Fonctionnalités :

* Implémentation de l’interface graphique
* Gestion des déplacements de la voiture
* Implémentation de la route et des points de contrôle
* Gestion de la vitesse
* Gestion du temps

5.Conception détaillée

* Implémentation de l’interface graphique

On utilise la classe **Affichage** pour l’interface graphique. Dans sa méthode **paint** héritée de la classe **JPanel** on appelle un ensemble de sous méthodes chacune chargée de l’affichage d’un élément de l’interface :

* **paintVoiture**: affiche la voiture à partir des valeurs données par les constantes de classe : **HAUTEUR\_FENETRE** (hauteur de la fenêtre) ; **LONGUEUR** (longueur de figure représentant la voiture) ; **LARGEUR** (largeur de la voiture). Pour l’instant la voiture est représentée par un carré c’est pour cela qu’on appelle la méthode **drawRect** de la classe **Graphics**. La voiture reste au milieu de l’écran tout le long de la partie, ce sont les autres éléments de l’interface qui changent leur position.
* **paintRoute** : affiche la route. On récupère les points visibles grâce à la méthode **getRouteGauche** de la classe **Route**. Cette méthode renvoie les points de l’extrémité gauche de la route, pour afficher aussi l’extrémité droite on se sert de l’attribut **Ecart** de la classe **Route** représentant la largeur de la route. Un point est considéré comme visible lorsqu’il est situé au-dessus la hauteur de l’écran moins la valeur de l’attribut **score** de la classe **Route** qui est incrémenté par la méthode **setScore** de la classe **Route** appelée par la classe **Avancer.** Cette incrémentation du score sert à représenter le défilement vers le haut de la route. Lorsqu’on un point de la route n’est plus considéré comme visible on le retire de la liste représentant les points de la route **pointsGauche** qui est un attribut de la classe **Route**. Ensuite pour gérer le défilement horizontal de la route on se sert de l’attribut **decalage** de la classe **Affichage** qui est incrémenté par les méthodes de la classe **Etat** appelés par la classe **Contrôle** lorsque le joueur déplace la voiture grâce à son clavier (on verra leur implémentation plus tard). L’attribut **decalage** est ajouté aux abscisses des différentes éléments à ajouter. Cette méthode n’est pas utilisée que pour l’affichage de la route mais aussi pour les autres sous méthodes d’affichage.
* **paintPointDeControle**: affiche les points de contrôle. Les points de contrôle sont représentés par des lignes horizontales sur la route. Le calcul de leurs cordonnées grâce au calcul de la pente va être traité plus tard. On récupère les points grâce à la méthode **getPointDeControle** de la classe **Route** dont le fonctionnement est similaire à celui de la méthode **getRouteGauche**.
* **paintDecor :** affiche le décor. Les éléments du décor sont : les montagnes et la ligne d’horizon. Ces deux éléments sont gérés par la classe **Decor** au sen de laquelle on retrouve : la constante **HORIZON** représentant la ligne de l’horizon ; et la méthode **createMontagne** qui génère aléatoirement des points dont les valeurs de leurs abscisses sont bornées par les constantes de classe  **X\_MAX** et **X\_MIN ;** et dont les valeurs de leurs ordonnées sont bornées par les constantes de classe **Y\_MIN** et **Y\_MAX.** Les différents points représentant la montagne sont stockées au sein de l’attribut **pointList** de la classe **Decor**.

Pour délimiter la larguer minimale et maximale que la voiture peut parcourir horizontalement on a décidé d’utiliser deux lignes verticales situés à chacune des deux extrémités et que le joueur ne peut pas franchir.

* **paintInformations**: affiche les informations relatives à la partie de joueur. Le temps représenté par l’attribut **temps** de la classe **Etat**, et la vitesse représentée par l’attribut **vitesse** de la classe **Etat**.
* Gestion des déplacements de la voiture

L’état de la voiture et de la partie sont contenues au sein de la classe **Etat** du modèle. L’attribut **x** de la classe **Etat** représente la valeur actuelle de l’abscisse du joueur, et la constante **saut** de la classe **Contrôle** représente la valeur d’un déplacement de la voiture. La classe **Contrôle** implémente la classe **KeyListener** et est donc chargée de répondre à l’interaction du joueur avec le clavier. Pour cela on se sert de l’attribut **keys** de la classe **Control** qui associe à chaque touche avec laquelle le joueur peut interagir (les flèches directionnelles droite et gauche) un booléen. Si le joueur appuie sur la touche alors son booléen est mis à vrai grâce à la méthode **keyPressed** de la classe **Control** et si le joueur relâche la touche alors son booléen est mis à faux grâce à la méthode **keyReleased** de la classe **Control**. Ces deux méthodes sont appelées par la méthode **update** de la classe **Control** elle-même appelée par la méthode **run** du thread **Déplacements**. On a décidé de partir sur cette implémentation pour éviter qu’il y ait des pauses lors des changements de direction et donc pour rendre le jeu plus fluide.

La modification de l’abscisse de la voiture est effectuée grâce aux méthodes **moveLeft** et **moveRight** de la classe **Etat** qui changent la valeur de l’attribut **x,** en vérifiant que les bornes dont a parlé avant de l’interface graphique ne sont pas franchies. Ces deux méthodes sont appelées au sein de la méthode **update**.

* Implémentation de la route et des points de contrôle
* Pour l’implémentation de la route on utilise l’algorithme les suivant :
  + On va commencer par fournir les attribut de classe **Route** utilisés : **pointsGauche**, qui représente la liste de points de la route et que l’on va appeler **List** dans le pseudo-code ; **objy**, qui représente l’ordonnée du prochain point qui va être ajouté à **List**; **limite**, qui est utilisée pour savoir si les limites aux extrémités donnée par les attributs **xmin** et **xmax** sont atteints ; **direction**, qui représente la direction actuelle de la route ; **x1**, qui représente l’abscisse du dernier point ajouté ; et **y**, qui représente l’ordonnée du dernier point ajouté.
  + PSEUDO CODE :

Méthode init() :

On ajoute le premier point à **List** avec les coordonnées **x1** et **y** initialisée dans la constructeur

**objy** = **y** – 100

Tant que **objy** > **abscisse de l’horizon**

(on ajoute des points au-delà de l’horizon pour prévoir l’accélération du défilement de la route)

Méthode pour ajouter un point :

**limite** = faux

Si **direction** == Droit

**y** = **objy**

On ajoute à **List** le point à la coordonnée **x1** et **y**

Sinon Si **direction** == Droite

**x1** = **x1** + (**y**-**objy**)

**y** = **objy**

Si **x1** + **Ecart** >= **xmax**

**x1** = **xmax** – **Ecart**

**limite** = vrai

**direction** = Gauche

Fin Si

On ajoute à **List** le point à la coordonnée **x1** et **y**

Sinon Si **direction** == Gauche

**x1** = **x1** – (**y**-**objy**)

**y** = **objy**

Si **x1** <= **xmin**

**x1** = **xmin**

**limite** = vrai

**direction** = Droite

Fin Si

On ajoute à **List** le point à la coordonnée **x1** et **y**

Fin Si

**objy** = **y** – (valeur tirée au hasard entre **ymin** et **ymax**)

Si **limite** == faux

r = valeur tirée au hasard entre 0 et 2

Si r == 0

**direction** = Droit

Sinon Si r == 1

**direction** = Droite

Sinon

**direction** = Gauche

Fin Si

Fin de la méthode pour ajouter un point

Fin de la méthode pour initialiser **list**

* Pour l’implémentation des points de contrôle on utilise l’algorithme suivant
* On va commencer par fournir les attributs de la classe **Route** utilisées : **pointsGauche**, qui représente la liste de points de la route que l’on va appeler **ListRoute**;