de partie
	Ajout des concurrents qu'il faut dépasser
	Ajout d'adversaires (relativement facile)

	Mémorisation des meilleurs scores
	Documentation du projet v0.9
Jalon : ça y est, nous avons un premier jeu complet, avec un score et une difficulté réglable via nos constantes (nombre d'obstacles, vitesse de défilement…)	
Séance 9 et 10 **Finalisation du projet**	Ajout d'images dans le décor (avec changements d'états gérés par des threads, voir tutoriel), animation de la piste, aspects graphiques plus soignés
	Ajout d'un écran d'accueil
	Possibilité de contrôler la vitesse de la moto (accélérer/ralen-tir)
	Le dessin du véhicule change en fonction des actions du joueur
	Création d'une sensation de profondeur (relativement difficile)
	Documentation du projet v1
	Préparation de la soutenance
Jalon : livraison du projet au chargé de TD	

II. Diagramme de Gantt

- **4ème Séance :**

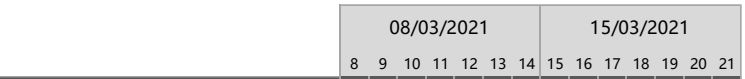
La première séance nous avons travaillé ensemble. Comme nous pensons que c'est important de se mettre d'accord sur la structure globale du projet au tout début du développement en binôme. Cela nous assura également une compréhension commune sur ce qui doit être réalisé spécifiquement à la future.

						01/02/2021						
						1	2	3	4	5	6	7
Tâche	**Personne**	**Progression**	**Début**	**Fin**	**Durée**	d	d	d	d	d	d	d
Séance 4												
Découverte, Compréhension du sujet	Liuyi & Jing	100%	01/02/2021	01/02/2021	45mins							
Analyse globale		100%	01/02/2021	01/02/2021	20mins							
Mise en œuvre de la structure MVC (Squelette de l'application)		100%	01/02/2021	01/02/2021	20mins							
Rédaction du plan de développement		100%	01/02/2021	01/02/2021	45mins							
Création des données du véhicule		100%	01/02/2021	01/02/2021	25mins							
Recherche d'images pour le véhicule		100%	01/02/2021	01/02/2021	5mins							
Implémentation d'une vue minimaliste		100%	01/02/2021	01/02/2021	40mins							
Gestion du clavier en continu		100%	06/02/2021	06/02/2021	45mins							
Documentation du projet v0.1		100%	06/02/2021	06/02/2021	1h							

- **5ème et 6ème Séance**

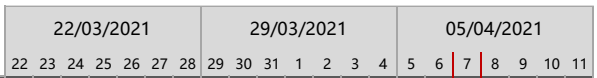
Tâche	Personne	Progression	Début	Fin	Durée
Séance 5 et 6					
Animation de la piste à vitesse constante	Liuyi	100%	08/02/2021	08/02/2021	1h30
Décompte des kilomètres et affichage	Jing	100%	08/02/2021	08/02/2021	1h
Mouvements du décor de fond selon les touches du clavier	Jing	100%	08/02/2021	08/02/2021	1h30
Amélioration du décor	Liuyi	100%	15/02/2021	15/02/2021	1h
Implémentation de la corbe Bézier	Liuyi	100%	15/02/2021	15/02/2021	1h
Mécanisme de contrôle sur la suspension du jeu	Liuyi	100%	15/02/2021	15/02/2021	40mins
Calcul de l'accélération en fonction de la position par rapport à la piste	Jing & Liuyi	100%	20/02/2021	23/02/2021	2h
Détection de collisions qui ralentissent le véhicule	Liuyi	100%	23/02/2021	23/02/2021	1h30
Documentation et des commentaires du projet v0.2	Jing & Liuyi	100%	15/02/2021	23/02/2021	des heures

- **7ème et 8ème Séance**



Tâche	Personne	Progression	Début	Fin	Durée
Séance 7 et 8					
Gestion de la vitesse	Liuyi	100%	08/03/2021	09/03/2021	5h
Ajout d'obstacles et détection de collisions qui ralentissent le véhicule	Liuyi	100%	13/03/2021	14/03/2021	5h
Ajout d'un décompte de temps et détermination de la fin de partie	Liuyi	100%	15/03/2021	15/03/2021	2h
Le dessin du véhicule change en fonction des actions du joueur	Liuyi	100%	15/03/2021	15/03/2021	30mins
Ajout le mécanisme des points de contrôles (représentés par une ligne horizontale sur la piste)	Liuyi	100%	20/03/2021	21/03/2021	6h
Documentation du projet v0.9	Jing&Liuyi	100%	08/03/2021	21/03/2021	des heures

- **9ème et 10ème Séance**



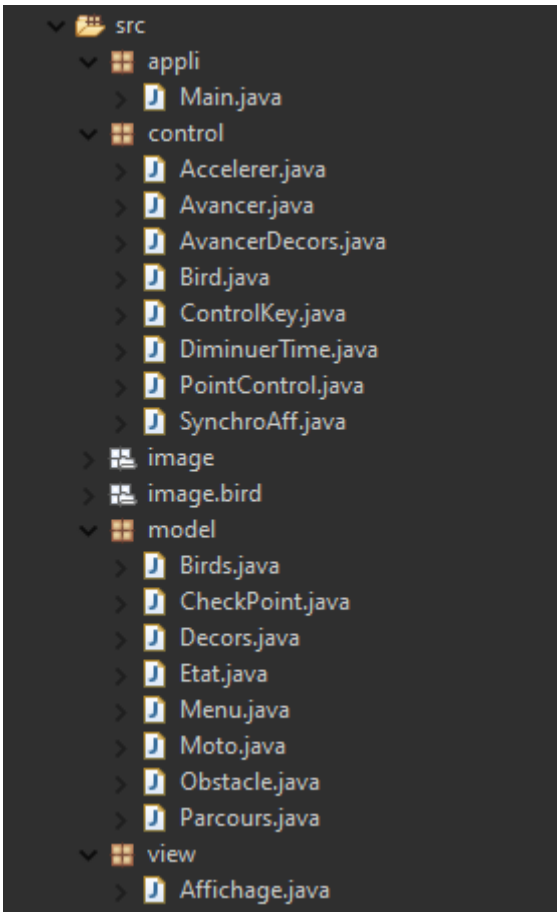
Tâche	Personne	Progression	Début	Fin	Durée
Séance 9 et 10					
Ajout d'images dans le décors, animation de la piste, aspects graphiques plus soignés	Liuyi	100%	22/03/2021	22/03/2021	1h30
Apparition aléatoire d'obstacles	Liuyi	100%	22/03/2021	25/03/2021	6h
Amélioration la fonctionnalité et l'affichage des points de contrôles	Liuyi	100%	29/03/2021	29/03/2021	3h
Ajout d'un petit menu	Liuyi	100%	29/03/2021	29/03/2021	2h
Le dessin de la piste se rétrécit au bout	Liuyi	100%	03/04/2021	06/04/2021	8h
Défilement d'objets sur le fond et du décors en fonction du véhicule	Jing	100%	05/04/2021	06/04/2021	2h
Possibilité de contrôler la vitesse de la moto (accélération)	Liuyi	100%	06/04/2021	07/04/2021	2h
Documentation du projet v1	Jing&Liuyi	100%	22/03/2021	07/04/2021	Des heures

Conception générale

En premier lieu, l'architecture du développement est basée sur le motif Modèle Vue Contrôleur (MVC). Ce pattern permet de bien organiser le code source. Il nous aide à bien définir le rôle de chaque fichier. En effet, le but est de séparer la logique du code en trois parties que l'on retrouve dans des fichiers distincts :

- Un état (Model) : définit l'ensemble des données qui caractérisent l'état de l'interface.
- Un affichage (View) : définit comment l'état du modèle est rendu visible à l'utilisateur.
- Un contrôleur (Control) : définit la manière dont l'état du modèle change. Il effectue les changements dans le modèle et informe la vue d'un changement. Lorsque l'interface est interactive, c'est aussi lui qui gère les événements.

Dans le projet, nous avons trois packages qui représentent respectivement le modèle, l'affichage et le contrôle comme le montre l'image ci-dessous. De plus, la classe *Main* qui fait démarrer le programme est placée individuellement dans le package *« appli »* et les images pour dessiner les éléments se trouvent dans le package *« image ».*

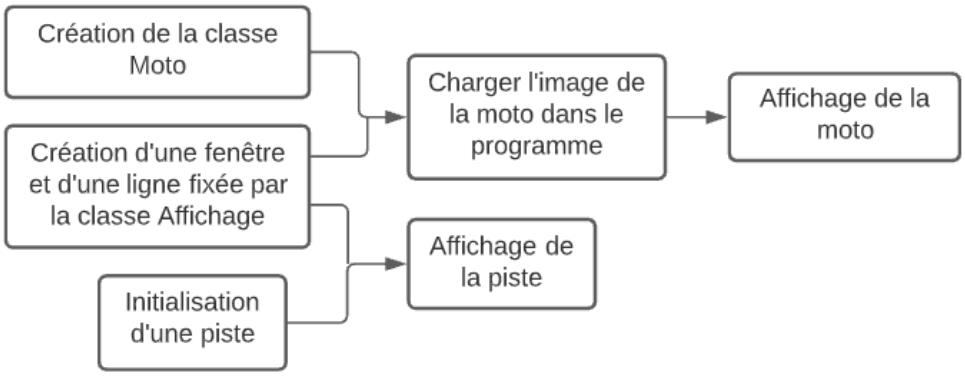


L'Architecture MVC du projet

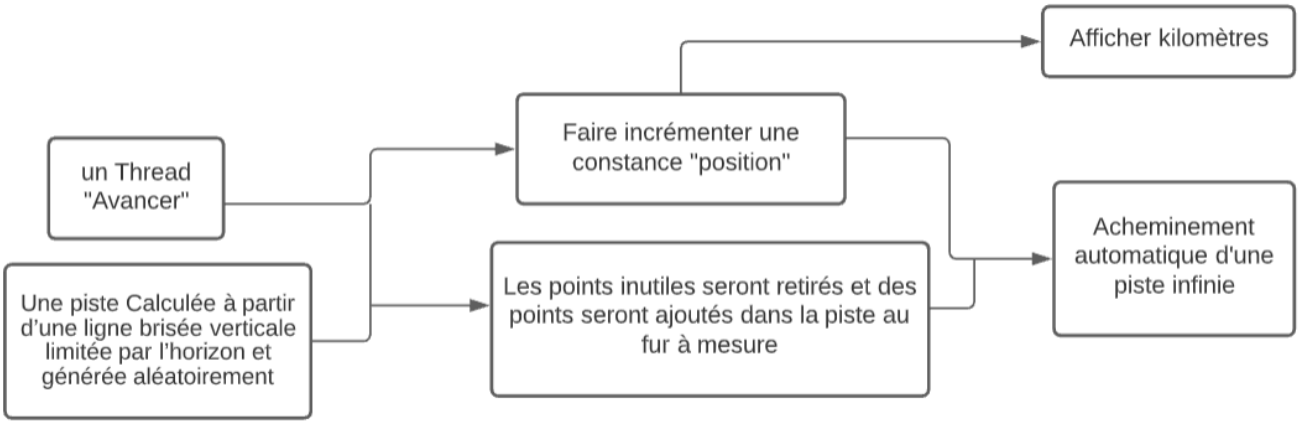
Ensuite, les fonctionnalités spécifiques sont présentées sous forme de « blocs fonctionnels ».

I. Une vue dans laquelle sont dessinés les éléments principaux

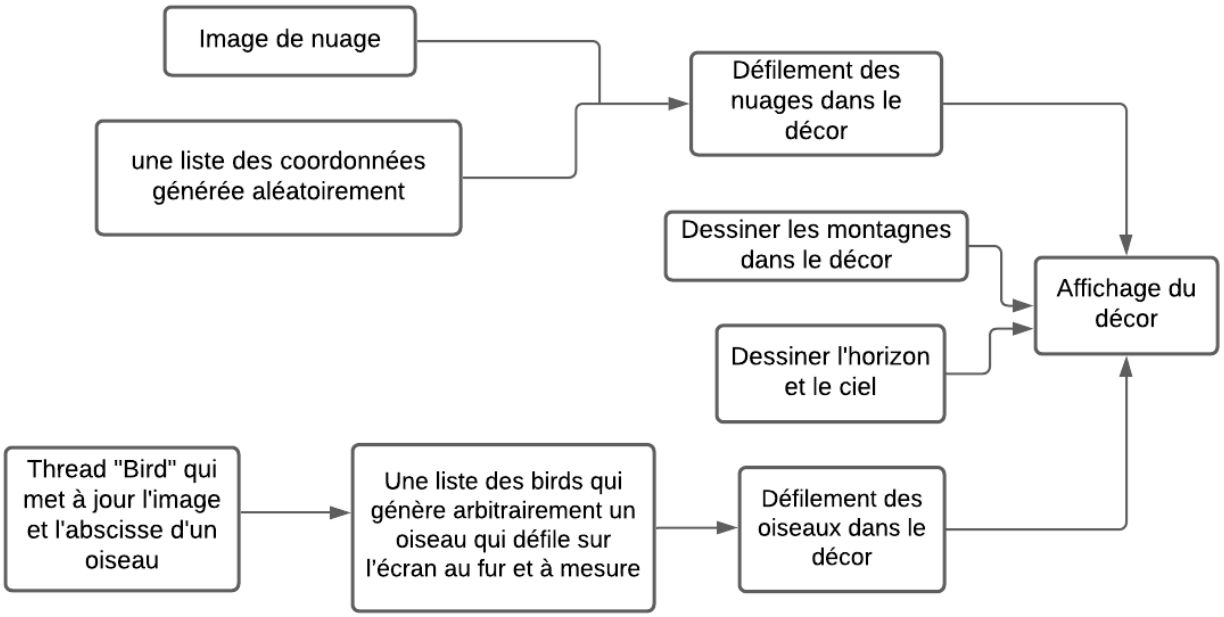
a. Affichage d'une moto et une piste



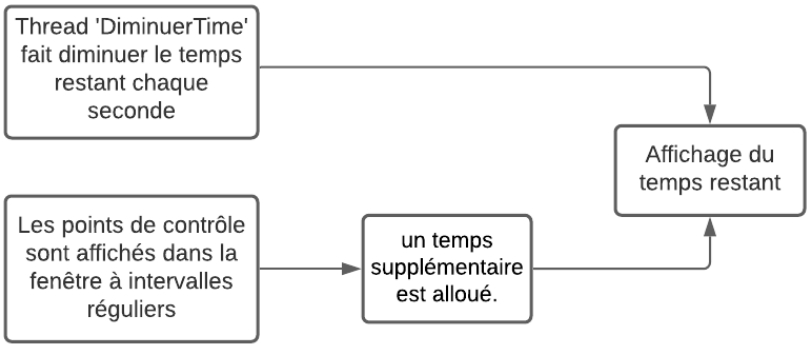
b. Une piste infinie et affichage du kilométrage



c. Affichage de l'horizon et du décor

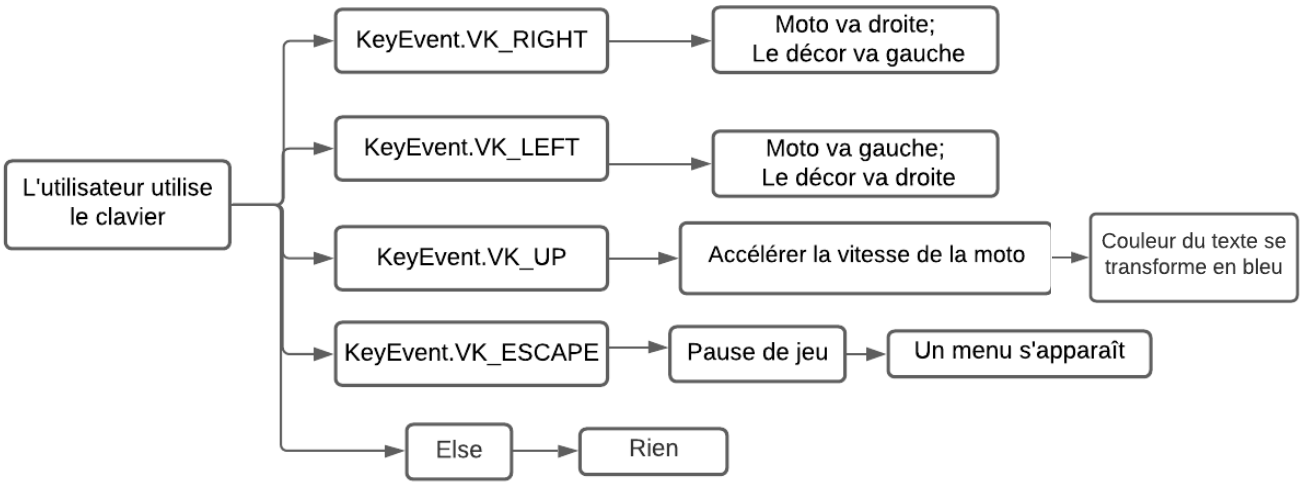


d. Affichage du temps restant et les points de contrôles.



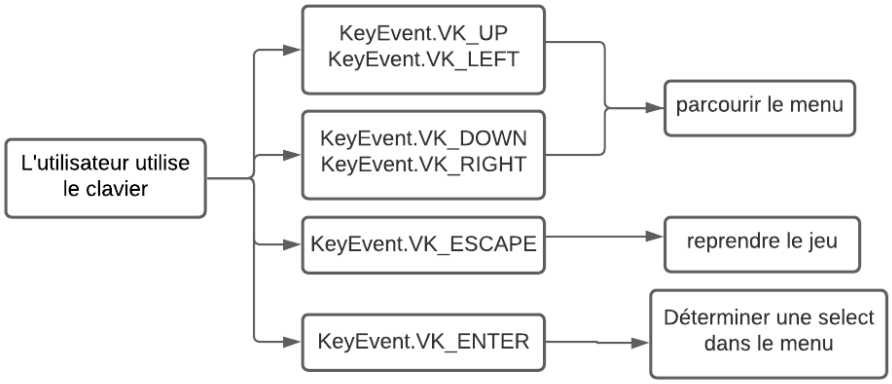
II. Un mécanisme de contrôle au clavier

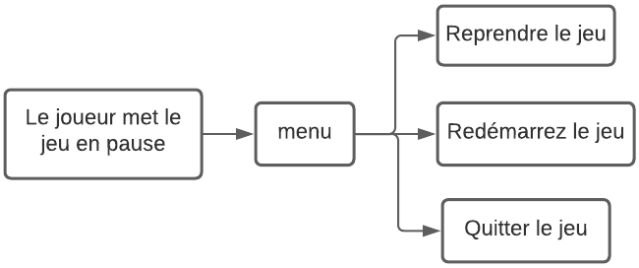
a. Pour l'animation du déplacement des éléments dans la fenêtre



b. Pour parcourir un menu lorsque la suspension du jeu

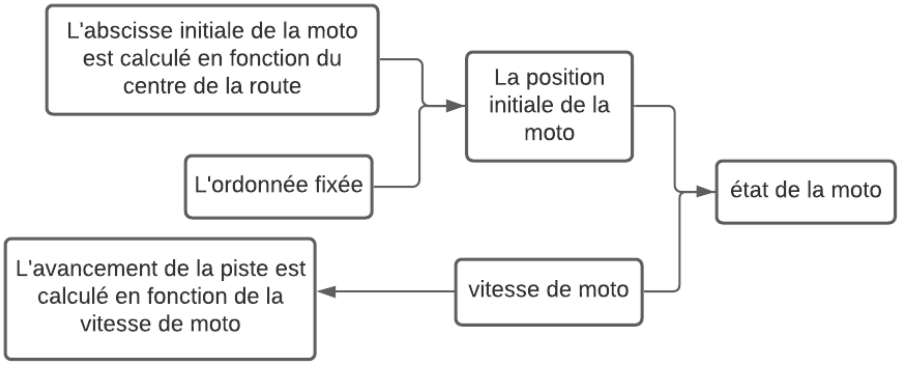
Lorsque le jeu est en pause, un menu s'affichera. L'utilisateur ne peut utiliser le menu que par le clavier.



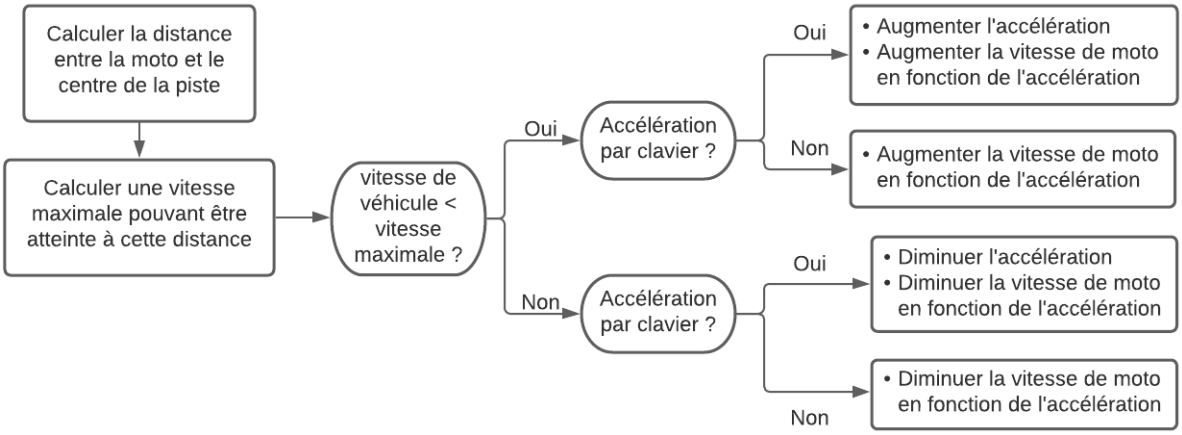


III. Un modèle de jeu comprenant des données et des implémentations des fonctionnalités

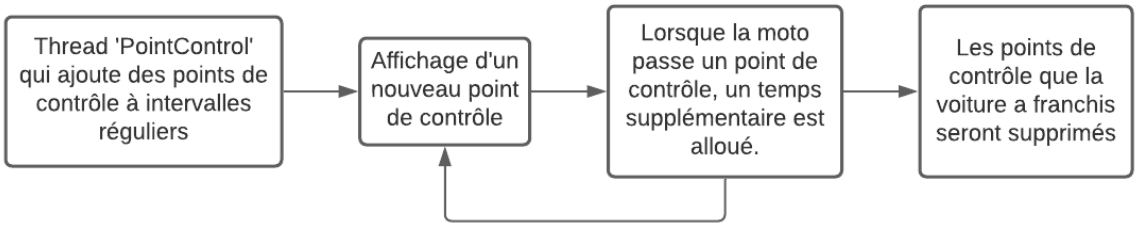
a. L'état du véhicule : position et vitesse



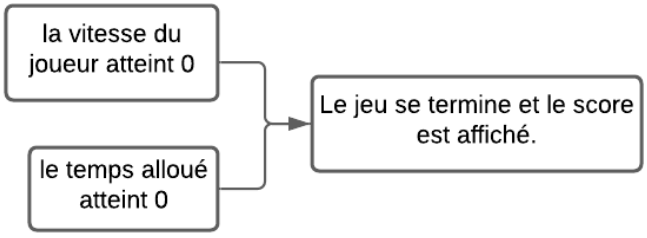
b. Mécanisme de calcul de l'accélération et de la vitesse



c. Mécanisme de point de contrôle (Gestion du temps supplémentaire alloué à jouer).

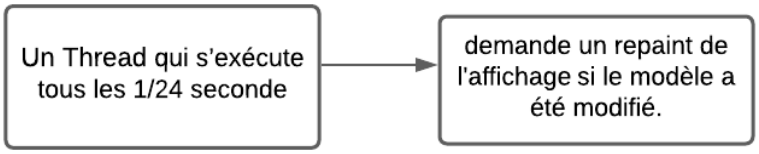


d. Mécanisme de contrôle sur la suspension du jeu.

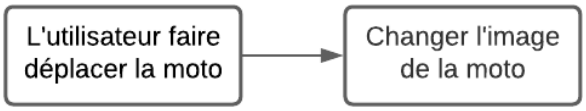


IV. Fonctionnalités complémentaires.

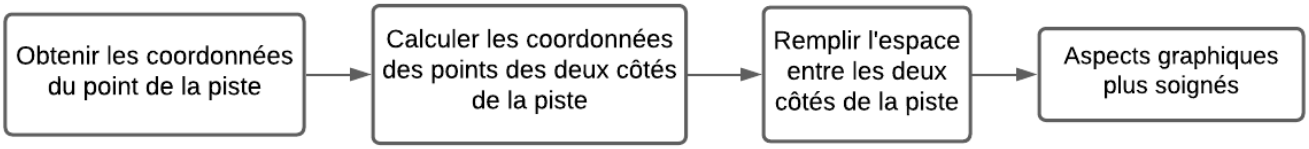
a. Implémentation d'un thread d'affichage qui s'exécute tous les 1/24 seconde



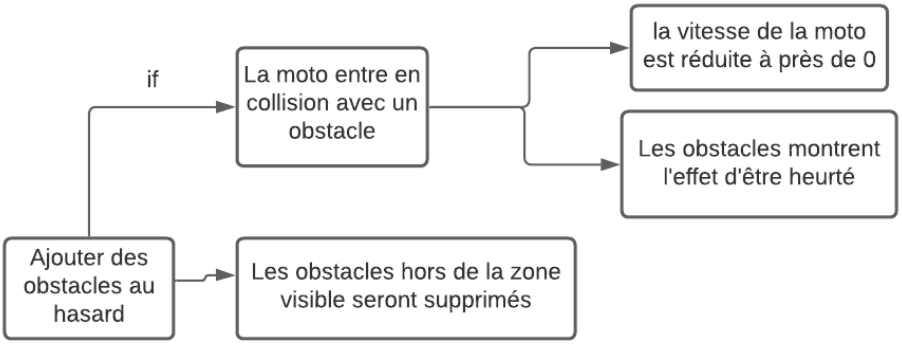
b. Le dessin du véhicule change en fonction des actions du joueur.



c. Le dessin de la piste se rétrécit au bout pour créer une sensation de profondeur.

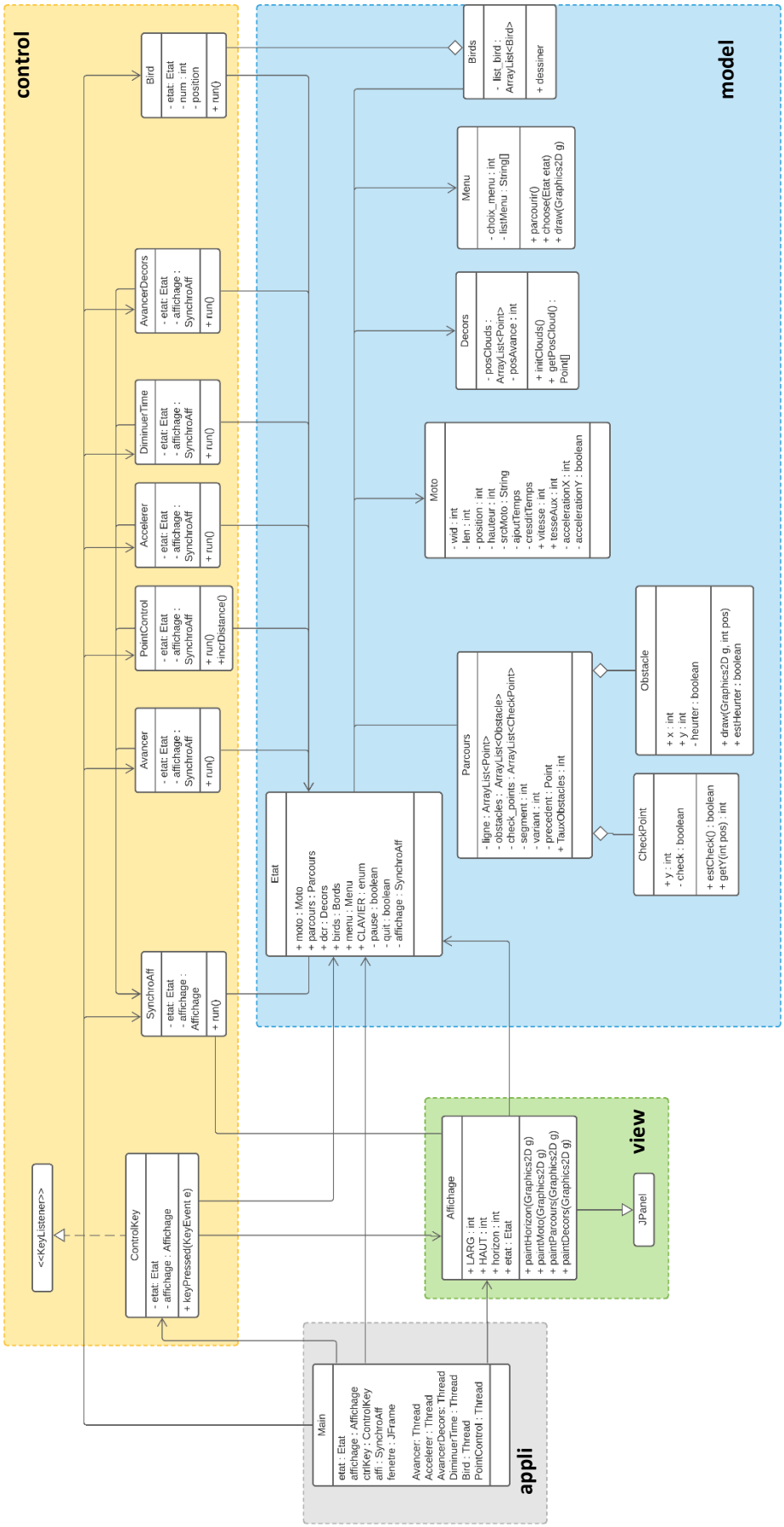


d. Apparition aléatoire d'obstacles sur la piste.



Conception détaillée

Nous allons d'abord vous montrer un diagramme global comme l'image montre ci-dessous. Elle correspond au pattern MVC que nous avons introduit dans la conception globale.

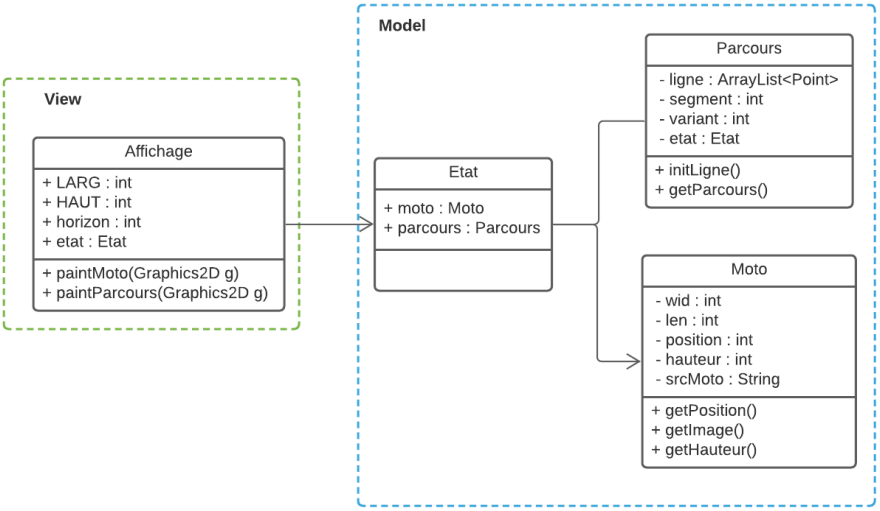


Dans notre projet, nous avons utilisé l'API Swing et la classe JPanel pour l'affichage de notre interface graphique. Pour importer des images, nous avons utilisé des méthodes de la classe *BufferImage* et de la classe File. La classe « java.awt.Point » nous permet de définir des objets caractérisés par des coordonnées (x,y). L'utilisation de threads permet de réaliser de la programmation concurrente, c'est-à-dire de donner l'impression qu'un programme effectue plusieurs choses en parallèle. Tous les threads sont démarrés dès le départ du jeu, ils sont lancés dans la classe main. Dans le thread, on fait incrémenter un compteur à chaque tour de la boucle en utilisant *Thread.sleep*.

Ensuite, nous vous expliquerons en détail toutes les fonctions que nous avons implémentées dans l'ordre.

I. Une vue dans laquelle sont dessinés les éléments principaux

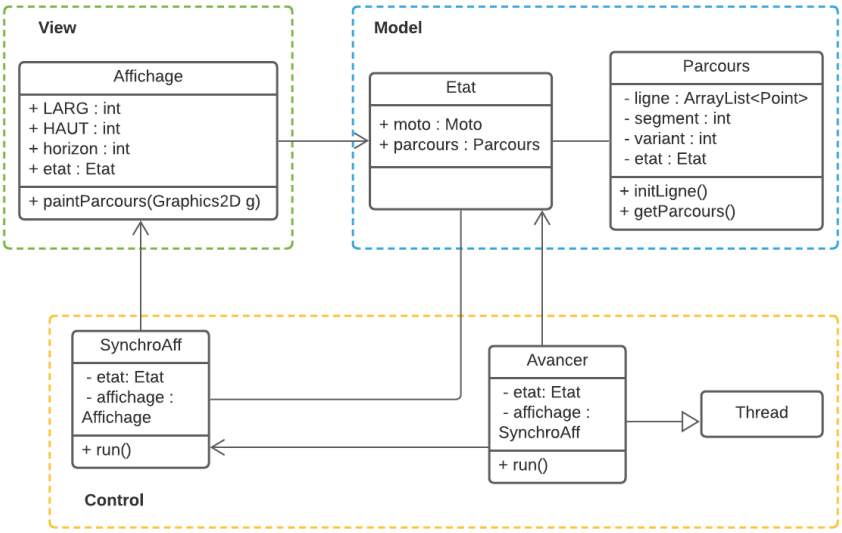
1) Affichage d'une moto et une piste



Dans la classe **MOTO**, *il y a la taille, les coordonnées, les images, le temps restant du jeu, l'accélération et la vitesse de la moto. Pour l'affichage de la moto, nous chargeons d'abord son image dans notre programme et puis elle est dessinée dans la classe* **AFFICHAGE** *grâce à la méthode « drawImage » de la classe Graphics.*

Pour la piste, nous avons créé une classe **PARCOURS** dont l'attribut principal est une liste de points. Au début du projet, le chemin est représenté par une ligne pointillée fixe. Au fur et à mesure de l'avancement du projet, des améliorations seront apportées pour la rendre plus animée à l'avenir. La méthode « initLigne » prend en charge l'initialisation de la piste et elle est appelée dans le constructeur. Lors de l'initialisation, des valeurs aléatoires d'ordonnée croissante sont généralisées. *Afin que la position de départ de la moto soit au centre de la piste, nous initialisons ses coordonnées en fonction de la position initiale de la piste ici.*

2) Animation d'une piste infinie et affichage du kilométrage



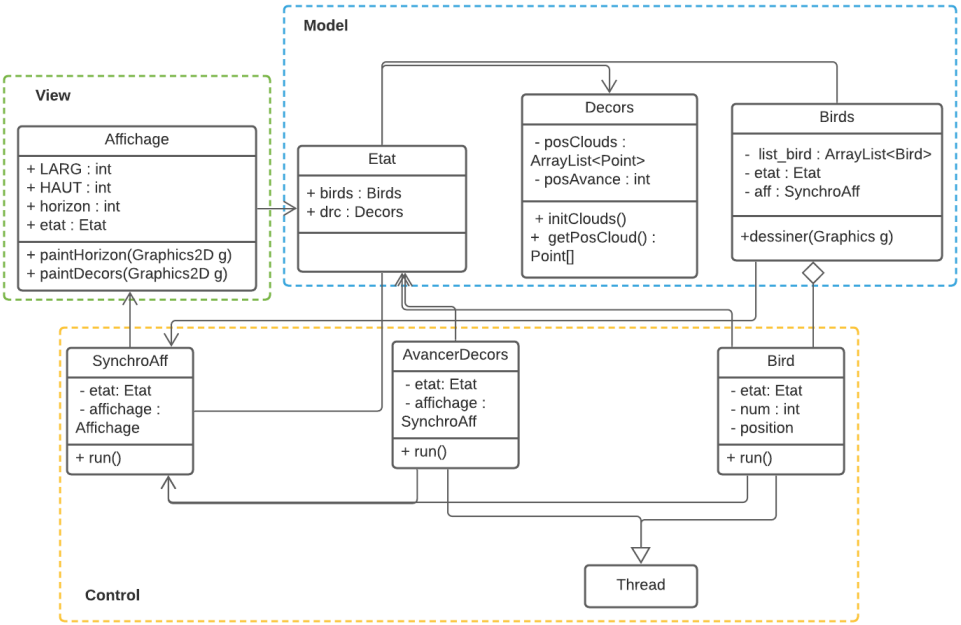
Nous appelons la méthode « avancer » de **PARCOURS** *à l'aide d'un thread* **AVANCER** *pour donner l'impression que la piste avance automatiquement. Dans cette méthode, on augmente la valeur de la variable « position » par une valeur calculée en fonction de la vitesse de la moto. De cette manière, la vitesse de la piste est étroitement liée à la vitesse de la moto, et cela donne l'impression que le changement de la vitesse de moto permet de contrôler la vitesse de la piste.*

Ensuite, nous utilisons la méthode « getParcours » pour ajouter en continu la valeur de « position » à chaque ordonnée du point de la ligne initiale. Comme mentionné précédemment, nous faisons augmenter continuellement la valeur de « position », on aura donc l'impression que la piste défile toute seule vers le bas. Afin de donner l'impression que la piste est infinie, lorsque son dernier point rentre dans la zone visible, on va générer un point supplémentaire pour que la ligne brisée ne s'interrompe pas. Lorsqu'un point est trop loin de la zone visible, on le retire de la liste. La méthode « getParcours » renvoie une copie de la ligne avec les points visibles dans la fenêtre. La méthode « paintParcours » d'**AFFICHAGE** dessine donc la piste à l'aide de « drawLine » de la classe Graphics.

Evidemment, la variable « position » représente le kilométrage de la moto. Donc la méthode « paintTexte » de classe **AFFICHAGE** *dessine le kilométrage en appelant la méthode « getParcours » de* **PARCOURS** *.*

3) Affichage de l' horizon et du décor

L'horizon de la fenêtre est défini comme une constante dans la classe **AFFICHAGE**. Il se situe dans le quart supérieur de la fenêtre. La méthode « paintHorizon », « paintDecors » rend l'interface plus soignée. « paintHorizon » remplit la zone au-dessous de l'horizon qui présente le ciel. La méthode « paintDecors » prend en charge l'affichage du décor, elle dessine les nuages en récupérant les coordonnées et l'image correspondante. Mais aussi, elle dessine également des Montagnes en arrière-plan grâce à la méthode « fillPolygon » de la classe Graphics.

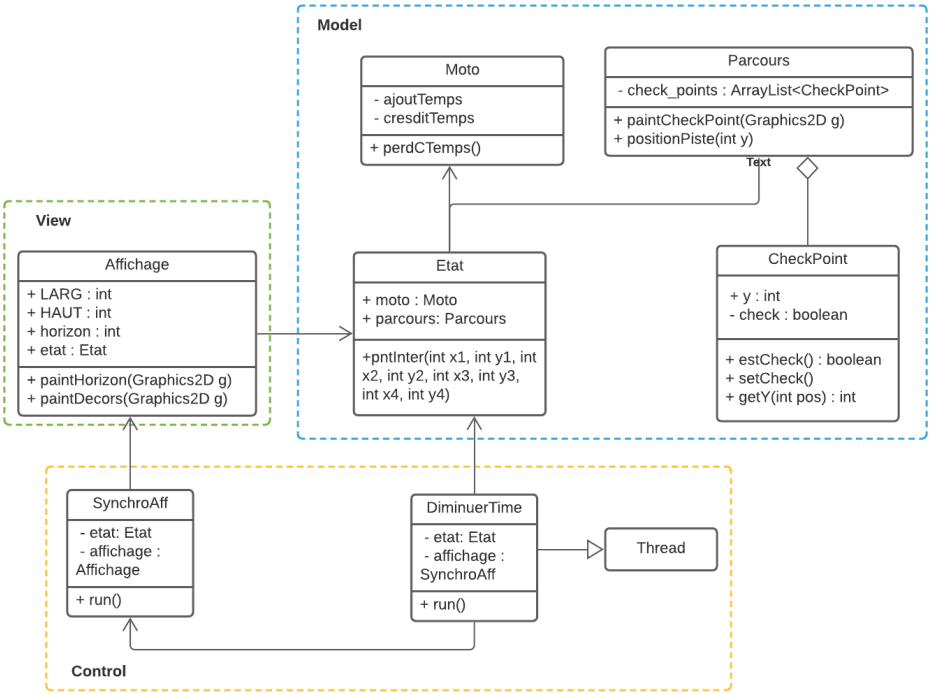


Nous avons créé la classe **DECORS** pour afficher le défilement des nuages dans le ciel dans la fenêtre. La réalisation du mouvement des nuages est très similaire à ce que l'on a présenté pour la piste. Nous avons créé un thread **AVANCERDECORS** pour faire avancer des nuages. Cependant, nous changeons ici l'abscisse des nuages avec la variable « posAvance » pour les faire avancer. Les coordonnées des nuages sont également générées aléatoirement, mais elles ne dépassent pas l'horizon.

Par ailleurs, nous avons créé une classe **BIRDS** qui faire afficher des oiseaux générés aléatoirement dans la fenêtre. Cette classe contient une liste de type ArrayList< **BIRD** > , il a aussi une méthode « dessiner » pour gérer l'affichage des oiseaux. Cette méthode place dans g l'image correspondant à l'état de l'oiseau, placée à la position courante, pour chaque élément de sa liste. Dans la classe **BIRD**, la fonction run() met à jour le « num » et la position de l'oiseau selon la variable délai. Un oiseau est retiré de la liste lorsqu'il est complètement sorti de la zone visible, et puis on génère un nouvel oiseau depuis la classe **BIRDS.**

En appelant la méthode « dessiner », nous pouvons donc voir des oiseaux se déplacer en battant des ailes dans le décor.

4) *Affichage du temps restant et les points de contrôles*



Un thread **DIMINUERTIME** appelle la méthode « perdCTemps » de classe **MOTO** pour faire diminuer la variable « creditTemps » (le temps restant) toutes les secondes, tant que le jeu n'est pas en pause et que le joueur ne perd pas le jeu.

AFFICHAGE appelle méthode « paintCheckPoint » de classe **ETAT** , pour dessiner les points de contrôles matérialisés par une porte sur la piste. Dans **PARCOURS**, il y a une liste contenant les ordonnées de tous les points de contrôle. Dans cette méthode, nous parcourons cette liste. Nous affichons uniquement les points de contrôle que la voiture n'a pas passés. Si le point de contrôle a déjà rencontré la voiture, alors il sera supprimé de la liste.

Lors du dessin d'un point de contrôle, nous devons calculer l'abscisse du point de contrôle sur la piste à l'aide de la méthode « positionPiste » du parcours qui détermine l'abscisse de la piste sur une ordonnée donnée. Elle utilise principalement la méthode « pntInter » de classe **ETAT.** Cette méthode est utilisée également dans plusieurs autres fonctionnalités : pour obtenir la coordonnée du point d'intersection à travers les coordonnées de quatre points donnés. La formule mathématique est présentée ci-dessous :

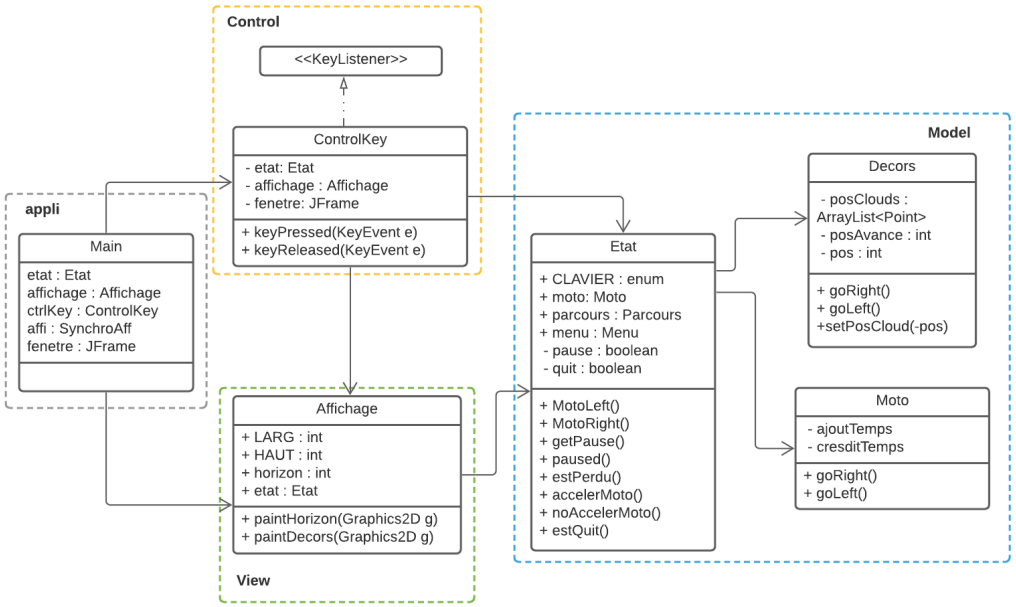
```
Point d'intersection Formule :
x = ((x1*y2-y1*x2)*(x3-x4)-(x1-x2)*(x3*y4-x4*y3))/((x1-x2)*(y3-y4)-(y1-y2)*(x3-x4))
y = ((x1*y2-y1*x2)*(y3-y4)-(y1-y2)*(x3*y4-x4*y3))/((x1-x2)*(y3-y4)-(y1-y2)*(x3-x4))
where
(x1,y1) starting point of segment 1
(x2,y2) ending point of segment 1
(x3,y3) starting point of segment 2
(x4,y4) ending point of segment 2
```

Chaque point de contrôle est dessiné sous forme de porte rouge sur la piste. Nous avons donc calculé la largeur de la piste pour obtenir une porte qui enjambe les extrémités gauche et droite de la piste.

II. Un mécanisme de contrôle au clavier pour l'animation du déplacement

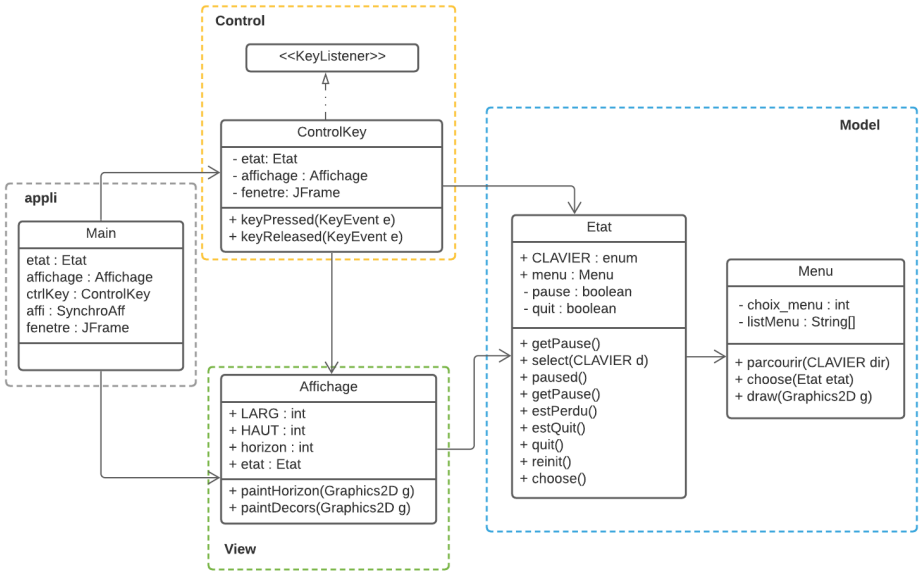
Nous avons créé une classe **CONTROLKEY** implémenté de l'interface KeyListener, il permet aux utilisateurs de contrôler le jeu via le clavier de l'ordi.

1) Pour l'animation du déplacement des éléments dans la fenêtre



Lorsque le jeu est en cours, si le joueur appuie sur le bouton de déplacement « gauche » / « droite » du clavier, la moto se déplacera vers la gauche / droite et le nuage d'arrière-plan se déplacera dans la direction opposée pour créer un effet de virage. Si le joueur appuie sur le bouton « haut », la moto accélérera et la police sur la fenêtre affichant le kilométrage, le temps restant et la vitesse deviendra bleue. Si vous relâchez ce bouton, la couleur de la police redeviendra bleue. Si le joueur souhaite mettre en pause ou redémarrer le jeu, il doit appuyer sur la touche d'échappement, le jeu est mis en pause et un menu apparaît.

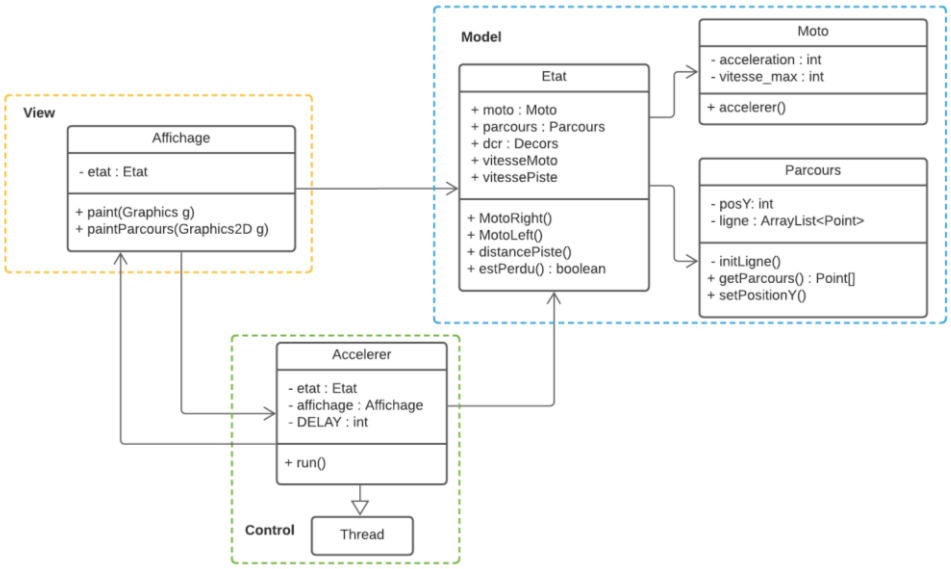
2) Pour parcourir un menu lorsque la suspension du jeu



Le jeu est mis en pause lorsque le menu apparaît. Les joueurs peuvent choisir dans le menu: reprendre le jeu, redémarrer le jeu et quitter le jeu. Pour l'utilisation du menu, si le joueur appuie sur la touche « haut » ou sur la touche « gauche », l'élément précédent du menu sera sélectionné; si la touche « bas » ou la touche « droite » est enfoncée, l'élément suivant du menu sera sélectionné, et le joueur doit appuyer sur la touche « entrer » pour confirmer sa sélection. S'il appuie sur la touche « escape » , cela équivaut à choisir de reprendre le jeu. Si le joueur choisir « quitter » , alors le jeu va terminer et la fenêtre sera fermée.

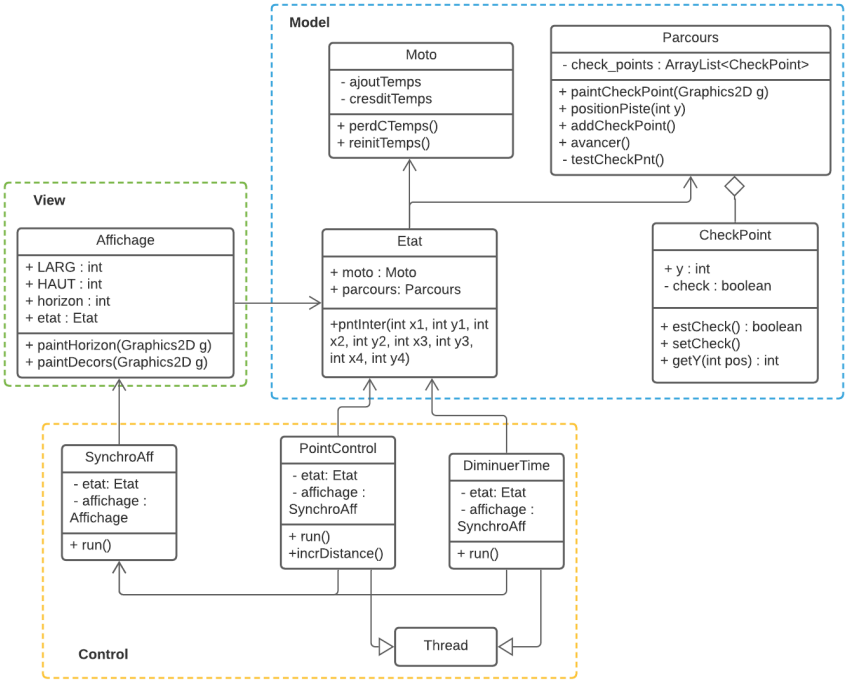
III. *Un modèle de jeu comprenant des données et des implémentations des fonctionnalités*

1) Mécanisme de calcul de l'accélération et de la vitesse du véhicule



Un thread **ACCELERER** appelle la méthode « accélérer » de classe **MOTO** pour le mécanisme d'accélération. Grâce à la méthode « distancePiste » de classe **PARCOURS** , nous pouvons obtenir la distance entre de la moto et la piste à partir de l'abscisse donnée. Nous utilisons cette distance pour calculer la vitesse (variable « vitesseAux ») que la moto peut atteindre cette distance. Si la vitesse actuelle de la moto est inférieure à « vitesseAux », cela signifie qu'il s'approche de la piste et on accélère en l'ajoutant à l'accélération; si la vitesse est supérieure à « vitesseAux », cela signifie qu'il s'éloigne de la piste, et nous soustrayons l'accélération à la vitesse actuelle pour ralentir. Si l'utilisateur accélère la voiture via le clavier, lorsque la voiture accélère, l'accélération sera plus grande; lorsque la voiture doit ralentir, l'accélération sera plus petite.

2) Mécanisme de point de contrôle



Un thread **PointControl** ajoute un objet **CheckPoint** à un intervalle régulier dans la liste « check_points » de classe **Parcours** . Un objet **CheckPoint** a l'ordonnée d'un point de contrôle, ainsi on peut voir si la moto a déjà passé ce point ou non. Lorsque la piste s'avance, il appelle la méthode «testCheckPnts » pour savoir s'il faut allouer un temps supplémentaire lors de la rencontre de la moto et un point de contrôle. Lorsque l'on dessine un point de contrôle, les points que la voiture a franchis seront supprimés.

3) Mécanisme de contrôle sur la suspension du jeu

Si la vitesse de la moto est à 0 ou le temps est à 0, alors le joueur est perdu. Dans ce cas-là, tous les threads s'arrêtent et un message s'affiche avec le score de l'utilisateur. Le score est calculé en fonction du kilométrage de la piste. Pour que l'utilisateur ne puisse plus déplacer le véhicule une fois il est perdu, nous désactivons ensuite l'état focalisable de classe **Affichage**.

IV. Fonctionnalités complémentaires.

1) Implémentation d'un thread d'affichage qui s'exécute toutes les 1/24 seconde

Dans notre ancienne version, chaque thread alerte le JPanel (**Affichage**) lorsqu'il effectue des modifications dans le modèle (avec des instructions revalidate et repaint) pour que la vue soit corrigée. Cela conduit naturellement à une surcharge au niveau de l'affichage, donc à des ralentissements, et donc à un visuel assez peu satisfaisant. Pour contourner ce problème, nous avons implémenté un thread d'affichage (**SynchroAff**) qui s'exécute tous les 1/24 seconde et qui demande un repaint si le modèle a été modifié. Les autres threads devront simplement prévenir ce thread d'affichage, et non plus directement la vue.

2) Apparition aléatoire d'obstacles sur la piste apparaissant au fur et à mesure.

Pour que les obstacles s'apparaissent arbitrairement au cours du défilement de la piste, nous prend aléatoirement une valeur de paramètre fixe avec un objet Random. Lorsque la valeur

aléatoire est égale à 0, nous ajouterons un objet **OBSTACLE** dans une liste d'obstacles déclarée dans classe **PARCOURS** . Lorsque nous ajoutons un **OBSTACLE**, nous utilisons la méthode « positionPiste » mentionnée précédemment pour déterminer les coordonnées de l'obstacle afin qu'il soit toujours autour de la piste. On détecte si la voiture entre en collision avec un obstacle dans la méthode « avancer » de classe **PARCOURS** . En cas de collision, la vitesse de la voiture chutera à près de zéro (mais pas égale à zéro, sinon le jeu se terminera), et la transparence de l'obstacle changera pour l'effet de collision.

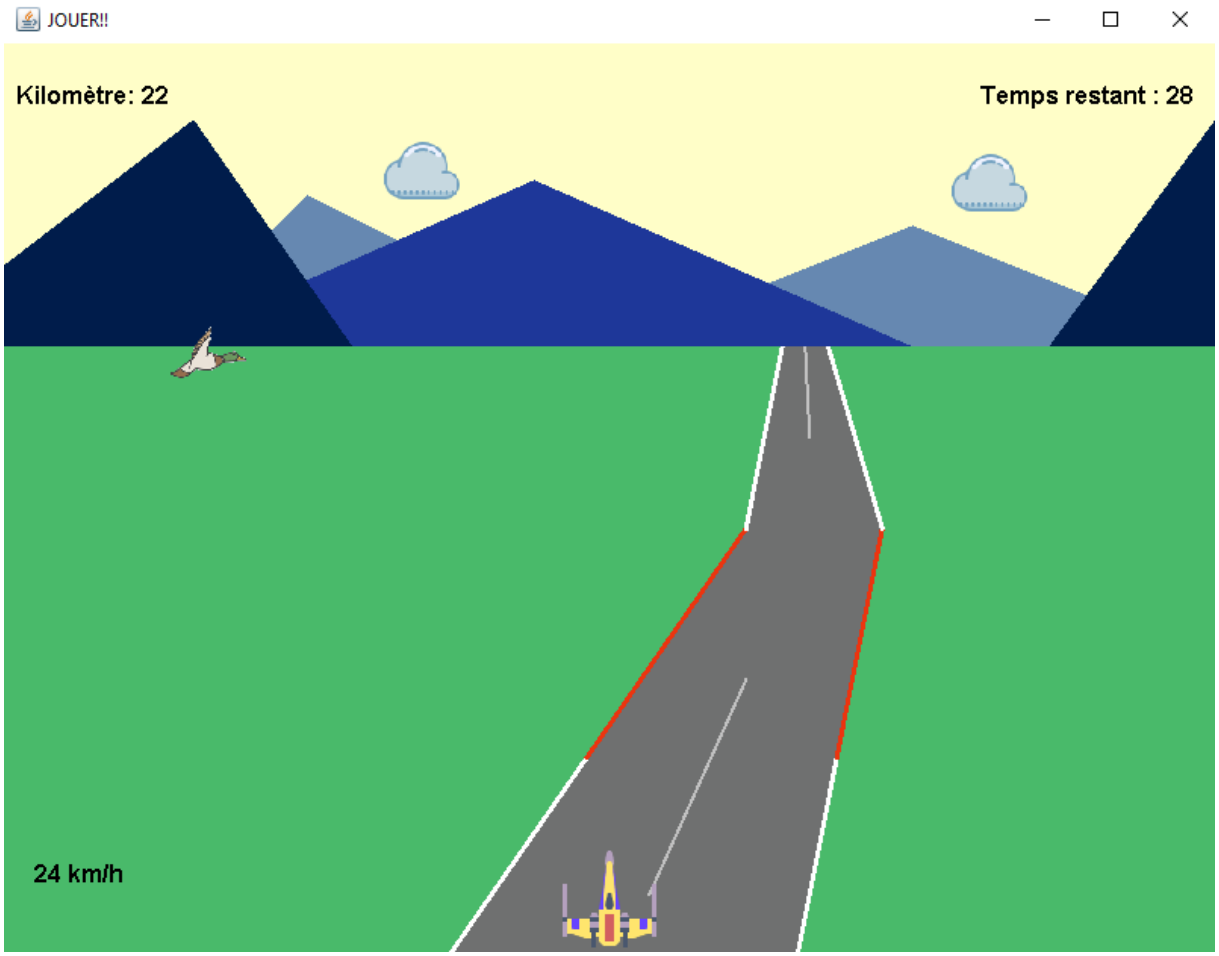
3) ***Le dessin du véhicule change en fonction des actions du joueur pour suggérer les mouvements à droite et à gauche.***

Dans classe **MOTO** il y a une méthode « setImage » qui permet de changer l'image correspondant au véhicule. Donc lorsque **CONTROLKEY** observe qu'il y a une touche «gauche » ou « droite », on change l'image de la moto en temps avec son déplacement.

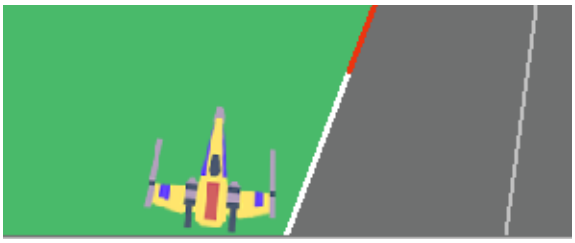
4) ***Le dessin de la piste se rétrécit au bout pour créer une sensation de profondeur.***

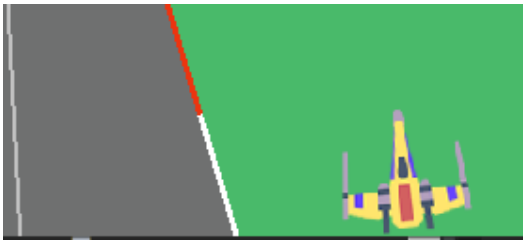
Nous réalisons cette fonctionnalité dans classe **AFFICHAGE**. Afin de créer l'effet de profondeur de la piste, nous avons constaté qu'à mesure que le champ de vision se rapproche, l'ordonnée de chaque point continue d'augmenter. Ces valeurs peuvent être utilisées pour calculer la largeur croissante de la trace. La méthode « paintParcours » calcule l'abscisse des points sur les côtés droit et gauche de la piste en ajoutant et en soustrayant la valeur calculée de l'abscisse d'origine en fonction de leur ordonnée. En conséquence, il semble que la route s'élargit progressivement dans l'interface. A la fin, nous avons utilisé « fillPolygon » et « drawLine » pour avoir une vue plus élégante.

Résultat

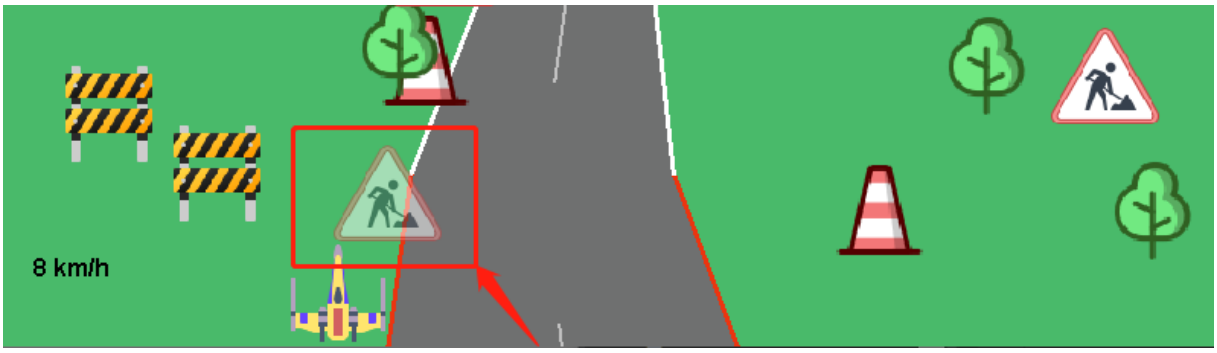


Au démarrage du jeu

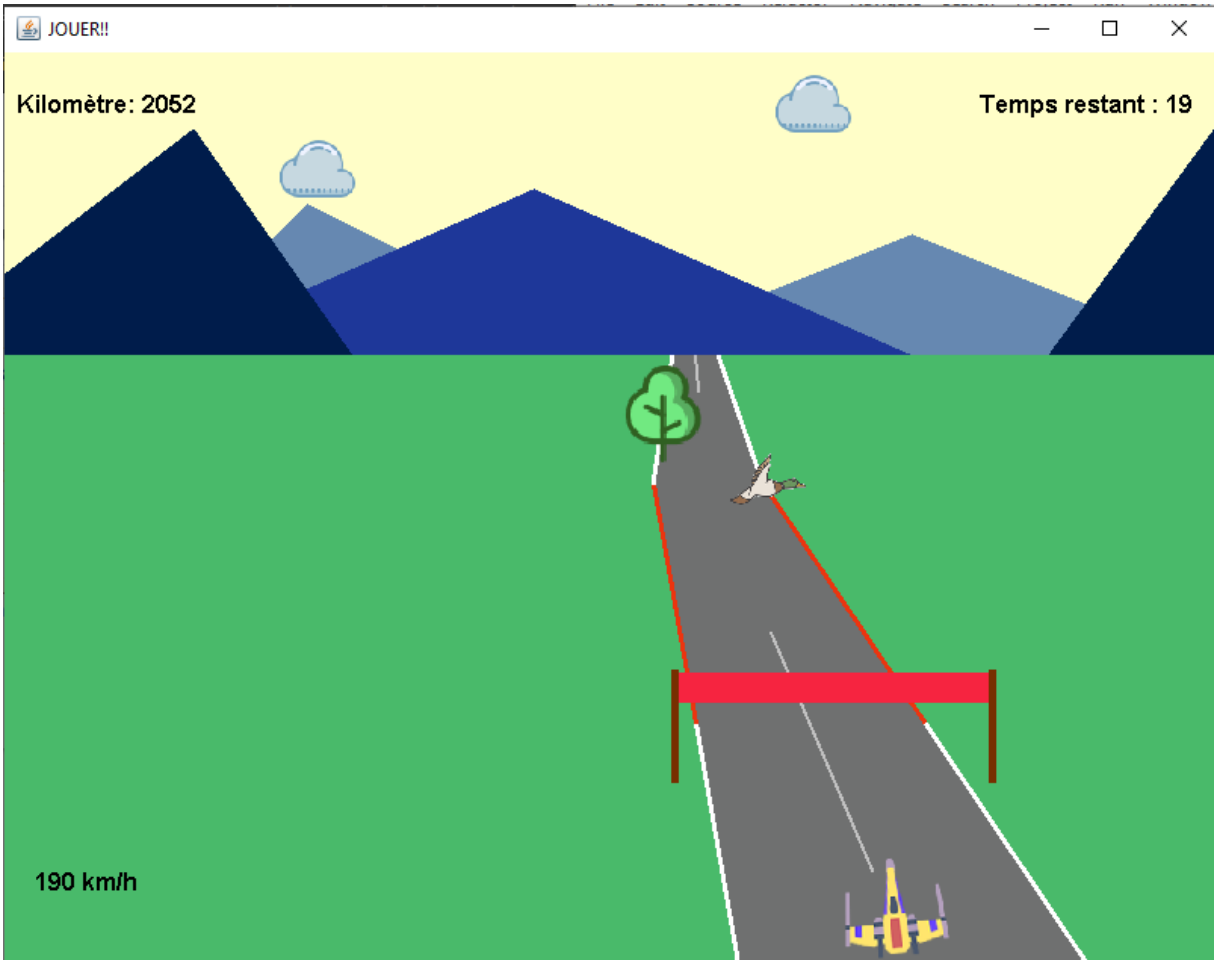




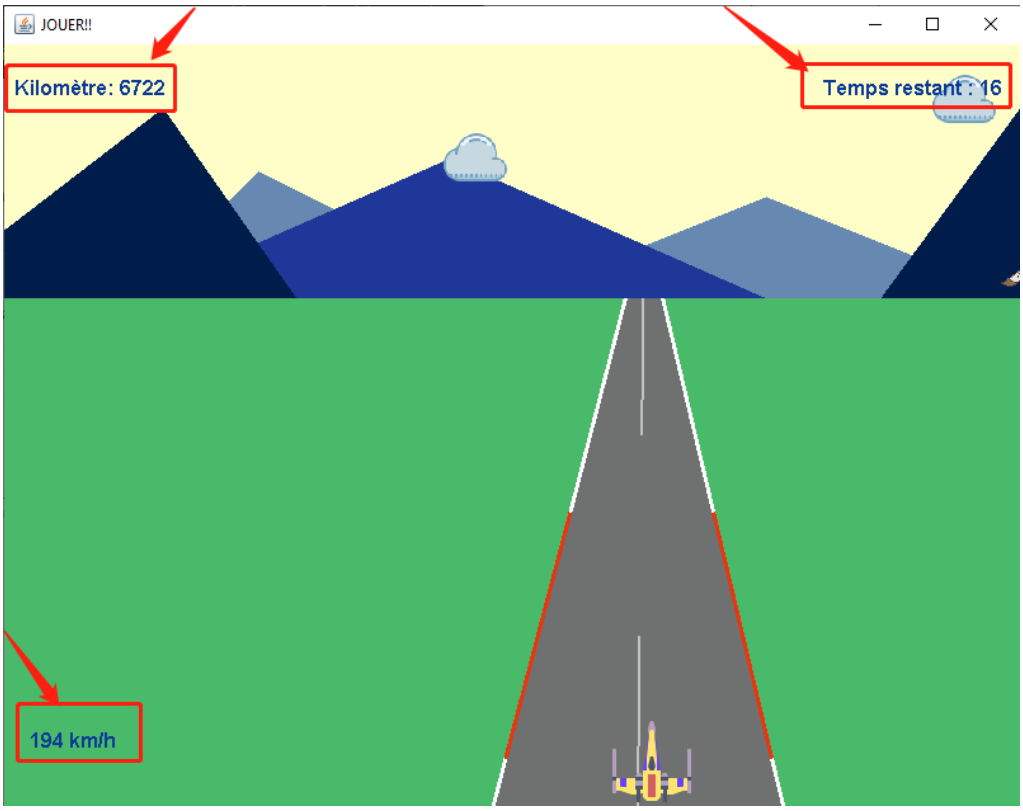
Le dessin du véhicule change en fonction des actions du joueur pour lui suggérer des mouvements à droite et à gauche



La moto est entrée en collision avec un obstacle généré aléatoirement

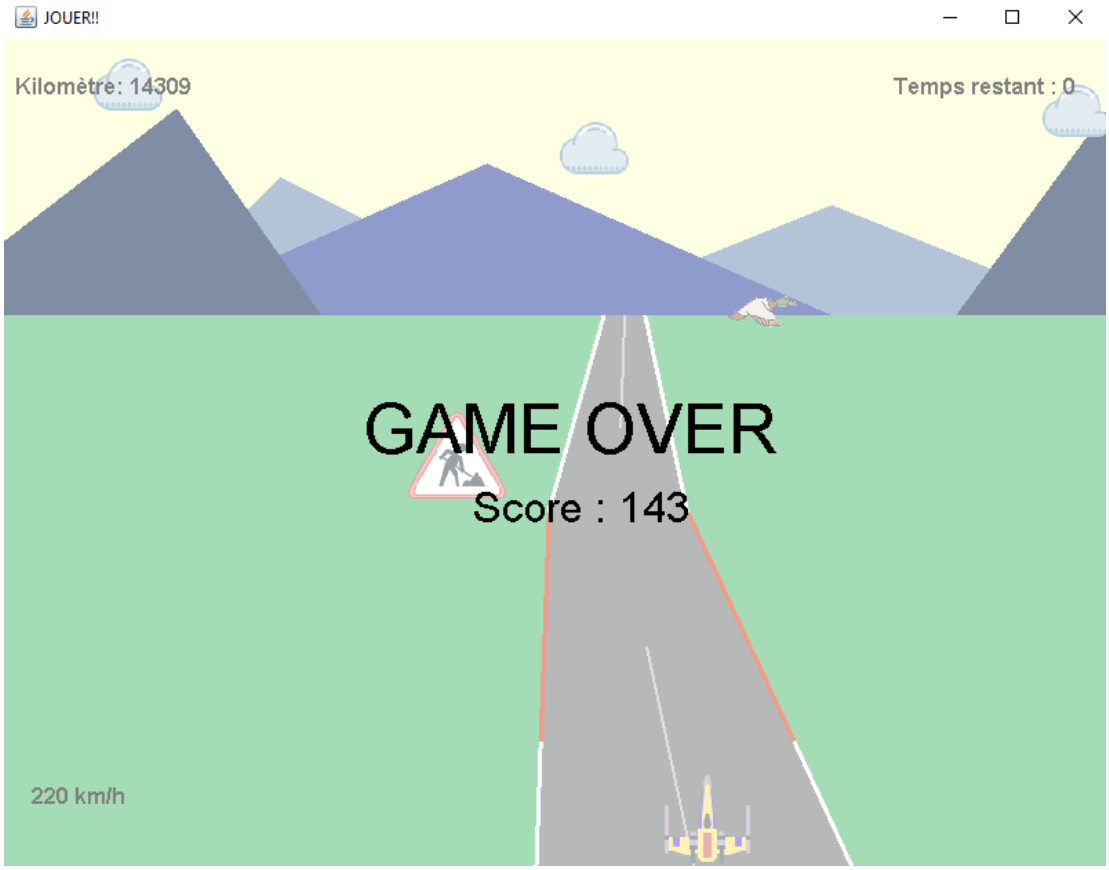


À chaque point de contrôle, un temps supplémentaire est alloué.



La moto accélère (le texte sur l'interface deviendra bleu)

Le menu qui apparaît lorsque le jeu est en pause



Le jeu se termine et le score s'affiche

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 ». Cliquez sur la fenêtre pour faire monter l'ovale.
- **Mode d'emploi (cas .jar exécutable)** : double-cliquez sur l'icône du fichier .jar. Cliquez sur la fenêtre pour faire monter l'ovale.
- **Règles du jeu** :

Afin de faire bouger la moto, il suffit juste d'appuyer sur la flèche gauche de votre clavier, cela vous permettra de contrôler le véhicule et de le bouger vers le gauche. De même pour son déplacement vers la droite si vous appuyez sur la flèche droite de votre clavier. Appuyez sur le bouton « haut » si vous voulez accélérer la moto. Vous pouvez aussi appuyer sur la touche « haut » en même temps que droite ou gauche pour vous déplacer et accélérer en même temps. Il y a plusieurs types d'obstacles et il faut que vous les évitiez. Si vous entrez en collision avec l'un d'eux, votre véhicule ralentira et risque de s'arrêter. Si vous voulez mettre en pause le jeu ou reprendre la partie après l'avoir mis en pause, appuyez sur la touche « echap ».

Documentation développeur

Dans le programme, nous avons un package par défaut et trois packages correspondent chacun à une partie du motif MVC. Dans le package appli, il y a seulement une classe **MAIN** pour déclencher le jeu.

Les principales constantes peuvent être ajustées pour changer l'affichage de l'interface graphique. Par exemple, dans la classe **AFFICHAGE**, vous pouvez modifier la taille de la fenêtre en modifiant la valeur de « LARG » et « HAUT » . De plus, le constante « horizon » correspond à la position de l'horizon fixé. Étant donné que le parcours est limité par l'horizon, lorsque vous changez la position de l'horizon, le début du parcours sera modifié relativement. Et puis, dans la classe **ETAT**, une constante *« vitesseMoto »* désigne le pixel déplacé de la moto et le décor lorsque l'utilisateur appuie. Vous pouvez également changer la valeur de *« wid »* , *« len »* pour modifier la taille dessinée de la **MOTO** , pareil dans la classe **OBSATCLE**.

Si vous souhaitez augmenter la difficulté du jeu, vous pouvez diminuer la valeur de « TauxObstacles » de classe **PARCOURS** donc il y aura plus d'obstacles générés au cours de jeu. Ou bien, vous pouvez réduire le temps crédit initial et le temps supplémentaire alloué à chaque fois que le chariot passe par le point de contrôle, c'est-à-dire diminue la valeur de la variable « creditTemps » et « ajoutTemps » de classe **MOTO.**

Conclusion et perspectives

I. DIFFICULTES RENCONTREES

Il n'a pas été facile pour nous de travailler en binôme à distance, en plus un de nous a eu un problème technique avec son ordinateur. Donc nous avons été obligées de travailler sur le même appareil au début du projet. Durant le projet, nous avancions lentement mais nous avons fait notre mieux pour y arriver.

Au début, nous avons pris du temps à bien comprendre le sujet et à analyser le squelette du projet. Au niveau de la programmation, nous avons eu une petite difficulté sur l'implémentation de *« KeyListener ».* Et puis, nous avons coincé sur l'animation de la piste pour qu'elle soit limitée par l'horizon. Ensuite, nous avons réfléchi plusieurs façons à bien implémenter le mécanisme de vitesse et d'accélération. A cause de cela, nous avons pris beaucoup de temps pour tester le mécanisme et la détection de collision.

Par ailleurs, nous avons au début réussi à implémenter la méthode *« setCurve »* de la classe QuadCurve2D pour dessiner une piste plus élégante avec des courbes de Brézier. Mais étant donné que le calcul des coordonnées et l'affichage pour créer une sensation de profondeur en courbe est plus compliquée qu'une ligne droite, et que nous n'avons pas réussi à trouver une façon propre de remplir la courbe, nous avons finalement de dessiner notre piste avec la méthode « drawLine ».

II. ELEMENTS REUSSIS

Pour que le développement avance dans ce contexte compliqué, nous avons utilisé discord pour programmer ensemble en faisant le streaming au début du projet.

Au début, nous avons pu appliquer les connaissances que nous avions apprises durant les séances de tutoriel. La piste est représentée par deux lignes brisées et limitée par un horizon, elle s'avance automatiquement vers le bas de l'interface. Nous avons réussi à dessiner une piste qui se rétrécit au bout pour créer une sensation de profondeur. Ainsi, nous avons amélioré notre maîtrise de Java Swing.

Le véhicule peut se déplacer d'une valeur calculée en fonction de sa distance à la piste lorsqu'on appuie sur les touches, ainsi le décor bouge dans le sens contraire. Il y a aussi des obstacles apparaissant aléatoirement au fur et à mesure autour de la piste.

Nous avons également réussi à mettre en ouvre le mécanisme de points de contrôle, le mécanisme d'accélération de la moto par exemple. De plus, nous avons réussi à réaliser plusieurs fonctionnalités supplémentaires, un menu pour changer l'état du jeu, changement de l'image du véhicule lors de déplacements par exemple.

III. AMELIORATIONS POSSIBLES

Pour conclure, nous avons globalement réussi à aboutir aux résultats attendus. En effet, nous souhaitions réaliser un jeu vidéo des années 80 de type « course de voiture » en vue à la première personne (le véhicule est vu de derrière). Le jeu a besoin au moins des mécanismes d'interface interactive pour contrôler la moto, ainsi des mécanismes basés sur la vitesse et le

temps. Pour cela, nous avons réussi à implémenter les fonctionnalités les plus basiques.

On a consacré beaucoup de temps pour réaliser un jeu qui ressemble à un vrai jeu. Cependant, nous n'avons pas eu le temps d'apporter toutes les améliorations. Donc nous pouvons réaliser par exemple mémorisation des meilleurs scores ou bien, un mécanisme permettant le véhicule de monter en haut de l'écran etc.