# Projet - PCII



# Introduction :

Le joueur dirige une voiture en se déplaçant de gauche à droite et de haut en bas, plus il s'écarte de la piste, plus sa vitesse baisse rapidement, le but étant d'atteindre le prochain checkpoint avant la fin du temps imparti tout en évitant les obstacles et les joueurs ennemis.

# I - Analyse :

- **Scrolling :** Le joueur est toujours au milieu de la fenêtre, quand on appuie sur Q ou D le décor est redessiné quelques pixels plus à droite ou plus à gauche.
- **Défilement piste :** La piste défile du haut vers le bas avec une certaine vitesse mise à jour toutes les secondes en fonction de la vitesse et de l'accélération actuelle
- **Génération piste :** Des points sont générés au fur et à mesure et les points qui sortent de la fenêtre sont supprimés
- **Défilement décor :** La montagne défile de gauche à droite ou de droite à gauche quand on se déplace avec Q ou D
- **Génération montagne :** Un nouveau point de la ligne brisée est généré quand le point qui va arriver dans la vue est au bord (début ou fin) de la liste de points générés.
- **Affichage montagne :** On affiche seulement les points (reliés par des lignes) qui sont dans la vue du joueur et un point en plus à gauche et à droite pour que la ligne brisée aille jusqu'aux bords de la fenêtre.
- **Calcul accélération :** On calcule l'accélération en fonction de la distance entre le joueur et le milieu de la piste
- **Calcul Vitesse :** La vitesse est mise à jour toutes les secondes, en fonction de l'accélération.
- **Game Over :** Quand le compte à rebours affiché en haut à gauche ou la vitesse atteignent 0 le joueur a perdu, on stoppe le jeu et on écrit Game Over au milieu de la fenêtre.
- **Animation moto :** Le joueur est représenté par 3 images, une moto qui va tout droit si le joueur ne se déplace pas sur les côtés ou bien une moto qui tourne à droite ou à gauche quand le joueur appuie sur Q ou D. Quand il vole, on lui ajoute des ailes et une ombre et quand la moto gagne de l'altitude on ajoute des traces de propulsion.
- **Génération Nuages :** À chaque rafraichissement de l'affichage si on a pas atteint le maximum de nuages on a une certaine probabilité de générer un nuage juste à droite de la fenêtre, il aura une hauteur, largeur et altitude aléatoires (mais bornées) . Quand un nuage sort de la vue du joueur, il est supprimé.
- **Défilement Nuages :** Chaque nuage se déplace de quelques pixels vers la gauche toutes les 100 millisecondes. Tous les nuages sont affichés car ceux qui ne sont plus visibles ont été supprimés.

- **Checkpoints damier :** Chaque checkpoint est généré avec un numéro de voie et n'est que sur un tiers de la piste. À chaque passage de checkpoint on gagne une seconde de moins que celui d'avant (on gagne au minimum 5 secondes).

- **Obstacles (fixes) :** Un obstacle est généré avec une certaine probabilité quand le nombre d'obstacles actuels est en dessous du maximum. Leur position X et leur taille sont actualisées en fonction de la largeur de la piste (pour prendre en compte la perspective).

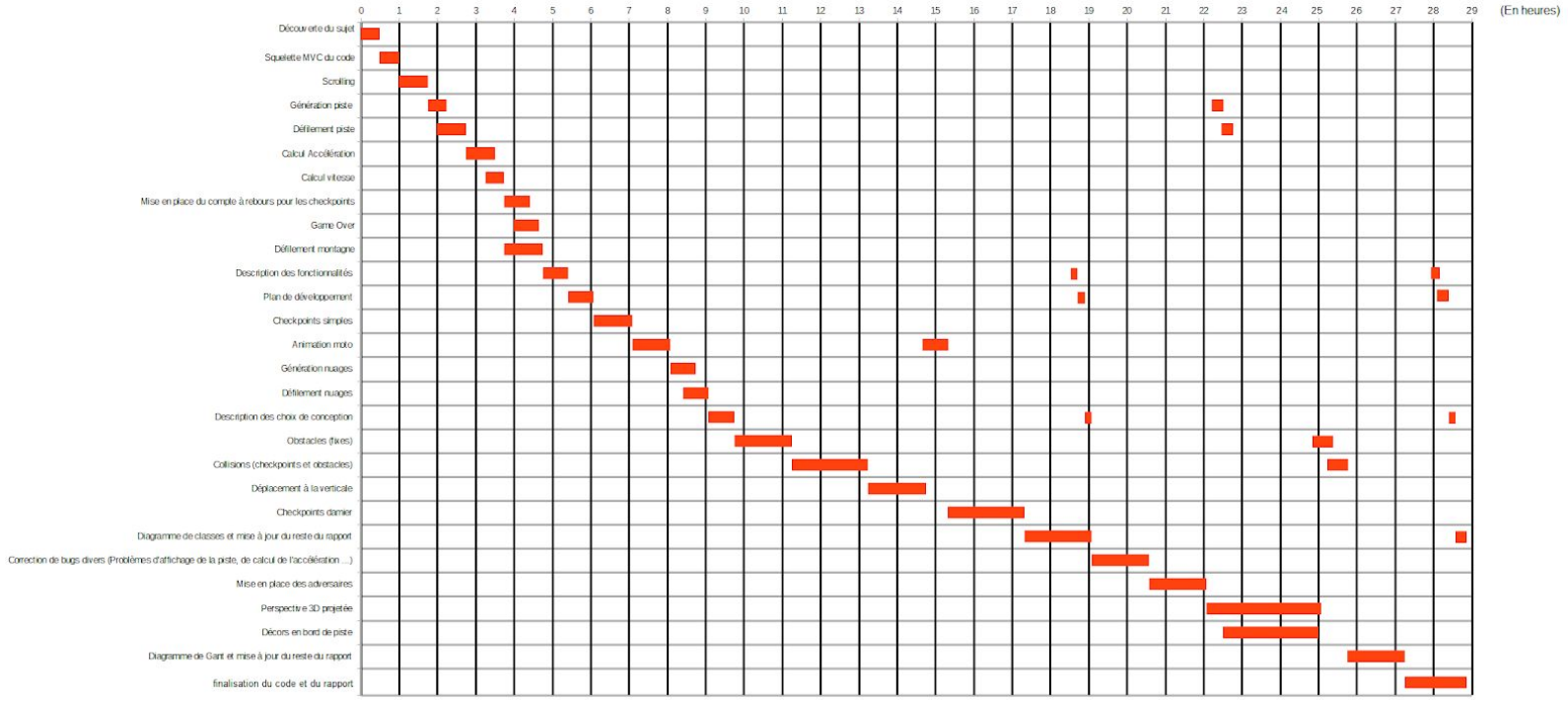
- **Collisions (checkpoints et obstacles) :** Quand la moto est chevauchée par un obstacle ou un checkpoint par la gauche ou la droite (ou les deux), il y a une collision. Une collision avec un obstacle fait disparaître ce dernier et fait chuter la vitesse de 1. Une collision avec un checkpoint ajoute du temps au timer et génère le prochain checkpoint.

- **Déplacement à la verticale :** Quand on presse la touche Z la moto décolle, la touche S la fait descendre. Aussi la moto descend quand la vitesse est < 4. Au niveau des collisions, la moto passe au-dessus des obstacles quand elle est à une altitude >= 1 et gagne du temps en passant au-dessus des checkpoints peu importe son altitude.
- **Mise en place des adversaires :** Les adversaires sont générés quand on n'a pas atteint le maximum et sont supprimés quand ils sortent du champ de vision. Quand il y a une collision entre le joueur et un adversaire, si le joueur est devant, alors l'adversaire recule sinon l'adversaire avance et le joueur perd de la vitesse. Il n'y a pas de collisions si le joueur est à au moins 2 d'altitude. La vitesse des adversaires s'adapte en fonction de leur position par rapport au milieu de la piste.

- **Perspective 3D projetée :** La piste est sur un axe z, la hauteur des objets et sur un axe y, et l'axe x correspond à l'axe x de la fenêtre. Ce repère 3D est ensuite projeté sur le plan de la fenêtre (qui coupe le repère 3D ).

- **Décors en bord de piste :** Les décors sur le bord de la piste sont générés de la même manière que les obstacles fixes et sont supprimés quand ils sortent de la vue (que ça soit par le bas ou les côtés de la fenêtre). Il y a différentes fonctions de dessins pour les décors, qui ont chacune une probabilité d'être appelée. Il n'y a pas de collisions avec les éléments de décor.

# II - Plan de développement :




**Séance 1 :**

- Découverte du sujet
- Squelette MVC du code
- **Scrolling**
- **Jalon :** la moto est représentée par une croix, la piste se décale à droite ou à gauche quand on appuie sur Q ou D, les points de la piste sont générés aléatoirement et la piste est fixe.

**Séance 2 :**

- **Génération piste**
- **Défilement piste**
- **Calcul Accélération**
- **Jalon :** la piste défile en fonction de la vitesse, on affiche le score (pixels parcourus) et l'accélération diminue en fonction de la distance entre la position de la moto et le milieu de la piste

**Séance 3 :**

- **Calcul vitesse**
- Mise en place du compte à rebours pour les checkpoints
- **Game Over**
- **Jalon :** la vitesse se met à jour toutes les secondes en fonction de l'accélération.

**Séance 4 :**

- **Défilement montagne**
- Description des fonctionnalités
- Plan de développement
- **Jalon :** la montagne défile en fonction des déplacements

**Séance 5 (25/03) :**

- Checkpoints simples
- **Animation moto**
- **Génération nuages**
- **Défilement nuages**
- Description des choix de conception
- **Jalon :** la moto doit atteindre le prochain checkpoint avant le temps imparti, la moto et les nuages sont animés par des images.

**Séance 6 (08/04) :**

- **Obstacles (fixes)**
- **Collisions (checkpoints et obstacles)**
- **Déplacement à la verticale**
- **Checkpoints damier**
- Diagramme de classes et mise à jour du reste du rapport
- Correction de bugs divers (Problèmes d'affichage de la piste, de calcul de l'accélération ...)
- **Jalon :** la moto doit éviter les obstacles, viser les checkpoints et peut se déplacer à la verticale

**Séance 7 (Aujourd'hui 22/04) :**

- **Mise en place des adversaires**
- **Perspective 3D projetée**
- **Décors en bord de piste**
- Diagramme de Gant et mise à jour du reste du rapport
- **Jalon** : version bêta du projet

**Séance 8 :**

- Finalisation de l'implémentation et des tests
- Finalisation de la documentation
- Préparation de la soutenance
- **Jalon :** remise du rapport et soutenance la semaine prochaine

# III - Conception :

## Modèle MVC :

Model : (Etat) contient les infos sur l'état du jeu, c'est à dire l'emplacement des différents éléments à afficher.

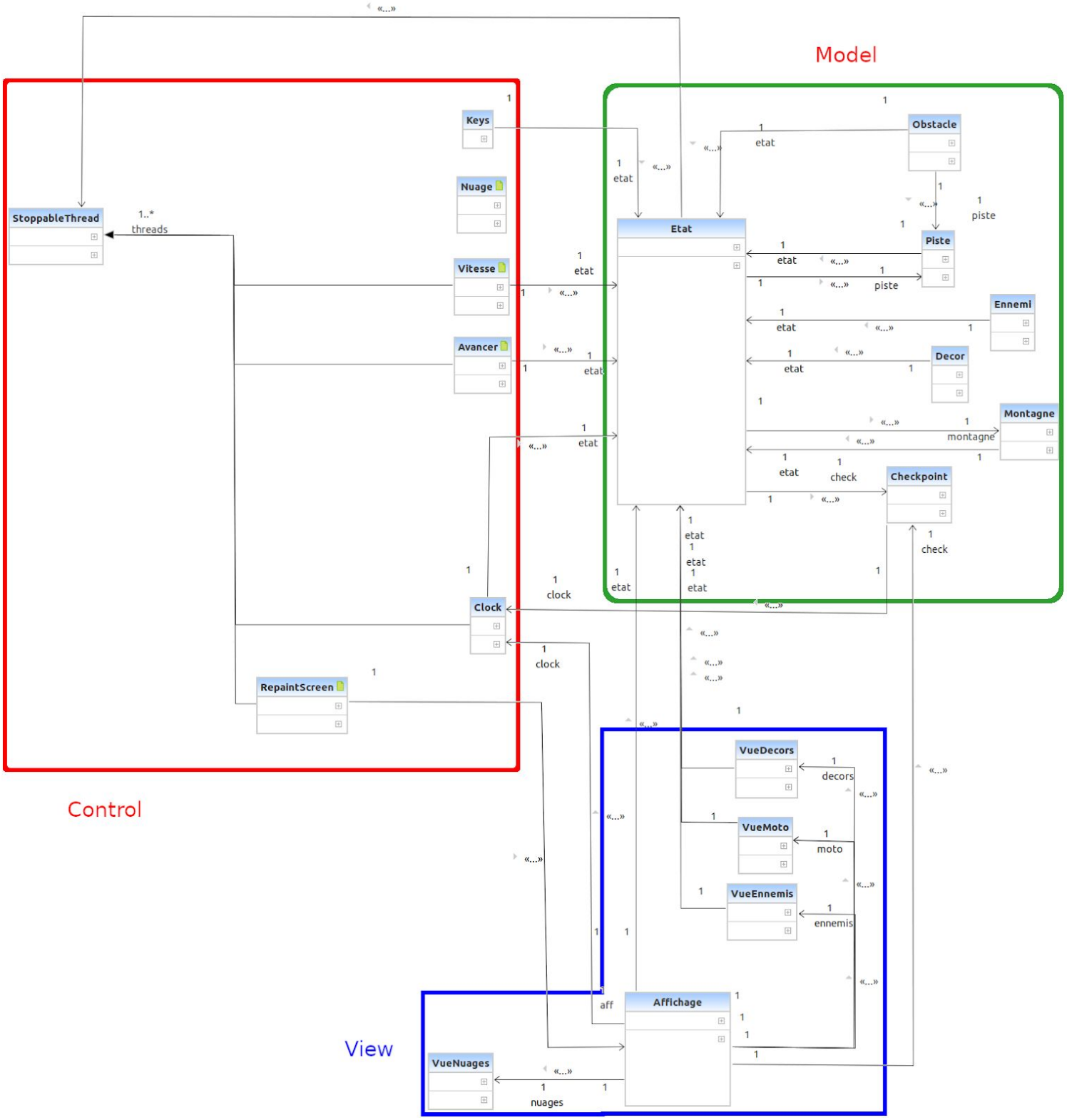
View : (Affichage) Affichage de la piste, des nuages, des montagnes et des infos sur l'état de la partie (Vitesse, Score, temps restants)

Control : Gestion des appuis clavier, défilement de la piste, mise à jour de la vitesse, défilement des nuages, rafraîchissement de l'affichage, lancement des tests de franchissement de checkpoints.

## Les Threads (control) :

- **Avancer :** appelle la fonction Etat.avance() toutes les 41ms pour mettre à jour la position du joueur dans le modèle.
- **Keys :** Quand l'utilisateur appuie sur Q ou D, on appelle Etat.goLeft() ou goRight(). Et quand il appuie sur Z ou S la moto monte ou descend. Aussi, quand la partie est finie et que l'utilisateur appuie sur R, on relance une nouvelle partie.
- **Nuage :** représente un nuage avec sa position et ses dimensions, on décale la position du nuage vers la gauche toutes les 100ms.
- **RepaintScreen :** appelle la fonction repaint de l'Affichage toutes les 41ms.
- **Vitesse :** toutes les secondes on appelle la procédure Etat.updateVitesse() pour mettre à jour la vitesse en fonction de l'accélération.

## Diagramme de classes :




Le diagramme complet avec est dans le dossier “UML”.

## Génération Piste :

```
incr <- HAUT/4
dec <- 20
posY <- 0

procédure initPoints:
        Pour y décroissant allant de HAUT à posHorizon:
                x <- randint(-largeurPiste/2-dec,-largeurPiste/2+dec) + LARG/2
                points.add(new Point(x,y))

procédure ajoutePoint:
        y <- points[points.size()-1].y - incr;
        x <- randint(-largeurPiste/2-dec,-largeurPiste/2+dec) + LARG/2;
        points.add( new Point(x,y))

procédure updatePoints:
        //on ajoute un point si le dernier point entre dans le champ de vision
        Si points[points.size-1] + posY >  posHorizon
        Alors ajoutePoint()

        //on retire le 1er point si le deuxième sort de la fenêtre (par le bas)
        Si points[1].y + posY >  HAUT
        Alors points.remove(0)
```

## Génération Montagne :

```
posX ← position x du joueur
procédure initPoints:
                points ← new ArrayList<Point>()
                Pour i allant de 0 à LARG i+= randint(ecartMin,ecartMax):
                        y ← randint(yMin,Affichage.posHorizon)
                        this.points.add(new Point(i,y))

procédure updatePoints:
                Si PosX < points[0].x
                Alors addPointGauche()
                Sinon:
                        Si PosX + LARG > points[points.size()-1].x
                        Alors addPointDroite()

procédure addPointGauche:
                x ← points[0].x - randint(ecartMin, ecartMax)
                y ← randint(0, posHorizon)
                // ajout au début
                points.add(0, new Point(x,y))
```

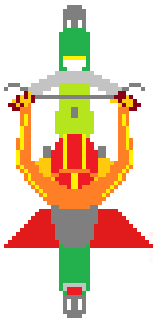
procédure addPointDroite:

```
x ← points.get(points.size()-1).x + randint(ecartMin, ecartMax)
y ← randint(0, Affichage.posHorizon)
// ajout à la fin
points.add(new Point(x,y))
```

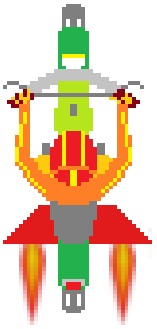
## Etat de la moto :

La direction de la moto est codée par un entier (entre 0 et 2) comme ci-dessous :


Quand la moto est dans les airs
on lui ajoute des ailes



Et quand elle gagne de l'altitude
on lui ajoute des traces de propulsion



# IV - Documentation utilisateur :

Dépendances :

- Nécessite Java 11 ou plus (toutes les versions)
- Lancer le fichier PCII.jar

Contrôle de la moto :
On contrôle la moto avec les touches Q et D pour aller à gauche et à droite
On peut aussi utiliser Z et S pour gagner ou perdre de l'altitude

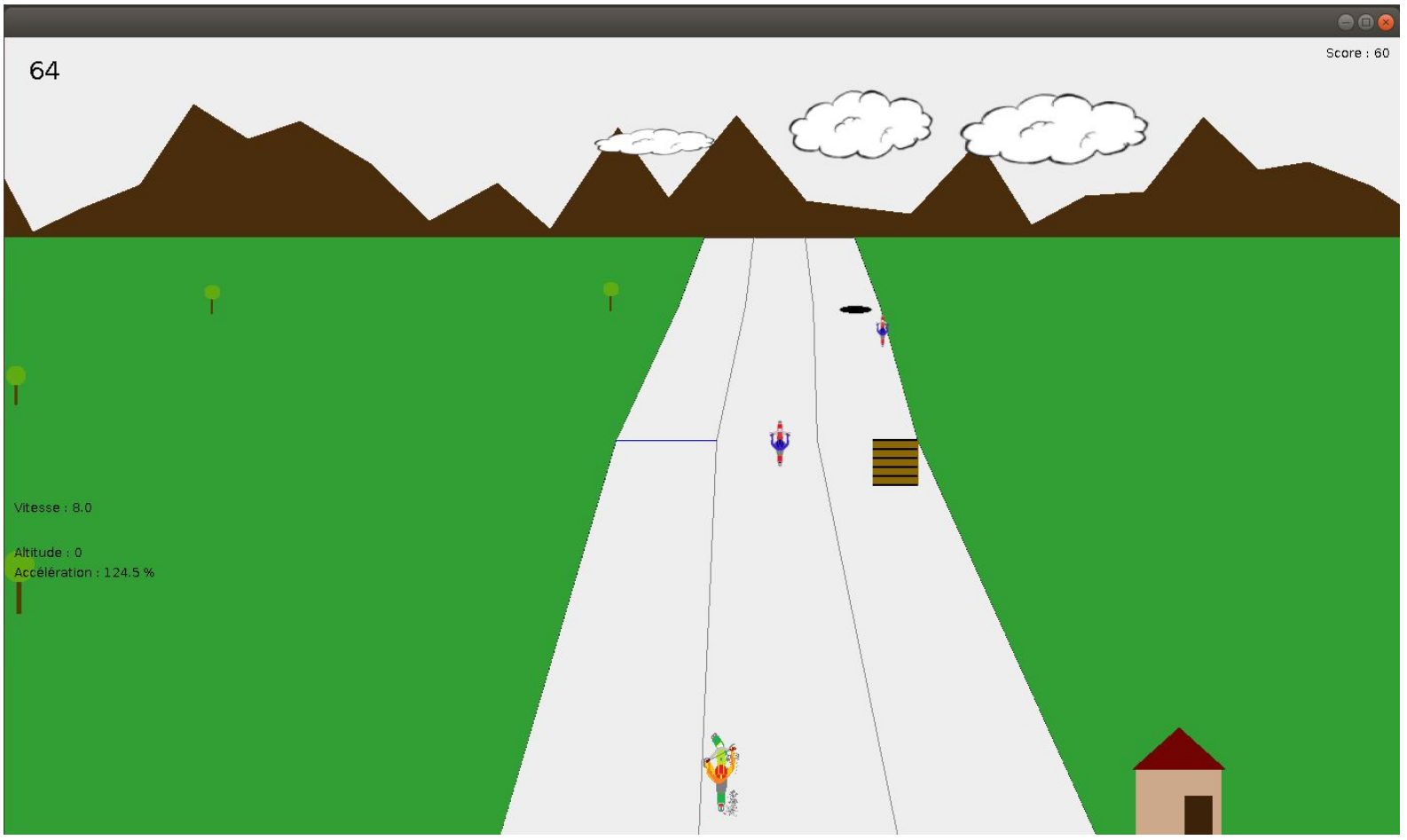
Game Over :
On a perdu quand le temps restant en haut à gauche atteint 0 ou que la moto n'a plus de vitesse.

Checkpoints:
On gagne du temps en passant sur les checkpoints (les lignes bleues) tout en évitant les obstacles sur la route et les autres motos. Mais attention, pour franchir un checkpoint, il faut que la moto soit au sol.

Obstacles :
Il y a deux obstacles différents, le panneau en bois qui fait perdre (5?) de vitesse et peut être évité avec une altitude >=1, et le trou qui fait perdre 7 de vitesse mais peut être évité dès qu'on a une altitude > 0.

Moto :
Une collision avec une moto ennemie fait perdre de la vitesse si on est derrière elle mais si on l'a dépassée, on lui fait une queue de poisson. Les motos ennemies peuvent être évitées avec une altitude >= 2.

# V - Résultat :



# Documentation Développeur :

## Modifier le programme :

Dans Etat :

- Pour changer la vitesse max du joueur on utilise **vitesseMax**
- Pour gérer la génération des décors et des ennemis on utilise **maxDecors** et **maxEnnemis** ainsi que **probaDecor** et **probaEnnemis**.
- Pour la vitesse des déplacements gauche/droite il faut modifier la constante **deplacement.**

Dans Affichage :

- pour changer les couleurs du sol et des montagnes on modifie **COLOR_HERB** et **COLOR_MOUNTAINS**.
- pour changer les dimensions de la fenêtre on modifie **LARG** et **HAUT**.

Pour plus d'infos sur les classes, attributs et méthodes, la documentation est disponible avec le raccourci "Javadoc" à la racine du zip.

## Idées d'améliorations :

- Améliorer le comportement des ennemis :
  On pourrait tester les collisions entre les obstacles et les ennemis pour qu'ils perdent de la vitesse quand il y en a une en s'inspirant de la procédure **Etat.testCollisions**. Il faudrait alors permettre aux ennemis d'éviter les obstacles en adaptant leurs déplacements en fonction des positions des obstacles qui sont assez proches d'eux (en approximant le temps de réaction du joueur).
  On pourrait aussi adapter les déplacements à la position du joueur quand l'ennemi est proche de lui pour que les ennemis tentent de bloquer le joueur.
- Amélioration des décors :
  On peut facilement améliorer les dessins des décors en modifiant les fonctions de dessins dans **VueDecors**.
  On peut aussi ajouter de nouveaux décors en y ajoutant des fonctions de dessins, ensuite il faut adapter le constructeur de la classe **Decor** avec une condition supplémentaire pour le nouveau décor.
- Fluidifier l'affichage
- Il y'a un problème dans la fonction "Etat.testCheckpoint" qui teste les collisions entre la moto et le checkpoint. Au bout d'un certain temps de jeu, quand on passe assez proche d'un checkpoint il se décale un peu en arrière. Peut-être à cause d'un problème de concurrence sur la fonction Piste.getPiste() ou d'un problème de projection.

# Conclusion :

Ce projet nous a permis d'apprendre à créer une interface graphique interactive en Java avec le modèle MVC et en utilisant le Multi-Threading et à organiser un projet avec de nombreuses fonctionnalités et une quantité de code assez conséquente. Mais aussi à produire une documentation compréhensible avec une Javadoc, des diagrammes de Gant et des diagrammes de classes.