

RAPPORT DE PROJET

Année 2020-2021

Sous la direction de

Thi Thuong Huyen Nguyen

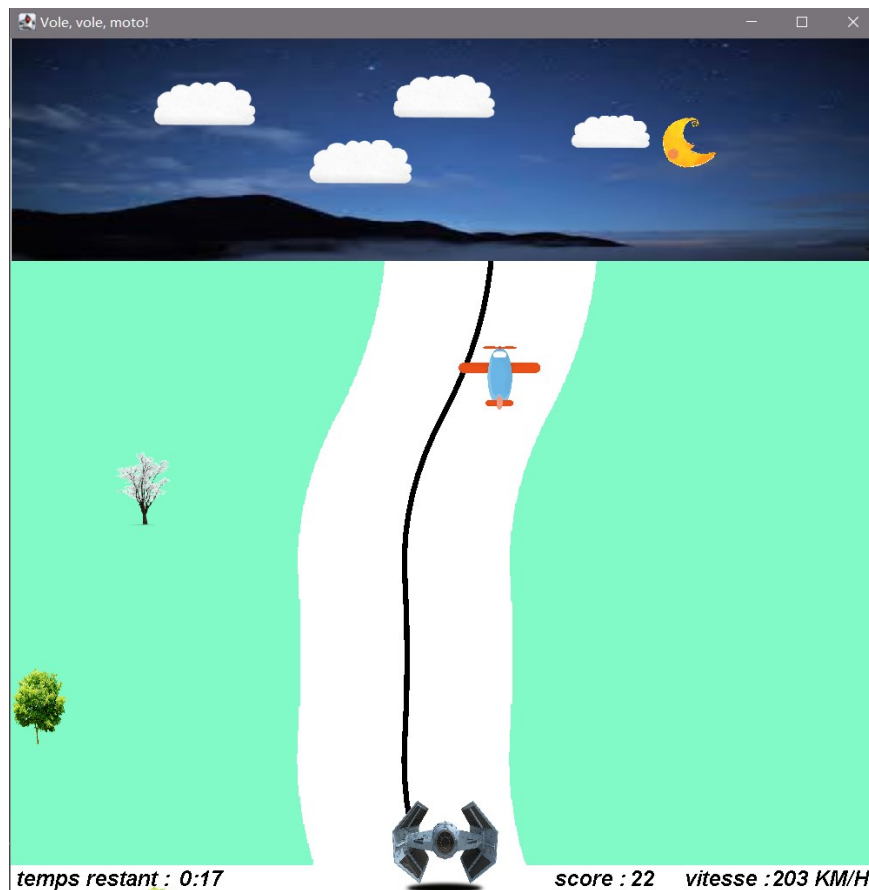
Fait par : Zhenhai XU

Yongjie LIU

License 3 Informatique

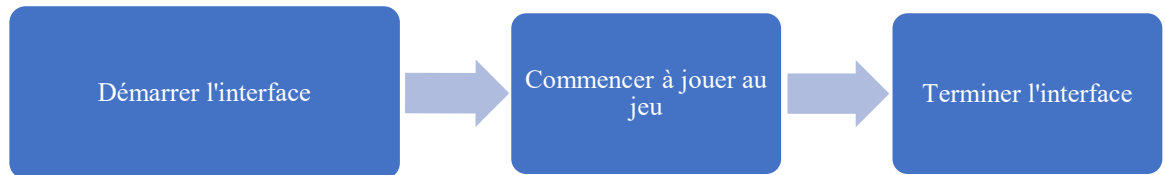
Introduction

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



Analyse globale

I. Le déroulement d'une partie



La première partie : Ouvrez le logiciel, il y a d'abord une interface de bienvenue, en bas de l'interface se trouve un bouton de démarrage, cliquez sur le bouton de 'Start' pour démarrer le jeu.

La Deuxième partie : Dans le jeu, le joueur utilise les touches fléchées pour jouer au jeu et doit éviter que le temps ou la vitesse restante du jeu ne soit nul, sinon il entrera dans l'interface de fin.

La fin de partie : Fin de l'interface affichera le score le plus élevé de l'histoire, le score obtenu par le joueur actuel et son classement.

II. Conception du véhicule du joueur

Nous définirons le véhicule du joueur dans le modèle. Utilisez les touches fléchées pour déplacer le véhicule, nous avons trois images qui reflètent les trois états de la moto, conduite normalement, conduite à gauche et conduite à droite.

III. Conception de l'horizon

L'horizon occupe quart de l'écran, en utilisant trois images à afficher (images d'affichage circulaire), et il y a des nuages en mouvement, la lune et le soleil.

IV. Conception de la route

La route dans l'interface doit être distinguée des côtés de la route et la courbure de la route doit être aléatoire. Pour que le véhicule du joueur avance

relativement, la route doit défiler vers le bas et doit être prolongée indéfiniment, de sorte que de nouveaux points de coordonnées doivent être générés pendant la partie. Les points de coordonnées utilisés doivent être supprimés à temps pour économiser la mémoire.

V. Décoration des deux côtés de la route

Générer plusieurs arbres des deux côtés de la route comme décorations. Le véhicule devra ralentir lorsqu'il frappe un arbre. Les points de coordonnées générés seront supprimés après avoir dépassé l'écran.

VI. Conception de l'obstacle

L'obstacle doit être sur la route et le véhicule ralentira lorsqu'il heurte l'obstacle

VII. Conception de l'adversaire

L'adversaire a sa propre vitesse, et le véhicule du joueur ralentira lorsqu'il heurtera l'adversaire

VIII. Gestion de la vitesse

La vitesse est déterminée par l'accélération. L'ampleur de l'accélération dépend de la distance par rapport au centre de la route (plus la distance est petite, plus l'accélération est grande). Il devrait y avoir une limite de vitesse maximale.

IX. Affichage des données

Les joueurs ont besoin d'un retour d'informations en temps réel. Le temps restant, le score actuel et la vitesse doivent être affichés à l'écran.

X. Les checkpoints

Augmentez le temps restant lorsque le véhicule passe le point de contrôle, permettant au joueur d'aller plus loin.

XI. L'interface de fin de jeu

Lorsque le jeu est terminé, le processus doit se terminer et le score du joueur doit être affiché.

Plan de développement

Liste des taches :

Binôme en **kaki**.
Zhenhai XU en **rouge**.
Yongjie LIU en **bleu**.

L'importance de chaque fonctionnalité est définie par la figure suivante :

Classement	A	B	C	S
signification	Les mécanismes clés du jeu	Fonctionnalités supplémentaires importantes du jeu	Améliorez l'achèvement du jeu (Une partie de l'optimisation est difficile)	Le rapport et debug sont très important(très difficile)

Figure 1 Le classement difficulté

Le niveau d'importance ne signifie pas sa difficulté.
La progression réelle du travail est affichée dans le diagramme de Gantt.

Séance 4 : Mise en place du projet

Fonctionnalité à développer	Importance	Durée estimée	Responsable
Analyse du problème.	A	1 jour	Binôme
Rédaction du plan de développement	A	1 jour	Binôme
Conception, développement et test d'une fenêtre avec la moto et la piste.	A	1 jour	Zhenhai XU
Conception, développement et test du mécanisme de déplacement de la moto	A	2 jours	Zhenhai XU
Rapport du projet v0.1	S	1 jour	Zhenhai XU

Figure 2 : cahier des charges Séance 4

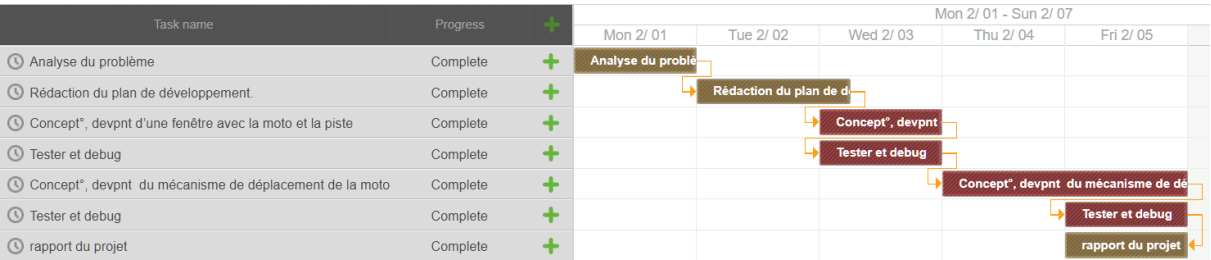


Figure 3 : diagramme de Gantt Séance 4

Séance 5 et 6 : moteur de jeu et première version

Fonctionnalité à développer	Importance	Durée estimée	Responsable
Conception, développement et test de l'animation de la piste à vitesse constante	A	2 jours	Yongjie LIU
Conception, développement et test de la génération des décors de fond	B	1 jour	Yongjie LIU
Conception, développement et test de la gestion du clavier en continu	A	2 jours	Zhenhai XU
Conception, développement et test de la Mvt du décor de fond selon les touches du clavier	B	1 jour	Zhenhai XU
Rapport du projet v0.2	S	1 jour	Binôme

Figure 4 : cahier des charges Séance 5

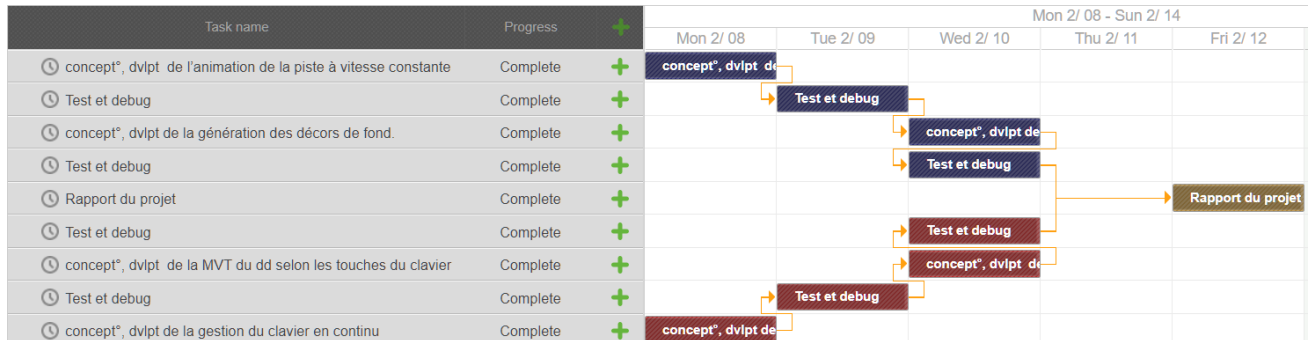


Figure 5 : diagramme de Gantt Séance 5

Fonctionnalité à développer	Importance	Durée estimée	Responsable
Consulter les commentaires de l'enseignant et prévoyez de modifier les codes	A	1 jour	Binôme
Conception, développement et test de la Mouvements des arbres	B	2 jours	Yongjie LIU
Conception, développement et test de la génération des arbres au bord de la piste	A	2 jours	Yongjie LIU
Réécrire une partie du code et supprimer le code répété à l'aide de l'héritage	A	2 jours	Zhenhai XU
Conception, développement et test du décompte des kilomètres et l'affichage	B	2 jours	Zhenhai XU
Amélioration la gestion du clavier en continu	A	1 jours	Zhenhai XU
Rapport du projet v0.3	S	2 jours	Binôme

Figure 6 : cahier des charges Séance 6

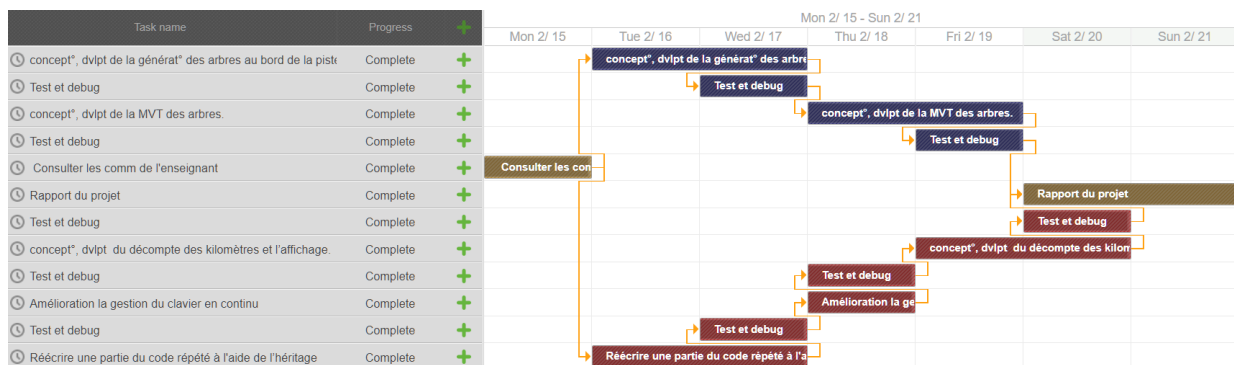


Figure 7 : diagramme de Gantt Séance 6

Séance 7 et 8 : mécanique du jeu

Fonctionnalité à développer	Importance	Durée estimée	Responsable
Ajout d'un décompte de temps et détermination de la fin de partie	A	1 jour	Yongjie LIU
Conception, développement et test ajout d'adversaires mouvement	B	2 jours	Yongjie LIU
Conception, développement et test ajout d'obstacles mouvement	B	2 jours	Yongjie LIU
Conception, développement et test gestion de la vitesse selon le point central de la piste.	A	2 jours	Zhenhai XU
Conception, développement et test Transformer la piste en une courbe de Bézier	C	2.5 jours	Zhenhai XU
Ajout d'un écran d'accueil	C	0.5 jours	Zhenhai XU
Ajout de la musique de fond	C	0.5 jour	Zhenhai XU
Ajout Écran de fin et affichage du score	A	1 jours	Zhenhai XU
Rapport du projet	S	1 jour	Binôme

Figure 8 : cahier des charges Séance 7

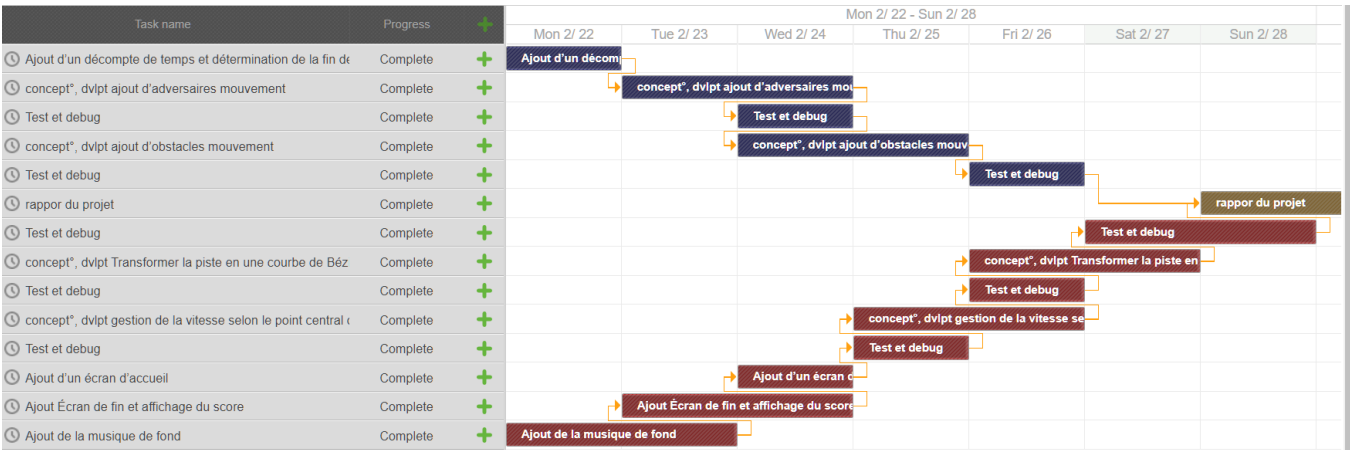


Figure 9 : diagramme de Gantt Séance 7

Fonctionnalité à développer	Importance	Durée estimée	Responsable
Conception, développement et test ajout de points de contrôles	A	2 jours	Yongjie LIU
Mécanisme détection de collisions qui ralentissent le véhicule (l'arbre, l'obstacle, et l'adversaires)	B	2 jours	Yongjie LIU
Mécanisme de récompense (les bonus score et temps)	B	2 jours	Yongjie LIU
Conception développement et test calculer l'abscisse de la courbe de Bézier (Utilisé pour remplir les deux côtés de la route et calculer la vitesse avec plus de précision)	C	4 jours	Zhenhai XU
Améliorer gestion de la vitesse selon l'abscisse de la courbe de Bézier	A	0.5 jours	Zhenhai XU
Conception développement et test dessiner les deux côtés de la route	B	1 jours	Zhenhai XU
Rapport du projet	S	1 jour	Binôme

Figure 10 : cahier des charges Séance 8

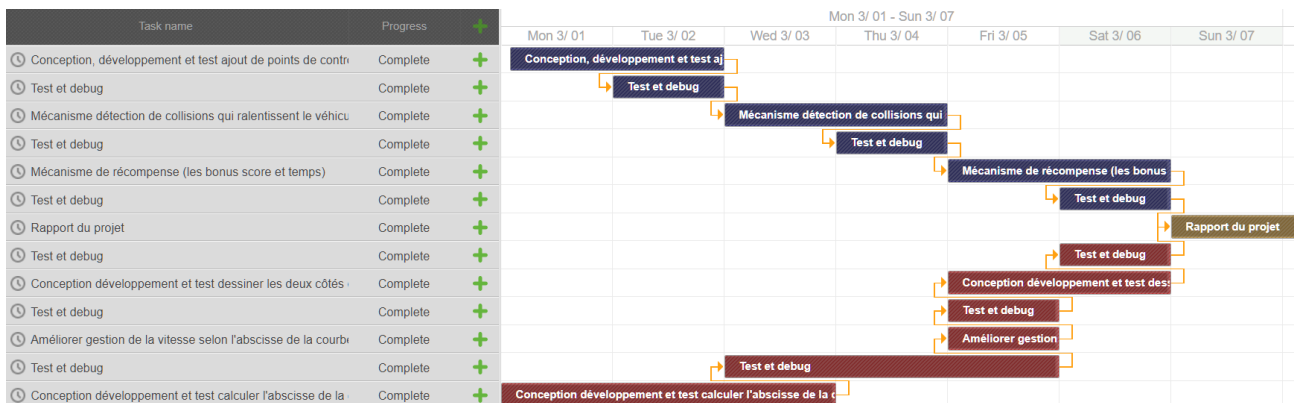


Figure 11 : diagramme de Gantt Séance 8

Séance 9 et 10 : Finalisation du projet

Fonctionnalité à développer	Importance	Durée estimée	Responsable
Création d'une sensation de profondeur	C	1 jours	Yongjie LIU
Mécanisme détection de collisions qui ralentissent le véhicule (l'arbre, l'obstacle, et l'adversaires)	B	3 jours	Yongjie LIU
La moto change l'image en fonction des touches du clavier	C	1 jour	Zhenhai XU
Stockez les données du jeu et affichez le classement à la fin	B	3 jours	Zhenhai XU
Aspects graphiques plus soignés	C	0.5 jour	Zhenhai XU
Tester et debug	A	1 jour	Binôme
Rapport du projet	S	2 jour	Binôme

Figure 12 : cahier des charges Séance 9

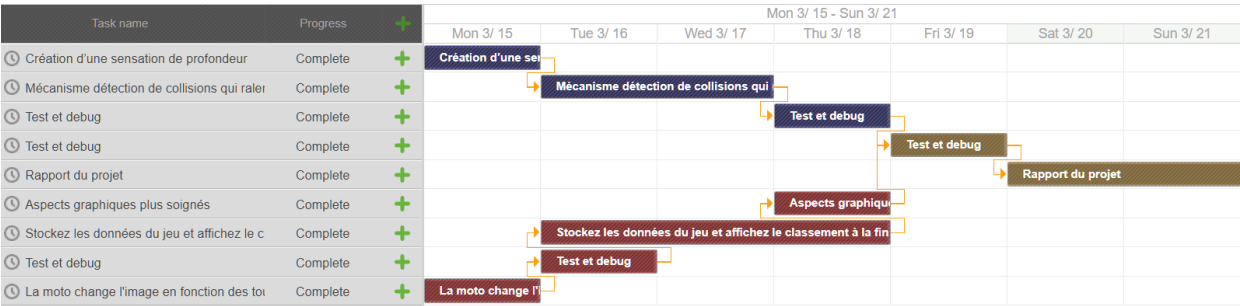


Figure 13 : diagramme de Gantt Séance 9

Fonctionnalité à développer	Importance	Durée estimée	Responsable
Tracez une courbe de Bézier au milieu de la route	C	1 jour	Zhenhai XU
Rapport du projet	S	2 jour	Binôme
Test et debug	S	3 jour	Binôme

Figure 14 : cahier des charges Séance 10

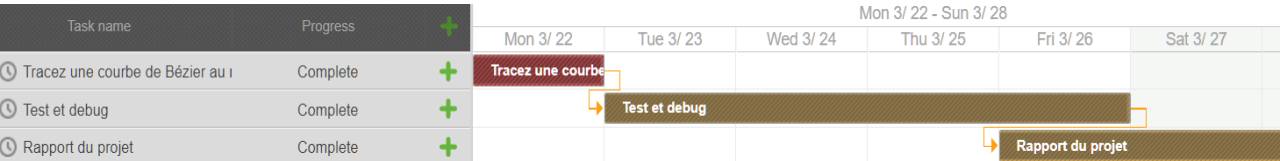


Figure 15 : diagramme de Gantt Séance 10

Conception Générale.

Dans ce projet, nous utilisons la pensée MVC et la divisons en trois parties, à savoir la Vue du Modèle et le Contrôle.

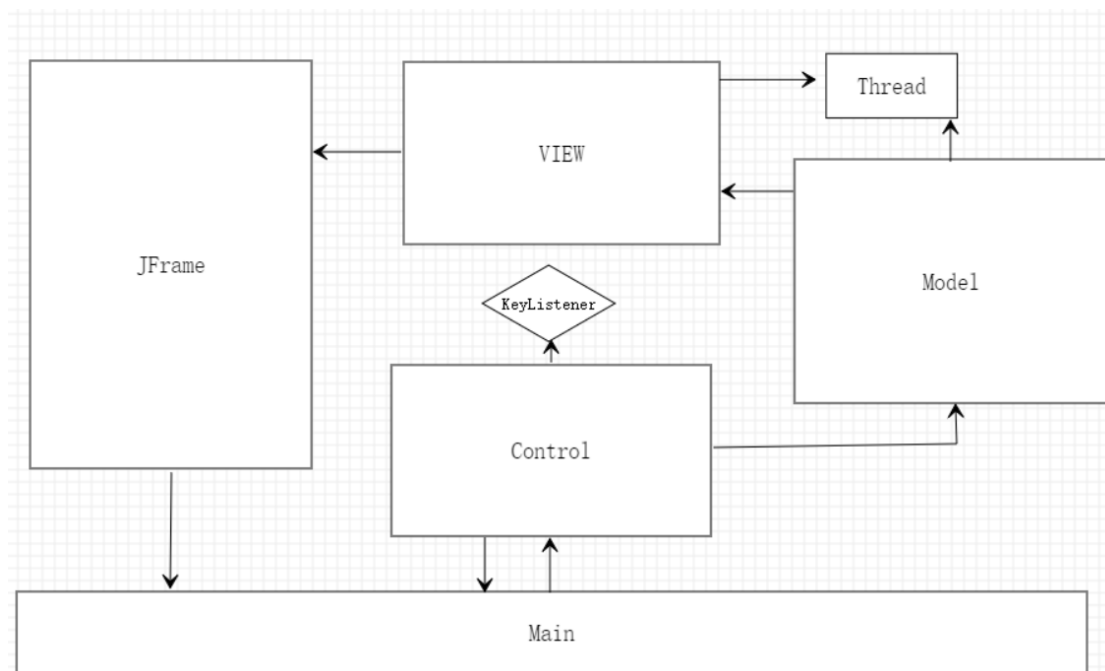


Figure 16 Introduction de la composition du projet

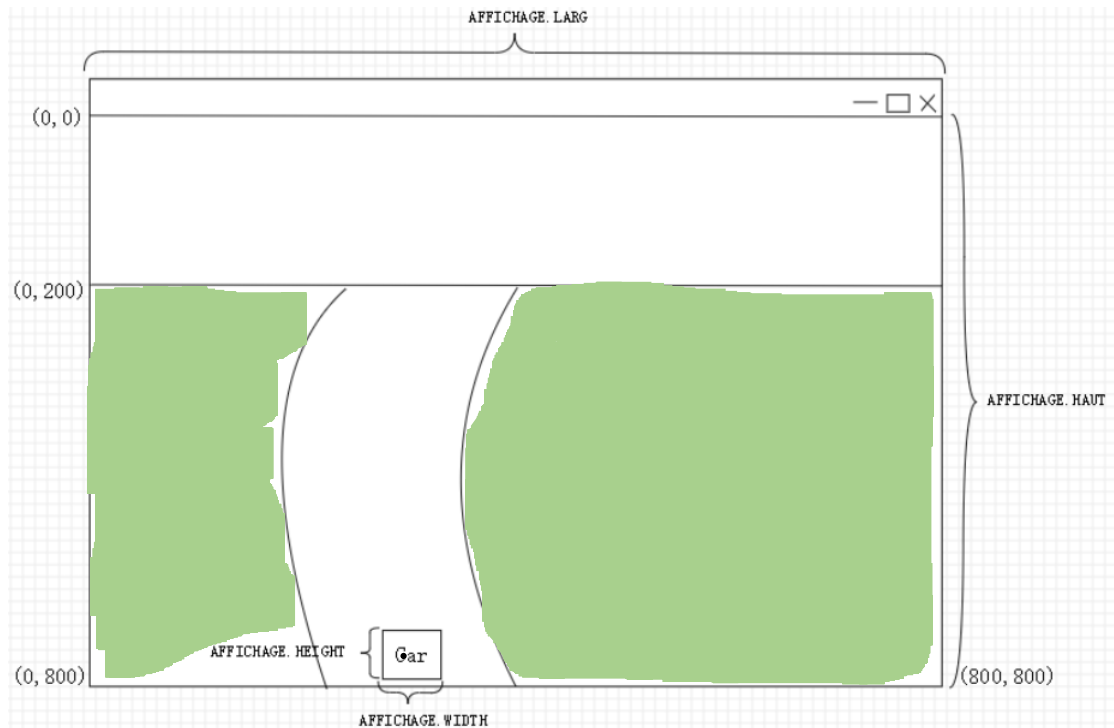


Figure 17 Introduction à l'interface graphique

Le milieu de la route est l'endroit où l'accélération est la plus élevée (les coordonnées de ce point sont calculées). S'éloigner du point central (se déplacer vers le haut et la gauche et la droite) réduira l'accélération. Lorsque l'accélération est négative, le véhicule ralentira vers le bas. Les côtés gauche et droit de la route sont dessinés et remplis avec l'abscisse du point central -100 et +100 pour montrer la voie blanche au milieu

Conception détaillée

I. Interaction entre les modules

- Le package Modèle est responsable des données et de l'état de l'arrière-plan, des véhicules et des voies etc... La modification des données dans le modèle peut changer les états de fonctionnement du jeu.
- Le package Vue est responsable du dessin, c'est-à-dire en acceptant les données dans d'autres packages, en dessinant la moto, l'arrière-plan et la voie, ainsi que le kilométrage et le temps restant affichés à l'écran.
- Le package de Contrôle met à jour les données du package vue en modifiant l'état de l'objet et réalise une animation à travers plusieurs peintures.

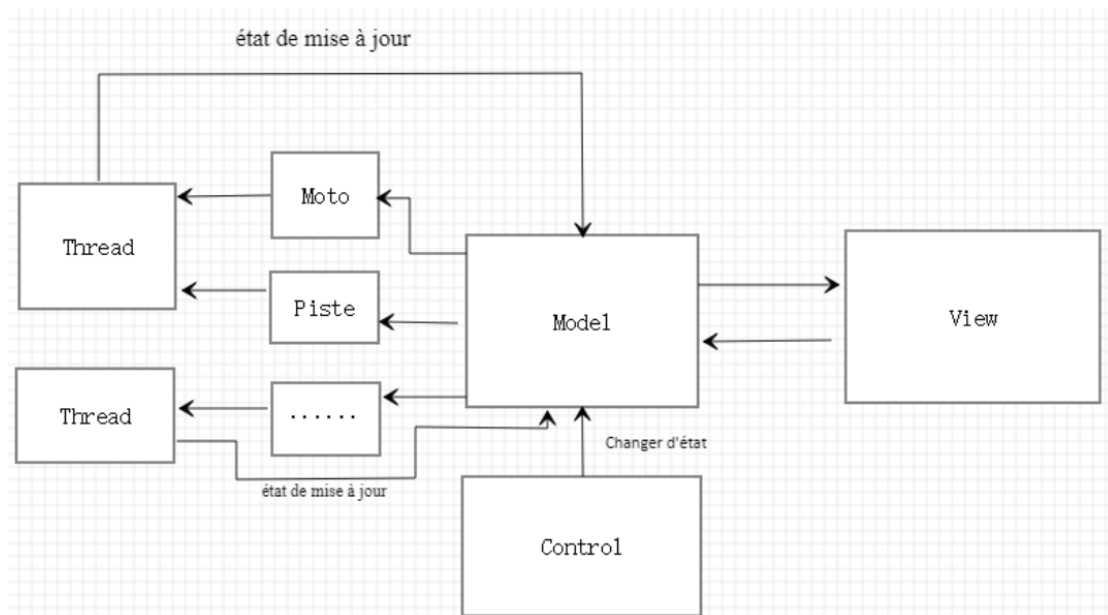


Figure 18 Diagramme d'interaction entre les modèles

II. Diagramme de classe du modèle

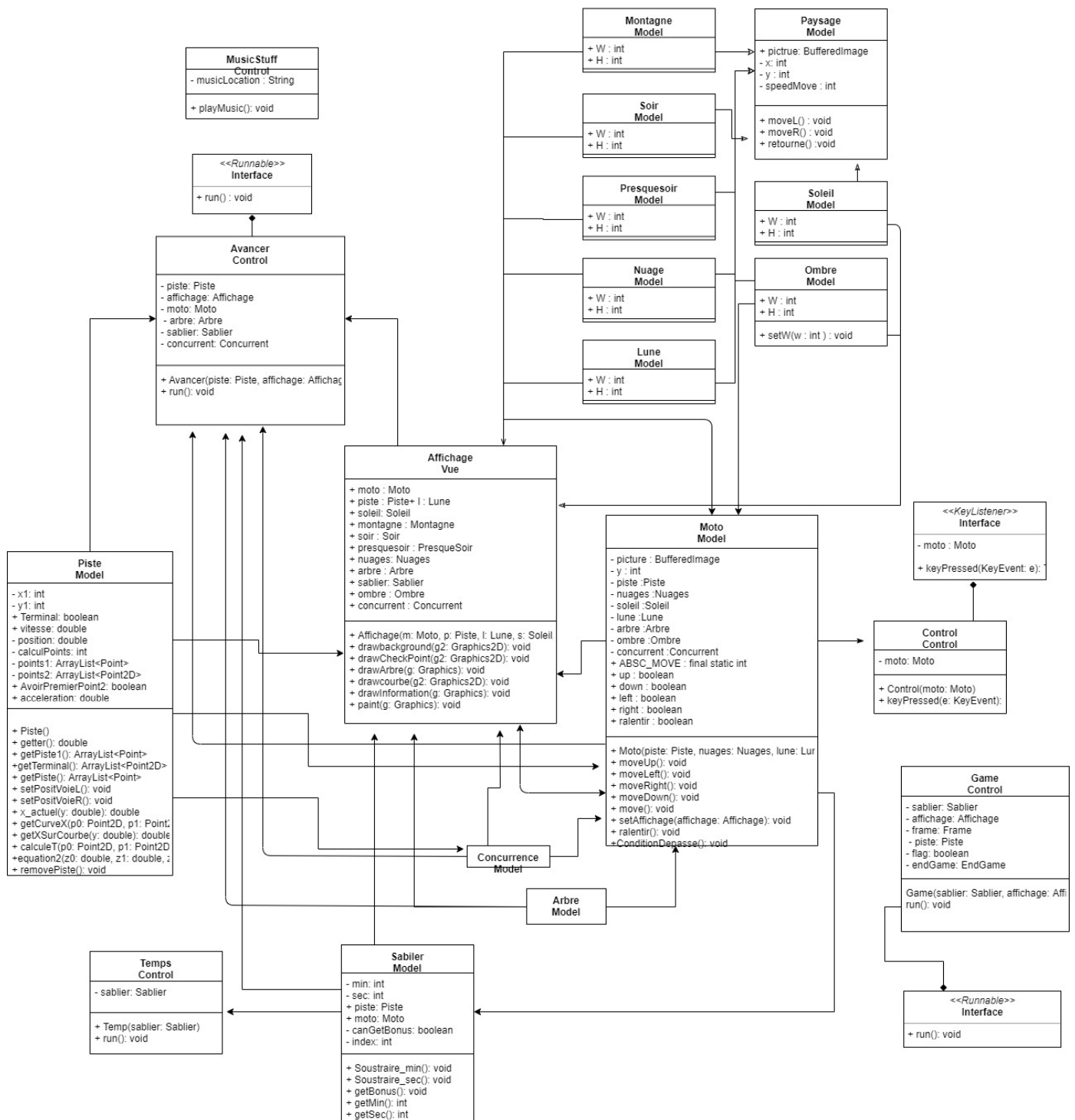


Figure 19 Diagramme de modèle

Dans cette diagramme, nous pouvons voir l'interaction entre chaque classe. De manière générale, les données sont transférées entre les modèles puis transmises à Vue. Les données et les images sont constamment mises à jour via des threads. Le contrôle est responsable de l'interaction entre le modèle connecté et le lecteur.

III. Conception du véhicule du joueur et Ombre

Classe utilisé :

```
> Moto.java  
> Ombre.java
```

Attribut import : Utilisé pour détecter l'état du véhicule.

```
public boolean up = false;  
public boolean down = false;  
public boolean left = false;  
public boolean right = false;  
public boolean ralentir=false;;
```

Dans cette classe, lorsque le véhicule reçoit une instruction de mouvement du joueur, il fera bouger les autres contenus sur l'écran l'un par rapport à l'autre sauf pour que le véhicule obtienne l'effet visuel du véhicule en mouvement.

Appuyez sur les touches gauche et droite pour changer l'image du véhicule, comme indiqué ci-dessous :



Figure 20 motoL

Figure 21 moto

Figure 22 motoR

```

public void moveLeft() throws IOException {
    if(left && this.absc > 0){
        try{
            if(right !=true)
                this.picture = ImageIO.read(new File("src/png/motoL.png"));
        } catch (IOException e) {
            e.printStackTrace();
        }
        absc -= ABSC_MOVE;
        nuages.moveR();
        soleil.moveR();
        lune.moveR();
        arbre.getArbrelisteR();
        piste.setPositVoieR();
        concurrent.getConcurrentlisteR();
        ombre.moveL();
    }
}

```

Le mouvement du véhicule ne dépasse pas la portée de l'écran.

Par exemple, Lorsque la voiture ne dépasse pas la portée de l'écran, après avoir appuyé sur le bouton gauche, tous les objets se déplaceront vers la droite et une nouvelle image est chargée en même temps (Figure 20 motoL), mais l'ombre se déplace de manière synchrone avec le véhicule, de sorte que l'ombre se déplace également vers la gauche.

```

public void moveUp() {
    if(up && this.y>=600) {
        this.y-=5;
        ombre.setW(-4);
        ombre.x -= 2;
        concurrent.fix=true;
    }
}

```

Lorsque le chariot s'élève vers le ciel, l'ombre doit être réduite et sa position ajustée.

```

public void move() throws IOException {
    moveUp();
    moveDown();
    moveLeft();
    moveRight();
    if(y<700&& up != true){
        y+=5;
        ombre.setW(4);
        ombre.x += 2;
    }
    if( left != true && right !=true){
        try {
            this.picture = ImageIO.read(new File("src/png/moto.png"));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
}

```

Enfin, nous détectons si le bouton est enfoncé, et si le bouton haut est relâché, le véhicule doit automatiquement descendre et ajuster l'ombre en même temps. Si vous relâchez les boutons gauche et droit, l'image du véhicule devient une image normale (Figure 21 moto).

```

public void ralentir(){
    for(int i=0;i<arbre.getArbresList().size();i++) {
        Arbre arbre= this.arbre.getArbresList().get(i);
        if(Math.abs(absc - arbre.x) <= arbre.W && Math.abs(y - arbre.y) <= arbre.H){
            ralentir=true;
        }
    }
}

```

Vérifiez si le véhicule heurte un arbre (utilisez la boucle for pour détecter tous les arbres dans ArbresList), s'il heurte, utilisez **void ralentir ()** pour rendre l'attribut **ralentir = vrai**, ce qui ralentira le véhicule.


```

public void ConditionDepasse(){
    if(Math.abs(getAbsc()-concurrent.getConcurrentList().get(0).x)<=85
        &&getY()-(<concurrent.getConcurrentList().get(0).y)<=80 &&getY()-(<concurrent.getConcurrentList().get(0).y)>=0){
        int x1=concurrent.getConcurrentList().get(0).x;
        concurrent.fix=false;
        int finalX = x1;
        concurrent.getConcurrentList().forEach(p->p.x= finalX);
        concurrent.getConcurrentList().forEach(p->p.y=concurrent.getConcurrentList().get(0).y);
        piste.acceleration= -1;
    }
    if(Math.abs(getAbsc()-concurrent.getConcurrentList().get(0).x)>=80){
        concurrent.fix=true;
    }
}
}


```

Détectez si le véhicule touche l'adversaire. S'il touche l'adversaire, l'accélération du véhicule deviendra un nombre négatif. Si le véhicule du joueur dépasse l'adversaire, l'accélération du véhicule reviendra à l'accélération d'origine. **Concurrent.fix** indique s'il y a contact avec le véhicule (vrai signifie il n'a pas de contact).


IV. Conception de l'horizon


Classe utilisé :


La classe mère du paysage à l'horizon


>  Paysage.java


Les classe suivants implémentent leurs fonctionnalité en héritant de Paysage:

>  Lune.java

>  Montagne.java

>  Nuages.java

>  Soir.java

>  Soleil.java

```


if (lune.getY() >= 0) {
    g.drawImage(soir.getPicture(),soir.getAbsc(), soir.getY(), soir.W, soir.H, null);
    lune.run();
    g.drawImage(lune.getPicture(), lune.getAbsc(), lune.getY(), lune.W, lune.H, null);
    nuages.setXl();
} else if (soleil.getY() >= 0) {
    g.drawImage(montagne.getPicture(), montagne.getAbsc(), montagne.getY(), montagne.W, montagne.H, null);
    soleil.run();
    g.drawImage(soleil.getPicture(), soleil.getAbsc(), soleil.getY(), soleil.W, soleil.H, null);
    nuages.setXr();
} else {
    lune.retourne();
    soleil.retourne();
}
g.drawImage(nuages.getPicture(),nuages.getAbsc(),getY(),nuages.W,nuages.H,null);

```

La lune apparaît la nuit et le soleil apparaît pendant la journée. Les Nuages sont toujours existents. Avec le fonctionnement de Thread, la lune continuera à se lever, et quand elle atteindra le point le plus élevé, elle se lèvera. L'image est remplacée par Montagne (journée), et le soleil se lèvera comme la lune, réalisant l'alternance du jour et nuit.

V. Conception de la route

Classe utilisé :

>  `Piste.java`

Ajoutez plusieurs points dans le constructeur pour dessiner des routes. Un ensemble de points de coordonnées sera généré aléatoirement dans **getPiste1 ()** et ajouté à la ArrayList pour les prochaines routes de dessin. Dans cette fonction, nous générerons également une nouvelle ArrayList pour le checkpoint suivant.

À travers le Thread, nous faisons défiler la route vers le bas et obtenons des points en même temps.

x_actuel () est utilisé pour calculer l'abscisse au milieu de la route, mais il est calculé sur la base de la pensée de la poly ligne. Au stade précoce du développement du jeu, nous nous appuyons sur cette fonction pour juger de l'accélération.

Dans la dernière étape du développement du jeu, nous avons utilisé la courbe de Bézier pour dessiner la plate-forme pour montrer la route blanche entre nous. Il n'est pas facile de calculer l'abscisse de la courbe de Bézier.

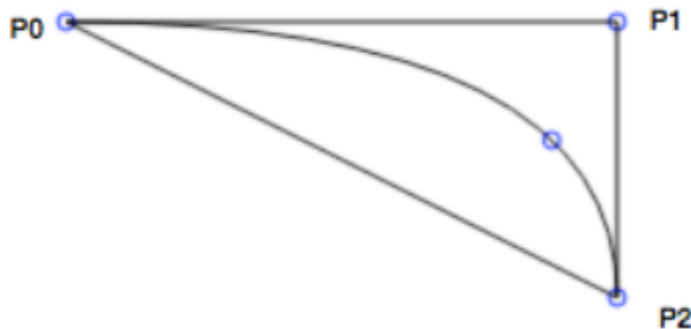


Figure 23 Diagramme schématisant la courbe de Bézier.

On sait que la formule de la courbe de Bézier est:

$$B(t) = P_0 + (P_1 - P_0)t = (1 - t)P_0 + tP_1, t \in [0, 1]$$

Figure 24 Formule de la courbe de Bézier.

Pour calculer les coordonnées sur la courbe, nous avons la formule suivante :

$$\begin{aligned}
 B_2(t) &= M_0 + (M_1 - M_0)t \\
 &= P_0 + (P_1 - P_0) + [P_1 + (P_2 - P_1)t - P_0 - (P_1 - P_0)t]t \\
 &= P_0 + 2(P_1 - P_0)t + (P_0 - 2P_1 + P_2)t^2 \\
 &= (1 - t)^2 P_0 + 2t(1 - t)P_1 + t^2 P_2
 \end{aligned}$$

Figure 25 Formule de la courbe de Bézier.

P est le point que nous voulons calculer (nous connaissons la valeur de l'ordonnée Y), nous utilisons la formule ci-dessus pour calculer d'abord la valeur t :

$$(P_2.x - P_0.x - 2 * P_1.x) t^2 + 2(x_{P1} - P_0.x) t + (P_0.x - P.x) = 0$$

Il est possible de résoudre l'équation pour obtenir deux valeurs :

$$a = (x_{P2} x_{P0} - 2 * x_{P1})$$

$$b = 2 * (x_{P1} - x_{P0})$$

$$c = (x_{P0} - x_P)$$

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Figure 26 Résolvez les deux solutions de l'équation.

S'il y a deux valeurs, si l'une d'elles n'est plus dans la plage de [0,1], l'autre solution est la bonne réponse. Si les deux valeurs sont dans la plage, nous devons utiliser la formule de la courbe de Bézier pour calculer les deux coordonnées Point (Xt1, p.y) (Xt2, p.y), puis juger lequel de ces deux points est proche de la valeur Y du point P. Plus la distance est petite, la solution correcte. Une fois la valeur t obtenue, on peut utiliser Besse La courbe Et calcule la valeur exacte de x.

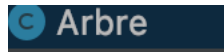
Cette valeur X peut être utilisée pour juger de l'accélération et tracer la ligne médiane de la route et des deux côtés de la route.

Les deux côtés de la route sont tracés de l'horizon au bas de l'écran. Nous avons toutes les valeurs Y (200 800), donc en dessinant plusieurs lignes droites, vous pouvez dessiner les deux côtés de la route avec les caractéristiques de la courbe de Bézier.

En même temps, nous pouvons tracer une ligne noire au milieu de la route pour rappeler au joueur où se trouve le point central de la route.

VI. Décoration des deux côtes d la route

Classe utilisé :



La classe Arbre qui est comme les décors, il apparait des deux côtes de la route. Afin de faire apparaitre des arbres des deux côtes de la route, on laisse l'arbre avoir deux

abscisses, soit x à droite `int xchoixDroit`, soit x à gauche

`int xchoixGauche`. On réalise cette fonctionnalité par

```
Random rd=new Random();
```

Pour création des arbres, on crée

```
List<Arbre> arbresList=new ArrayList<Arbre>();
```

Et puis on crée la fonction void getArbre() pour obtenir beaucoup d'arbre.

Quand une moto heurte un arbre, elle va décélérer Nous implémentons cette fonction dans la classe moto, via moto.ralentir pour réaliser cette fonction de décélération.

VII. Conception de l'obstacle

Classe utilisé :

Concurrent

La classe concurrente qui est les Obstacles « mobiles » sur la piste : il s'agit de concurrents qu'il faut dépasser.

La fonction **getConccurent ()** : ajouter les concurrents dans la liste<concurrent>.

La fonction **concurrentMove ()** : comme il y a un attribut booléen **fix**, si **fix** est vrai, ça veut dire que concurrent n'a pas encore touché de la moto, alors la vitesse du concurrent est le plus vite que la vitesse de la moto. Cette fonction peut fonctionner, sinon ça ne peut pas marche.

La fonction **empêcheMotodepasse ()** : c'est pour empêcher la moto qui dépasse le concurrent. Le code :

```
if(concurrentList.get(0).y>400 && fix) {  
    int x1 = (int) (Math.sin(Math.PI / 6 * index) * 50);  
    this.x = 400 + x1;  
    index++;  
    this.concurrentList.forEach(p -> p.x = x);  
}
```

Lorsque le concurrent s'approche de la moto, il commence à se déplacer selon une trajectoire sinusale. On définit **x=(int)(Math.sin(Math.PI/6*index) *50)**. Et puis **concurrentList.get(0).x=this.x**.

Comme on définit **concurrentList.size=1**, ça veut dire qu'il y a toujours un seul concurrent sur la piste en même temps, lorsque le concurrent est dépassé par la moto, on supprime ce concurrent et ajouter un nouveau concurrent.


Lorsque **fix =false**, ça veut dire le concurrent a touché la moto. C'est comme la figure :



. Figure 27 Schéma de principe de la collision entre le véhicule et l'adversaire.

VIII. Conception de l'adversaire

Classe utilisé :

>  **Concurrent.java**

La classe concurrent a de nombreux adversaires avec des images différentes. Ici, j'utilise des nombres aléatoires pour afficher différents concurrents. Un seul concurrent apparaît à l'écran à chaque fois. Ce n'est que lorsque la moto dépasse ce concurrent qu'il en générera un nouveau.

```
String RandomConcurrent="concurrent"+Randomindex;  
try {  
    concurents = ImageIO.read(new File("src/png/" + RandomConcurrent + ".png"));
```

IX. Gestion de la vitesse

Classe utilisée :

Piste.java

Nous savons que la formule pour calculer la vitesse est :

$$v = v_0 + at$$

Nous définissons t dans le thread, et la vitesse changera toutes les 0,1 s.

```
double t = 0.1;
piste.setVitesse(t);
```

La formule de vitesse est appliquée dans piste. SetVitesse ()

```
public void setVitesse(double time){
    if(this.vitesse<=0){
        gameOver=true;
    }
    if( this.acceleration <=0 ){
        this.vitesse += this.acceleration * time;
    }else if(this.vitesse <= this.VITESSELIMITE){
        this.vitesse += this.acceleration * time;}
}
```

Au début du jeu, le véhicule a une vitesse initiale v0, qui change avec le temps et l'accélération a. Cela diminuera l'accélération du véhicule lorsqu'il est éloigné de la voie et en contact avec des obstacles, des arbres et des adversaires.

```
public void setAcceleration(Moto moto) {
    double x = x_actuel(moto.getY());
    double c =Math.sqrt(Math.pow((moto.getAbsc()+Affichage.WIDTH/2-x),2)+Math.pow((moto.getY()+Affichage.HEIGHT-y1),2));
    if (c==0) {
        this.acceleration=100;
    }
    else if (c <=100) {
        this.acceleration=100/c;
    }else{
        this.acceleration=-c/100;
    }
}
```

Cette fonction permet de juger de la distance entre la moto et la piste. En calculant le point central de la route, plus on s'éloigne du point central, l'accélération diminuera (elle deviendra un nombre négatif), et plus la distance sera proche, l'accélération augmentera.

Le changement de vitesse consiste à changer l'attribut vitesse et attribut accélération dans la Classe Piste pour changer la vitesse.

La fonction `void changeVitesse(){vitesse -= 2*Avancer; }` est la valeur de change la vitesse lorsque la moto heurte un arbre.

X. Affichage des données

Afin de l'affichage le temps restant, on crée la classe : **Sablier.java** et un thread

```
Temp.java

public void Soustraire_min() {
    if (sec == 0) {
        min--;
        sec=59;
    }
}

public void Soustraire_sec() {
    sec--;
}

public void run() {
    while (true) {
        try {
            Thread.sleep(1000);
            sablier.Soustraire_min();
            sablier.Soustraire_sec();
        }
        catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

Les deux classes réalisent un compte à rebours.

Afin de pouvoir réaliser à chaque point de contrôle, un temps supplémentaire est alloué.

On utilise la fonction:

```
public void getBonus() {
    int bonus = 30;
    if (piste.AvoirPremierPoint2&&
        Math.abs(piste.getTerminal().get(piste.getTerminal().size()-1).getY()- 800) <= 20 && canGetBonus) {
        if (sec <= 59-bonus) {
            sec += bonus;
            if (bonus>=5)
                bonus--;
        } else {
            sec = sec + bonus - 59;
            min += 1;
            if (bonus>=5)
                bonus--;
        }
        canGetBonus =false;
        //System.out.print(index+" ");
    }
    index++;
    if(index>=80){
        canGetBonus =true;
        index=0;
    }
}
```

Ici, on importe la classe piste, Lorsque le véhicule passe le point final, il ajoutera du temps pendant 30 secondes (à mesure que le temps change, la récompense diminuera). Afin d'obtenir l'augmentation normale du temps, nous utilisons un attribut booléen canGetBonus, cet attribut deviendra faux à chaque fois que l'heure sera ajoutée, puis la laissera devenir vraie après un certain temps.

XI. Les checkpoints

Classe utilisé :

Piste.java

On réalisé la fonctionnalité checkpoint dans cette classe, on définit une nouvelle liste :

```
ArrayList<Point2D> points2=new ArrayList<>();//pour construire le terminue
```

Nous générons de nouveaux points dans cette fonction et les ajoutons à points2, sur une variable définie calculPoints = 0, et Terminal = false, puis nous laissons calculPoints augmenter continuellement dans la fonction, et laissons Terminal = true lorsque les conditions sont remplies, si Terminal = true, nous pouvons ajouter un nouveau point dans Points2, puis dessiner le point de contrôle via l'affichage.


```

public ArrayList<Point> getPiste1(){
    this.points1.forEach(p->p.y+=vitesse);
    this.points2.forEach(p->p.setLocation(p.getX(),p.getY()+vitesse));
    if(calculPoints%15<=1){
        Terminal=true;
    }
    if(points1.get(points1.size()-1).y>=Affichage.HAUT/6){
        int x=new Random().nextInt(100)+Affichage.LARG/2 - 50;
        int y= points1.get(points1.size()-1).y-150;
        this.points1.add(new Point(x,y));
        calculPoints++;
        if(Terminal) {
            double tour_x = points1.get(points1.size() - 1).x;
            double tour_y = points1.get(points1.size() - 1).y;
            this.points2.add(new Point2D.Double(tour_x, tour_y));
            AvoirPremierPoint2=true;
            Terminal=false;
        }
    }
    if(points1.get(1).y>= Affichage.HAUT){
        removePiste();
    }
    return this.points1;
}

```

XII. L'interface de début de jeu

Classe utilisé :

 StartGame.java

Ajoutez des images et des boutons à la planche à JFrame. Lorsque vous appuyez sur le bouton, le jeu démarre. J'utilise la boucle while pour détecter l'état de début du jeu dans Main (). Lorsque le jeu peut démarrer, supprimez l'écran actuel et ajoutez Affichage à la JFrame. L'état de la souris est focalisé sur le bouton, nous devons nous recentrer sur l'écran, sinon le détecteur de clavier ne fonctionnera pas correctement.


```

frame.remove(gameStart);
frame.requestFocus();

```

XIII. L'interface de fin de jeu

Classe utilisé :


>  [EndGame.java](#)

Lorsque le jeu est terminé, nous supprimons le contenu de la JFrame et ajoutons un nouvel écran. L'écran doit afficher le score du joueur. Le score sera également sauvegardé dans le fichier et comparé aux autres scores. S'il s'agit du score le plus élevé, enregistrez ce score au début du fichier et affichez le classement.

```
Thread.sleep(100);|
piste.setterPosition();
if(sablier.getMin()==0&& sablier.getSec()==0 || piste.gameOver== true){
    frame.remove(affichage);
    endGame.highScore();
    JLabel label = endGame.newLable();
    frame.add(label);
    frame.repaint();
}
```

XIV. Musique ajouter

Classe utilisé :

>  [MusicStuff.java](#)

Nous définissons le chemin de stockage de la musique en classe, et jouons de la musique en boucle à travers le Thread.

RESULTATS

Affichage du jeu au démarrage



Figure 28 Interface de démarrage

Interface de démarrage, avec titre et bouton de démarrage. Cliquez sur Start Game pour lancer dans le jeu.

Affichage du jeu lors d'une sortie de route

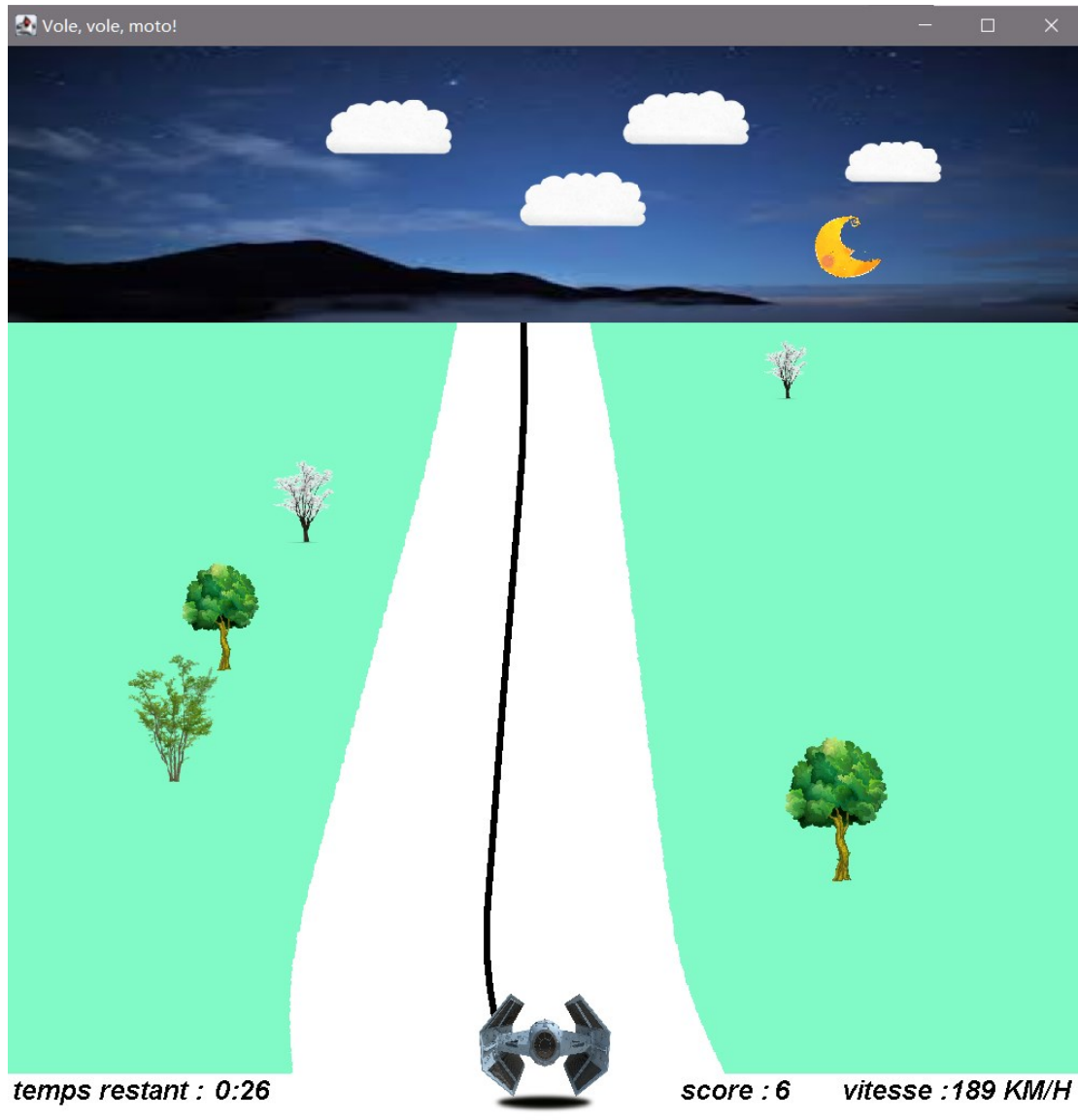


Figure 29 Interface de sortie de route

Affichage du jeu lors de l'apparition de l'adversaire

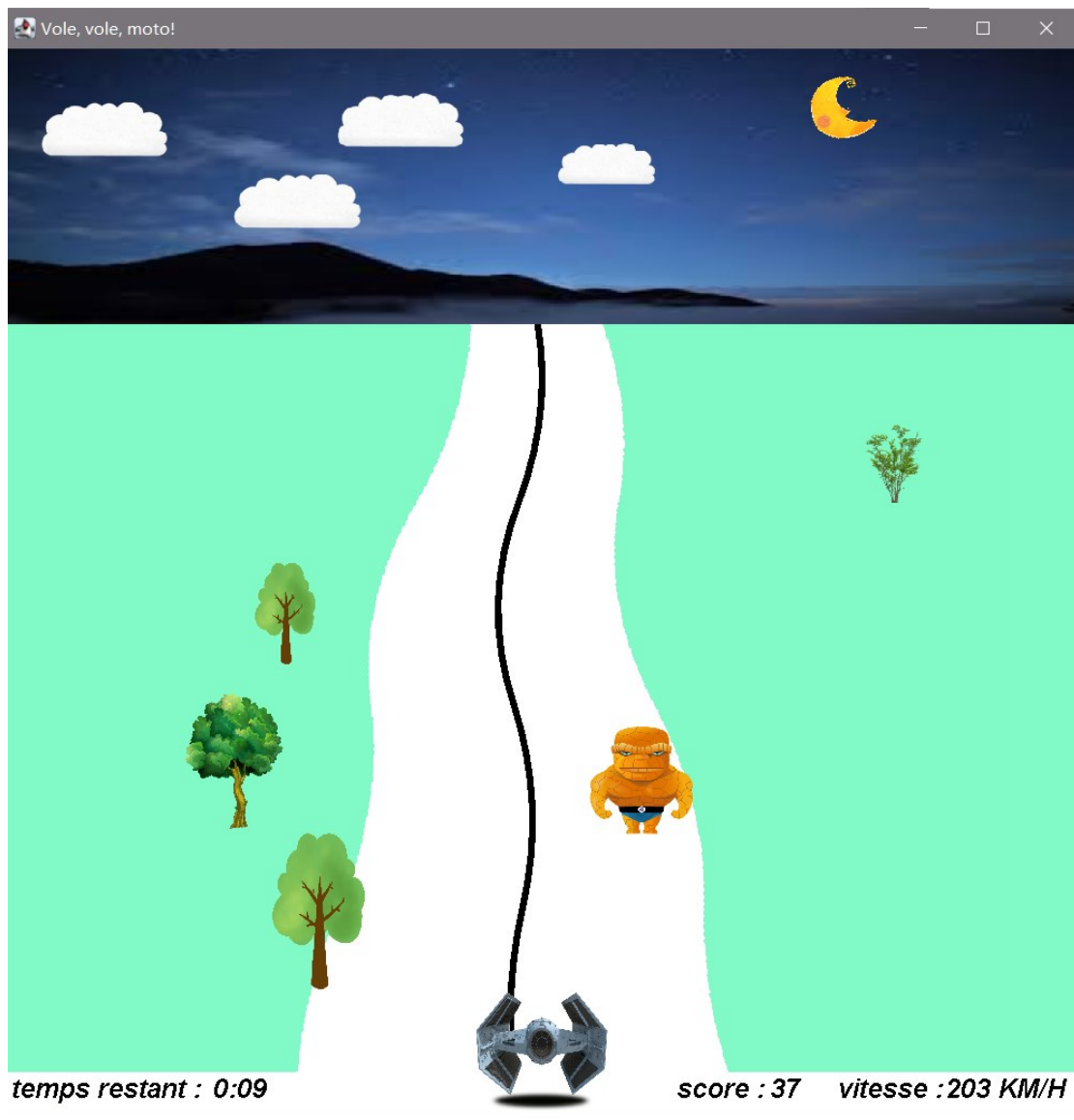


Figure 30 l'apparition de l'adversaire

C'est un adversaire mobile, mais aussi un obstacle.

Affichage du jeu lors de l'apparition de checkpoint



Figure 31 l'apparition de checkpoint

Le franchissement de la ligne rouge (point de contrôle) peut augmenter le temps restant.

L'état du véhicule montant et se déplaçant vers la gauche

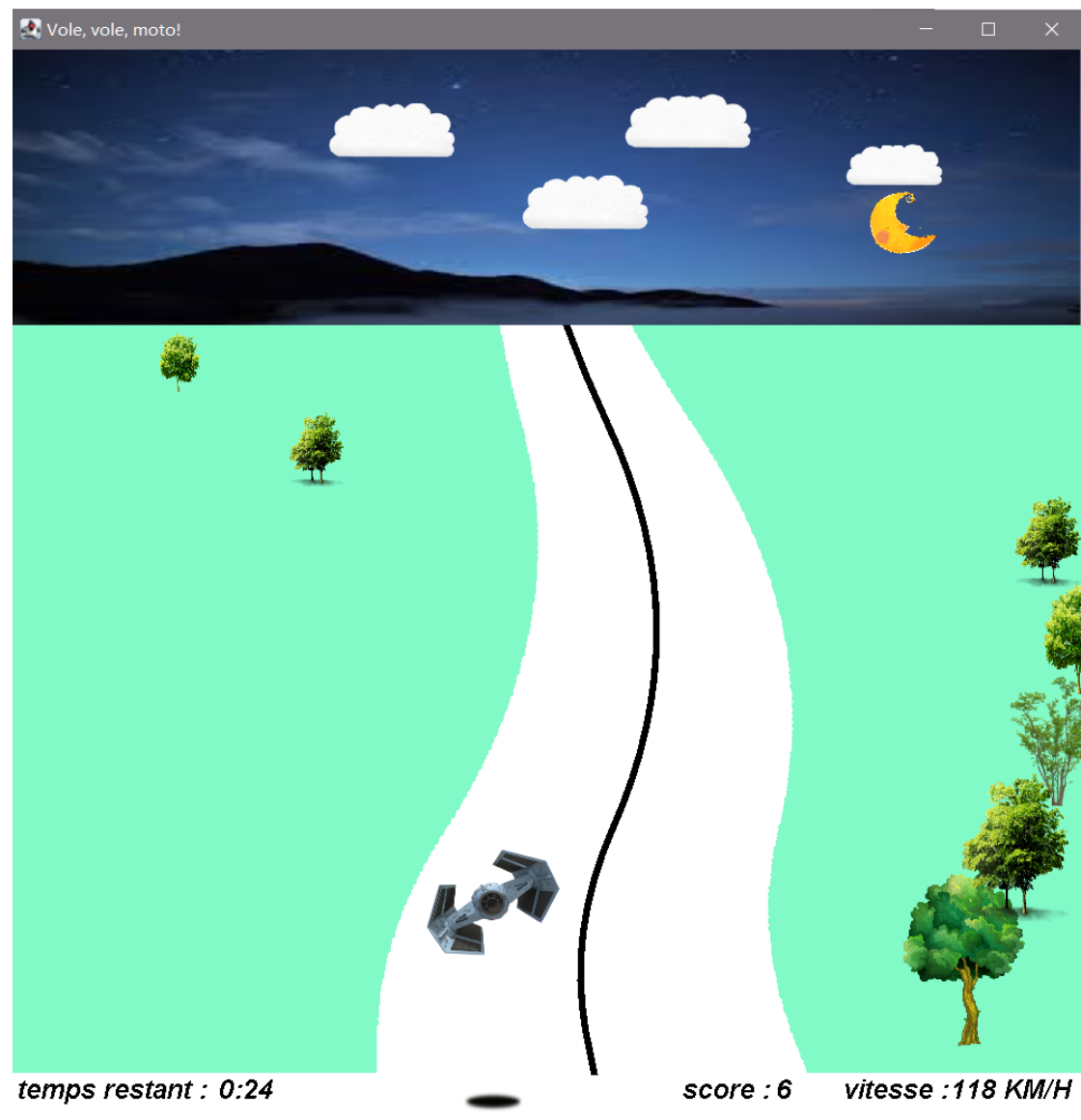


Figure 32 l'état du véhicule montant et se déplaçant vers la gauche

Appuyez sur le bouton gauche pour incliner le véhicule vers la gauche, appuyez sur le bouton haut pour soulever le véhicule, à mesure que le véhicule se lève, vous pouvez voir que l'ombre devient plus petite

Affichage l'écran fin

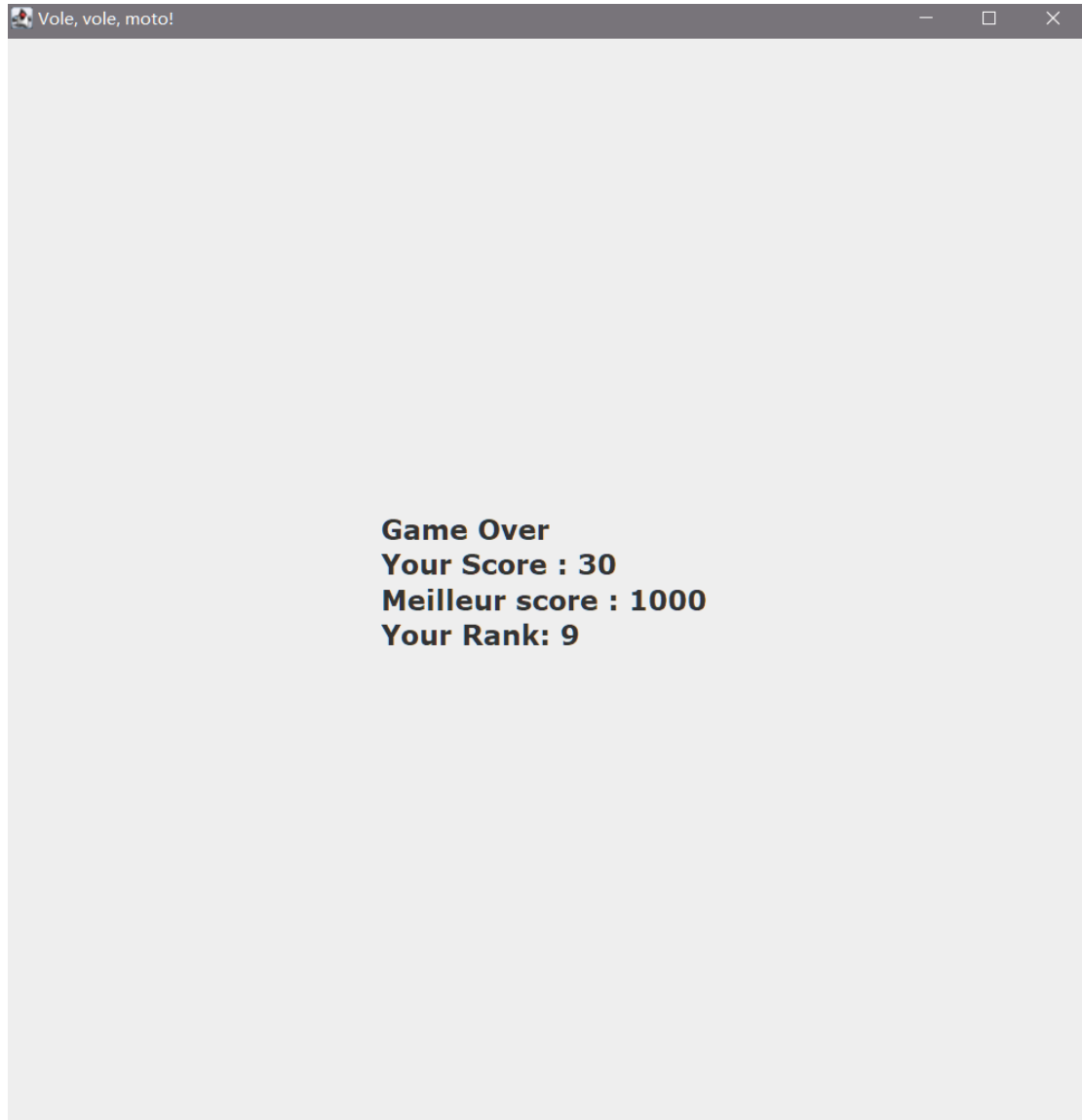


Figure 33 Affichage l'écran fin

Affiche le score et le score le plus élevé de l'histoire, ainsi que le classement du joueur

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 ». Appuyé le clavier sur la fenêtre pour faire déplacer de la moto.
- Mode d'emploi (cas .jar exécutable) : Appuyé le clavier sur la fenêtre pour faire déplacer de la moto. La moto peut juste déplacer A gauche ou A droite.

Documentation développeur

Le programme est lancé dans Main.java. Nous avons une classe mère Paysage.java. Si vous ajoutez simplement des décorations, vous pouvez hériter de cette classe parente pour réduire la quantité de code. Data.txt enregistre le score historique (L'état initial du texte a été écrit comme 0 et ne peut pas être supprimé, sinon une erreur de lecture se produira)

Fonctionnalités que nous attendons avec impatience d'ajouter :

- Perspective renforcée des objets plus correcte (nous avons simplement changé la taille de l'image et les coordonnées de la route.)
- Embellissez l'interface de démarrage et terminez l'interface.
- Ajouter des objets bonus (points supplémentaires, accélérer)
- Ajouter la météo (pluie, neige)

L'embellissement de l'interface de démarrage doit être effectué dans View.StartGame.java.

Embellir l'interface de fin doit être fait dans View.EndGame.java.

Vous pouvez ajouter la météo dans View.Affichage.java, et vous devrez peut-être créer une nouvelle classe dans le modèle pour améliorer le mécanisme météo.

La perspective renforcée des objets peut être développée dans Model.

Conclusion

Nous avons bien complété les fonctions requises par ce projet, page d'accueil, mécanisme d'accélération, mécanisme de clavier, route courbe de Bézier, décoration de fond intéressante, adversaires un peu intelligents et obstacles, mais nous pouvons mieux organiser la structure du code, ce qui aidera les autres à prendre en charge le développement ultérieur.

Le premier problème que nous avons rencontré a été l'écriture de code répétitif. Comme l'héritage n'était pas utilisé, nous avons écrit beaucoup de code similaire.

Le deuxième problème est le calcul de l'accélération. Comme nous n'avons pas bien compris le sujet, nous avons utilisé un changement fixe pour l'affichage de l'accélération, ce qui signifie que la vitesse de sortie de la piste est réduite à une valeur fixe, et la vitesse devient 0 au point le plus éloigné, ce qui est un peu contre-intuitif. Après la communication du professeur, nous avons eu l'idée du calcul de la vitesse.

La troisième question est le dessin des courbes de Bézier. Nous avons passé une partie du temps à réfléchir, et finalement nous avons vérifié les documents sur Internet et avons utilisé notre persévérance et notre travail acharné pour surmonter ce problème.

Un nouveau problème est apparu après le tracé de la courbe. Le jugement d'accélération n'est plus applicable au point central d'origine (car la courbe et la ligne brisée ne se chevauchent pas à certains endroits). Afin de pouvoir juger l'accélération avec plus de précision, nous voulons calculer le Bézier L'abscisse de la courbe, nous avons rencontré beaucoup de difficultés ici, et enfin vu les informations sur Internet, après beaucoup d'essais nous avons obtenu le résultat, ce point peut aussi être une bonne couleur des deux côtés de la route.

TABLE DES FIGURES

Figure 1	Le classement difficulté
Figure 2	: cahier des charges Séance 4
Figure 3	: diagramme de Gantt Séance 4
Figure 4	: cahier des charges Séance 5
Figure 5	: diagramme de Gantt Séance 5
Figure 6	: cahier des charges Séance 6
Figure 7	: diagramme de Gantt Séance 6
Figure 8	: cahier des charges Séance 7
Figure 9	: diagramme de Gantt Séance 7
Figure 10	: cahier des charges Séance 8
Figure 11	: diagramme de Gantt Séance 8
Figure 12	: cahier des charges Séance 9
Figure 13	: diagramme de Gantt Séance 9
Figure 14	: cahier des charges Séance 10
Figure 15	: diagramme de Gantt Séance 10
Figure 16	Introduction de la composition du projet
Figure 17	Introduction à l'interface graphique
Figure 18	Diagramme d'interaction entre les modèles
Figure 19	Diagramme de modèle
Figure 20	motoL
Figure 21	moto
Figure 22	motoR
Figure 23	Diagramme schématique de la courbe de Bézier.
Figure 24	Formule de la courbe de Bézier.
Figure 25	Formule de la courbe de Bézier.
Figure 26	Résolvez les deux solutions de l'équation.
Figure 27	Schéma de principe de la collision entre le véhicule et l'adversaire.
Figure 28	Interface de démarrage
Figure 29	Interface de sortie de route
Figure 30	l'apparition de l'adversaire
Figure 31	l'apparition de checkpoint
Figure 32	l'état du véhicule montant et se déplaçant vers la gauche
Figure 33	Affichage l'écran fin