Projet PCII

Introduction

Nous allons à travers ce projet réaliser un jeu vidéo de type « course de voiture » à la première personne. Le but de ce jeu est de contrôler un véhicule aérien qui doit esquiver les obstacles et dépasser les autres véhicules appelés les concurrents. Nous allons, tout d'abord, utiliser la bibliothèque Swing afin de programmer l'interface graphique et après avoir affiché la moto, implémenter la gestion des déplacements par le biais des flèches directionnelles du clavier. Par la suite, l'environnement de la moto sera implémenté en commençant par l'animation de la route.

Analyse globale

Les principales fonctionnalités à développer sont l'interface graphique avec la création de la moto et la gestion du clavier qui va permettre de la déplacer sur la fenêtre de jeu.
Ces fonctionnalités sont prioritaires car elles permettent de mettre en place l'environnement de jeu et sont plutôt simples à implémenter.

Dans le prolongement de ces fonctionnalités vient la création de l'environnement de jeu. Nous commençons tout d'abord par l'animation de la route à vitesse constante qui est une fonctionnalité prioritaire dans la mesure où toute la mécanique du jeu repose sur son animation dans la fenêtre.

Les fonctionnalités qui ne sont pas prioritaires mais que nous avons tout de même implémentées sont la stylisation de la route avec les courbes de bézier et l'affichage d'un oiseau animé à travers la fenêtre. L'animation de l'oiseau reste simple à réaliser mais pour les courbes de Bézier il faut faire attention à avoir une ligne continue entre les différentes courbes qui composent la route.

D'autres fonctionnalités prioritaires sont la création de points de contrôle et de crédit de temps. Ces fonctionnalités permettront par la suite de définir une condition de fin de partie.

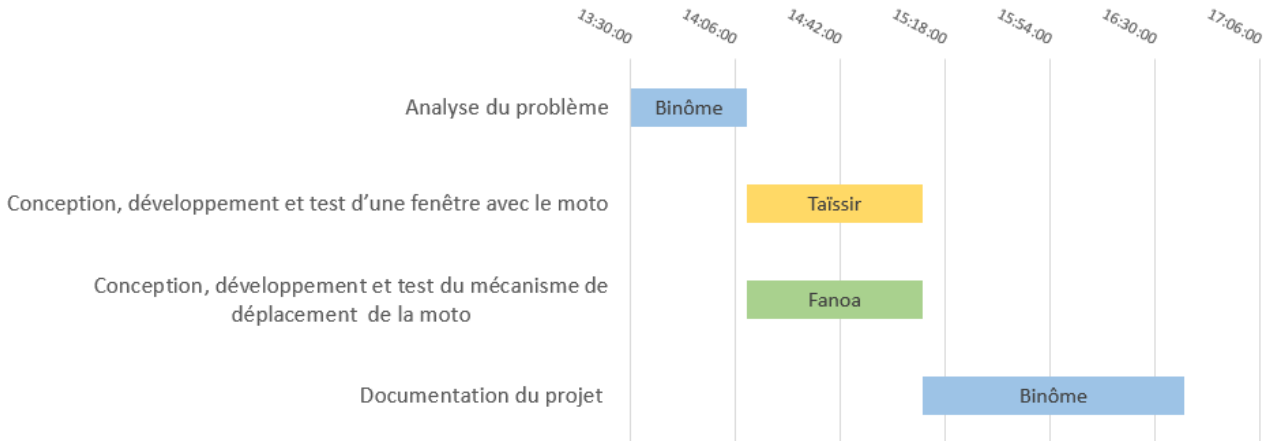
Plan de développement

Liste des tâches : (théorique)

- Analyse du problème (40 mn)
- Conception, développement et test d'une fenêtre avec le moto (60 mn)

- Conception, développement et test du mécanisme de déplacement de la moto (60 mn)
- Documentation du projet (90 mn)

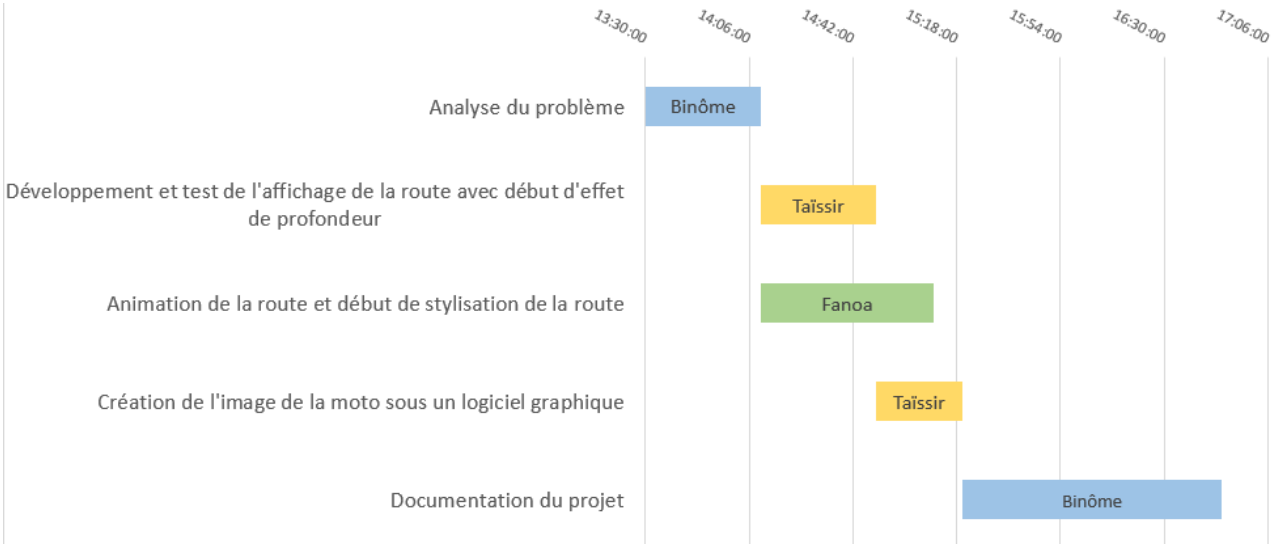
Diagramme de Gantt des différentes premières tâches du projet:



Liste des tâches de la deuxième séance : (théorique)

- Analyse du problème (40 mn)
- Développement et test de l'affichage de la route avec début d'effet de profondeur (40 mn)
- Animation de la route et début de stylisation de la route(60 mn)
- Création de l'image de la moto sous un logiciel graphique(30 mn)
- Documentation du projet (90 mn)

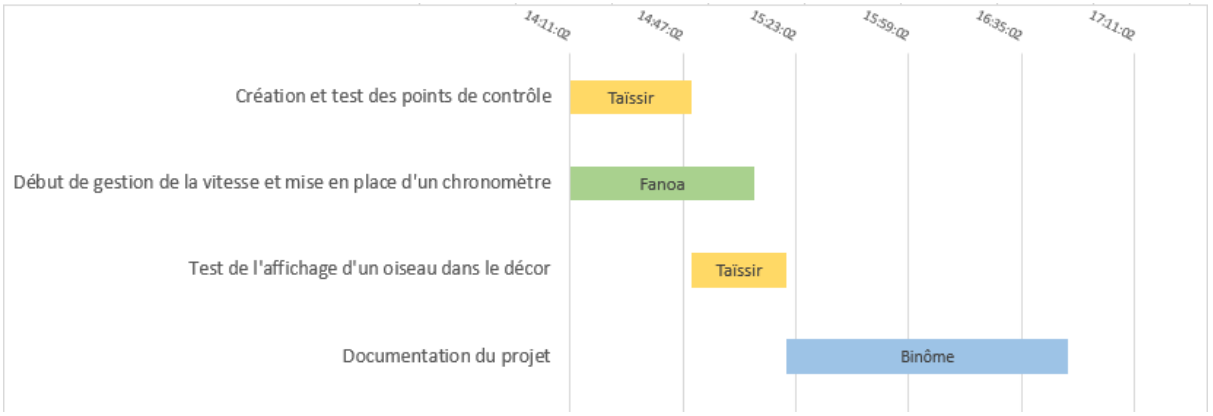
Diagramme de Gantt des différentes tâches du projet:



Liste des tâches de la séance : (théorique)

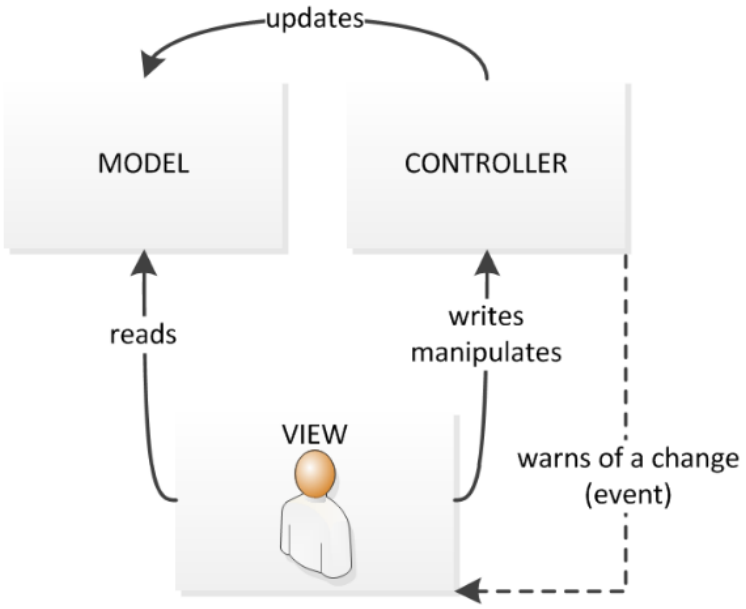
- Création et test des points de contrôle(40 mn)
- Début de gestion de la vitesse et mise en place d'un chronomètre(60 mn)
- Test de l'affichage d'un oiseau dans le décor (30 mn)
- Documentation du projet (90 mn)

Diagramme de Gantt des différentes tâches du projet:



Conception générale

Nous avons adopté le motif MVC, Model View Controller pour le développement de notre interface graphique.



Le motif MVC (*source : Wikimedia*)

Ce motif est composé de quatre classes principales :

- Affichage pour la vue
- Etat pour le modèle
- Control pour le contrôleur
- Main pour lancer le programme

Conception détaillée

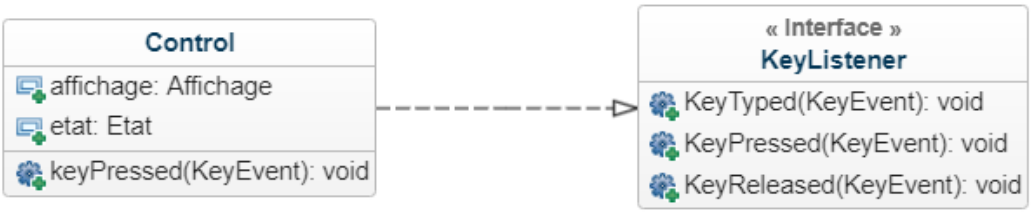
Tout d'abord au lancement du jeu, une fenêtre avec le menu principal s'ouvre. Cette fenêtre est créée dans la classe Main qui hérite de la classe JPanel. La fenêtre est composée d'une JFrame dans laquelle sont ajoutés un JLabel contenant l'image de fond et deux JButton. Les deux boutons sont le bouton start pour lancer le jeu et quitter pour quitter. Le bouton start permet au clic de fermer la fenêtre actuelle et de lancer la fenêtre de jeu.

Le développement d'une fenêtre de test consiste principalement en la création et l'implémentation d'une classe Affichage. Cette classe hérite de la classe JPanel, ce qui nous permet de l'ajouter à une fenêtre (on réalise cette opération dans le main avec l'instruction "fenetre.add(affichage)". Tout ce qui est "peint" par l'affichage y figurera donc. La classe affichage est construite autour d'une méthode principale, héritée de JPanel, la méthode paint(Graphic c). C'est au sein de cette méthode qu'on dessine sur la fenêtre les différents éléments du programme. Ici, nous avons choisis pour commencer de d'utiliser des images temporaires pour

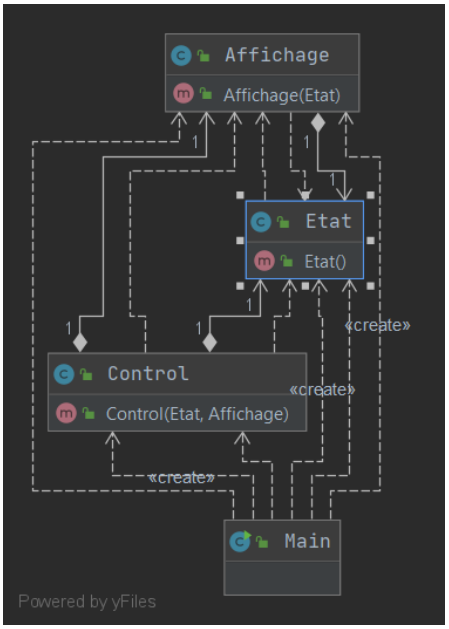
représenter facilement chacun des éléments du jeu. Beaucoup des éléments de cet affichage sont provisoires car l'objectif est ici de faire le minimum pour pouvoir interagir avec le modèle. Une fois que les fonctionnalités de base du modèle seront implémentées, un affichage plus intéressant pourra être produit.

Pour le déplacement de la moto dans la fenêtre de jeu, nous utilisons la programmation événementielle avec la classe KeyListener. La position de la moto est définie dans la classe Etat avec la constante "pos", qui va subir des modifications de coordonnées selon la touche du clavier appuyée. En effet, la classe Etat contient quatre méthodes qui vont gérer le déplacement : moveUp(), moveDown(), et moveLeft() et moveRight(). Ces méthodes vont être utilisées dans la méthode KeyPressed de la classe Control qui va appeler l'une des quatres méthodes selon la touche pressée.
La classe Control implémente l'interface KeyListener pour la gestion du clavier comme on peut le voir dans le diagramme suivant :

Diagramme de classe présentant l'implémentation de l'interface KeyListener dans la classe Control :



On peut également observer l'ensemble des interractions entre les classes sur un diagramme plus général :



L'animation de la route infinie se fait par le biais de la classe Avancer. En effet, la classe Avancer hérite de Thread qui permet de réaliser de la programmation concurrente. Avancer hérite de la méthode run() qui va permettre d'actualiser la route en un intervalle de temps fixe. La méthode run() appelle updateRoute() dans la classe Route pour mettre à jour les points qui composent la route.
Cette méthode met à jour les coordonnées des points qui composent la route et s'assure de la création du dernier point ce qui permet de donner l'impression que la route est infinie.

Diagramme de classe présentant la classe Avancer qui hérite de Thread:

Nous avons commencé a stylisé la route à l'aide de courbe de Bézier en utilisant la bibliothèque AWT qui définit les classes `QuadCurve2D` et `CubicCurve2D`.
Un QuadCurve2D nécessite 3 points : le point de début, le point de contrôle et le point de fin; le point de contrôle permet d'ajuster la courbe entre le point de début et de fin ce qui permet de donner la forme arrondie. Ainsi, nous avons utilisé ce principe sur les points qui constituent la courbe. Pour qu'il y ait un effet de continuité entre deux courbes du parcours, on prend les 3 points successifs du parcours puis on relie le point de la première moitié à celui de la deuxième moitié avec comme point de contrôle le point qui se situe entre les deux. On complète ensuite le parcours, en traçant le segment entre le point de la dernière moitié au dernier point du parcours.

Pour implémenter un décor dynamique, nous avons commencé à afficher un oiseau qui se déplace le long de l'écran. l'image de l'oiseau est une image gif qui a été découpée en 8 images et qui implémente un thread. Ce thread va permettre d'afficher successivement les images pour animer cet oiseau. La classe VueOiseau qui permet d'afficher l'oiseau est créée dans le constructeur de la classe Affichage.

Ce décor dynamique est accompagné de points de contrôle qui vont permettre de créditer le temps restant aux véhicules. Les points de contrôle sont créés dans la classe Route par le biais de la méthode createCheckpoint(). Dans notre application, ces points de contrôles sont représentés par deux points dans une ArrayList appelées Checkpoints. Ainsi, la liste de points n'est composée que de deux points qui vont ensuite être reliés dans la classe Affichage pour former un point de contrôle.

La position du point de contrôle est mise à jour avec la méthode updateCheckpoint() qui supprime les points lorsqu'elles dépassent de l'affichage.

```
/** Mise a jour des points de controle*/
updateCheckpoint()
    //Verifie que la liste n'est pas vide
    if(checkpoints not empty)
        //L'ordonnée des points se déplacent en meme temps que la route
        Checkpoints[0].y += Affichage.getMove()
        Checkpoints[1].y += Affichage.getMove()
        //La liste est vidée quand les points dépassent l'horizon
        if (Checkpoints[0].y > HEIGHT())
            Checkpoints.clear();
```

Après une certaine distance parcourue, deux nouveaux points sont ajoutés à la liste pour former un nouveau point de contrôle. Cette appel à createCheckpoint() se fait dans le thread avancer et qui selon la distance parcourue par la voiture, va appeler cette méthode.

Pour implémenter le temps restant à la voiture pour atteindre le prochain point de contrôle, nous avons créé la classe Chrono qui implémente un Thread. Ce thread va permettre d'initialiser un temps et de le diminuer chaque seconde. Une fois que la voiture passe un point de contrôle, le temps est crédité. Cela est vérifié dans la classe vérifié qui compare l'ordonnée de la voiture à celle du point de contrôle et crédite le temps si l'ordonnée de la voiture est inférieure.

Nous avons aussi commencé à implémenter la gestion de la vitesse de la voiture. La vitesse est contrôlée par rapport à cette proximité à la route. Pour l'instant nous avons fait en sorte que la voiture voit sa vitesse augmenter si elle s'approche du milieu de la fenêtre, à peu près là où se situe la route. Par la suite, nous allons faire en sorte que la voiture augmente sa vitesse si elle se situe vraiment au milieu de la route.

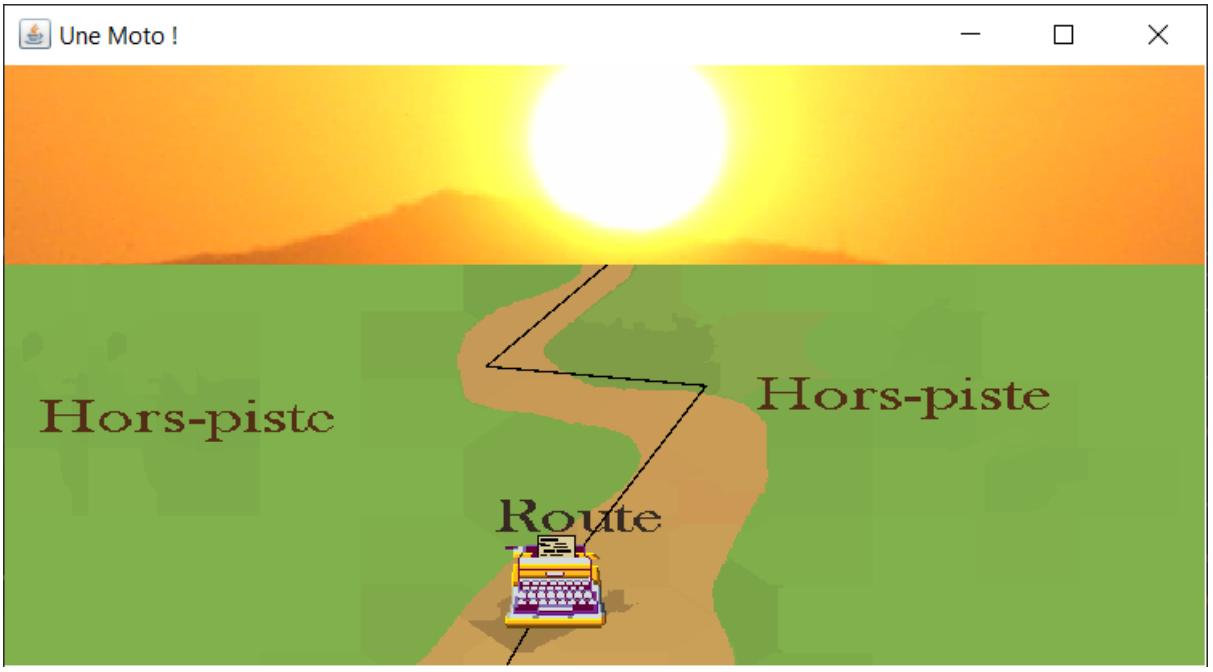
La classe obstacle permet la création des obstacles de l'environnement. Un obstacle possède comme attribut : une position représentée par un point, une image avec sa largeur et sa hauteur, et un booléen visible qui indique si l'obstacle est affiché à l'écran. Les obstacles sont réunis dans la classe Route dans une ArrayList. Ils sont créés par le biais de la méthode createObstacle() qui crée une instance d'obstacle et l'ajoute à l'arraylist.
La méthode updateObstacle() sert ensuite à actualiser leur position pendant l'avancée de la moto. Cette méthode est appelée dans la méthode du thread Avancer et modifie les coordonnées des obstacles.

La collision entre un obstacle et la moto est gérée dans la classe Etat par la méthode CheckCollision(). Une image du jeu est définie par un rectangle et en récupérant pour chaque image le rectangle associé, on peut vérifier la collision de

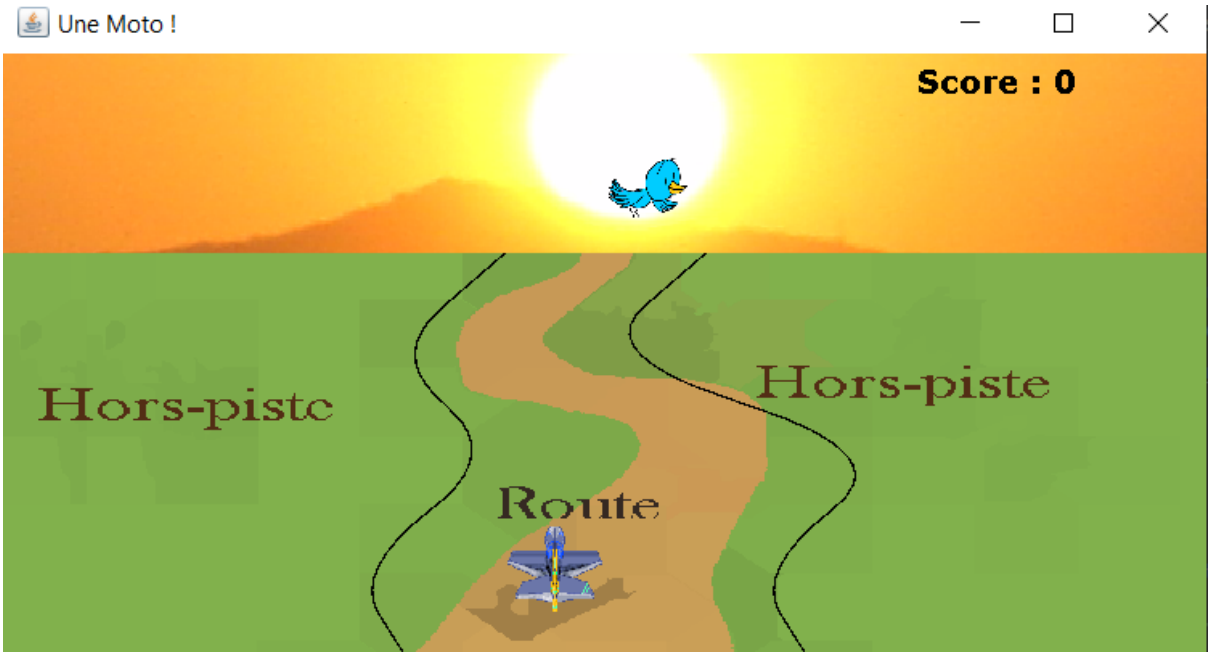
deux rectangles par le biais de la méthode intersect(). Ainsi, le rectangle associé à la moto est récupéré puis on teste si elle entre en collision avec l'un des rectangles associés aux obstacles. S' ils entrent en collision, la variable booléenne visible de l'obstacle devient false et n'est plus affichée à l'écran.

Résultat

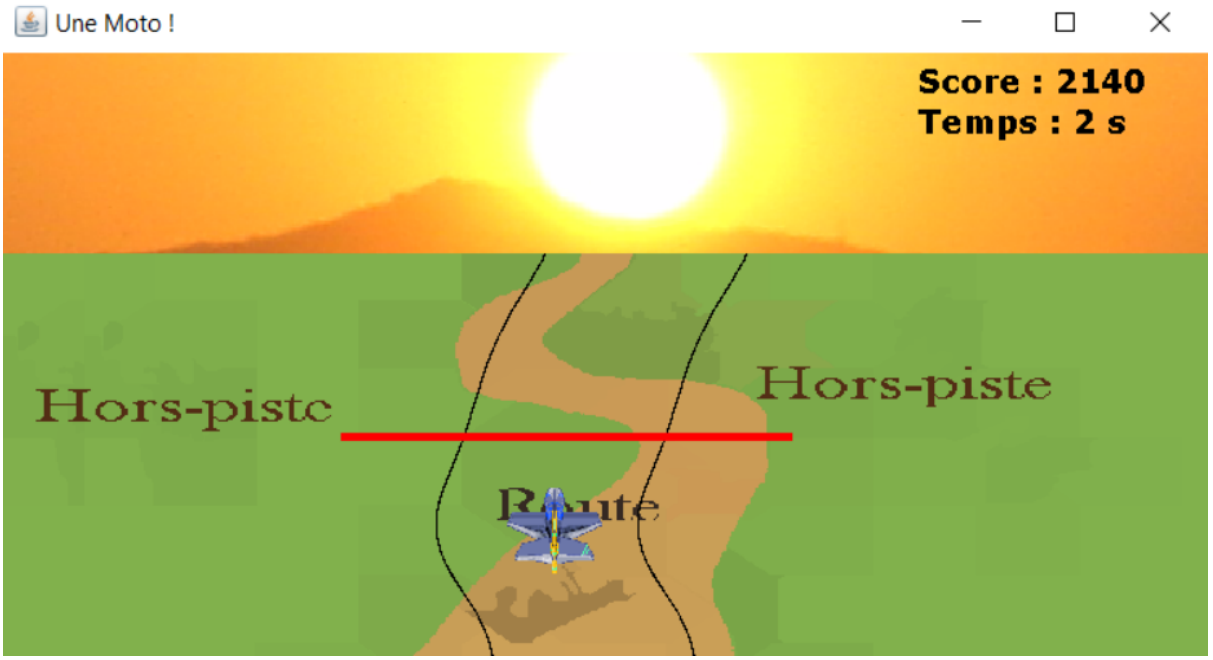
L'image ci-dessous montre notre première version du jeu avec l'affichage de la fenêtre lorsque le programme est lancé, on y retrouve la moto au centre avec son environnement que nous allons implémenté au fil des séances.



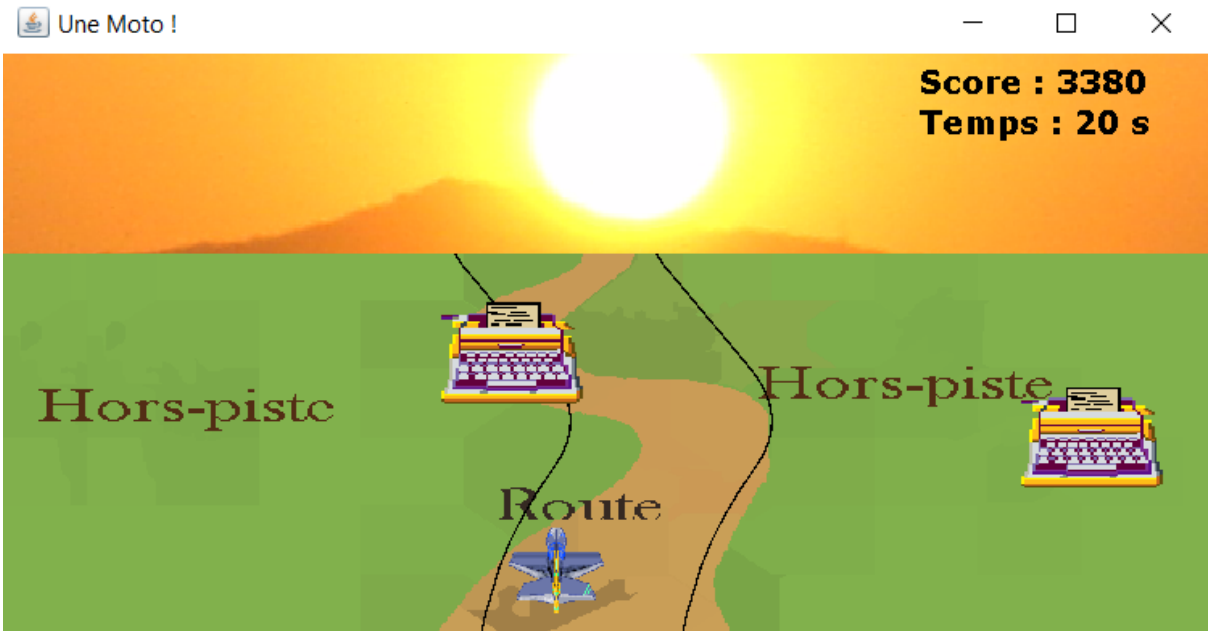
Le premier élément de l'environnement que nous avons implémenté est la route infinie. On peut aussi voir sur la capture d'écran suivante l'image que nous avons créé pour la moto, l'affichage du premier oiseau dynamique et la stylisation de la route par le biais de courbe de Bézier.



La capture d'écran suivante montre l'affichage du temps restant à la voiture pour atteindre le prochain point de contrôle et l'affichage d'un point de contrôle matérialisé par une ligne horizontale sur la route.



L'image suivante montre l'affichage des obstacles à l'écran avec une image que nous allons modifier par la suite.



Documentation utilisateur

- Prérequis : Java avec un IDE
- Mode d'emploi : importer le projet dans l'IDE, sélectionner la classe Main puis "Run as Java Application"
  Une fenêtre s'ouvre avec l'écran titre, appuyez sur le bouton start pour lancer le jeu. Une fenêtre s'ouvre avec la moto au centre, utiliser la flèche du haut pour lancer le jeu et les flèches directionnelles pour déplacer la moto dans son environnement.

Documentation développeur

Les classes principales à regarder sont celles qui implémentent le schéma MVC : Affichage, Etat, Control. La classe Main permet de lancer le programme.

Les principales constantes que l'on peut modifier pour changer le fonctionnement du code sont, tout d'abord dans la classe Affichage, la taille de la fenêtre de jeu défini par les constantes "WIDTH" et "HEIGHT". Les constantes liées à la moto peuvent aussi être modifiées, notamment sa largeur et hauteur définis respectivement par les constantes "largeurMoto" et "hauteurMoto" et de combien de pixels elle peut se déplacer définie par la constante "move".
Dans la classe Etat, la position initiale de la moto peut être modifiée en changeant les coordonnées de la constante "pos".
Dans la classe Oiseau, le déplacement de l'oiseau peut être changé en modifiant la variable délai: plus le délai est faible plus l'oiseau se déplace rapidement dans la fenêtre. Sa hauteur et sa position peuvent aussi être modifiées avec respectivement

les variables hauteur et pos.
Dans la classe Chrono, le temps initial donné à la voiture pour atteindre le prochain point de contrôle est défini par la variable start qui peut être augmentée ou diminuée pour ajuster la difficultée du jeu.
L'image d'un obstacle peut être modifiée dans la classe Obstacle en changeant le fichier stocké par la variable image.

Les prochaines fonctionnalités seront la création des concurrents que la moto doit dépasser.

Conclusion et perspectives

Nous avons donc commencé à créer l'environnement de jeu avec, tout d'abord, la moto et son déplacement dans la fenêtre d'affichage, et l'affichage de la route infinie qui permet de donner l'effet que la moto avance dans la fenêtre. Cette route fait partie du décor que nous allons compléter au fil des séances.
Le décor dynamique s'est vu compléter avec la création d'un oiseau parcourant l'écran et la création de points de contrôle.
On pourra voir s'ajouter à ce décor les concurrents représentés par d'autres motos.

Nous éprouvons encore des difficultés à donner un effet de profondeur à cette route, c'est un problème que nous espérons résoudre sous peu.