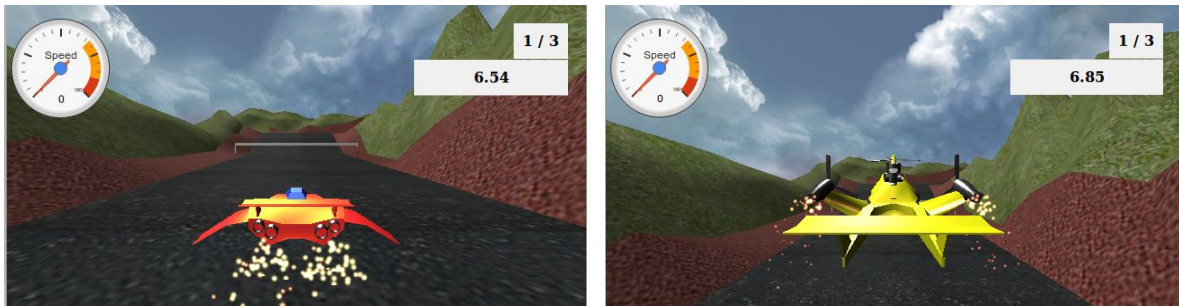


Modélisation Animation Rendu



Antoine Ferey
June Benvegnu-Sallou
Emmanuel Loisan

<https://github.com/eloisance/Project-Race>



Dans le cadre de l'UE M.A.R, nous avons eu à développer un jeu de course à partir d'une base existante en utilisant la librairie ThreeJS.

Commandes

- Touche "espace" → saut
- Touche "backspace" → reset de la partie
- Touche "p" → changement de mode de caméra
- Touche "n" → changement de véhicule + reset de la partie

Caméras

Nous avons implémenté les caméras fixes en fonction des plans. A chaque fois que le véhicule change de plan, une nouvelle caméra prend le relais. Nous avons donc une variable qui stocke pour chaque plan la position de la caméra.

La caméra fixe étant déjà implémentée, nous avons juste à switcher entre les deux modes.

Gestions des évènements

Saut

Cette fonctionnalité a été implémentée en augmentant momentanément l'altitude à laquelle se situe le véhicule. Après un délai d'une seconde, le véhicule reprend son altitude standard vis-à-vis du terrain de course. Nous voulions améliorer cette fonctionnalité en la rendant plus réaliste via l'utilisation d'un mouvement plus fluide mais n'avons pas pu faute de temps. Le véhicule aurait décrit un mouvement parabolique pour augmenter son altitude puis redescendre.

Reset

La fonctionnalité du reset était intéressante à implémenter. Il fallait savoir quelle variable correspondait à quelle caractéristique du véhicule. Il fallait réinitialiser la position, la vitesse et l'angle du véhicule.

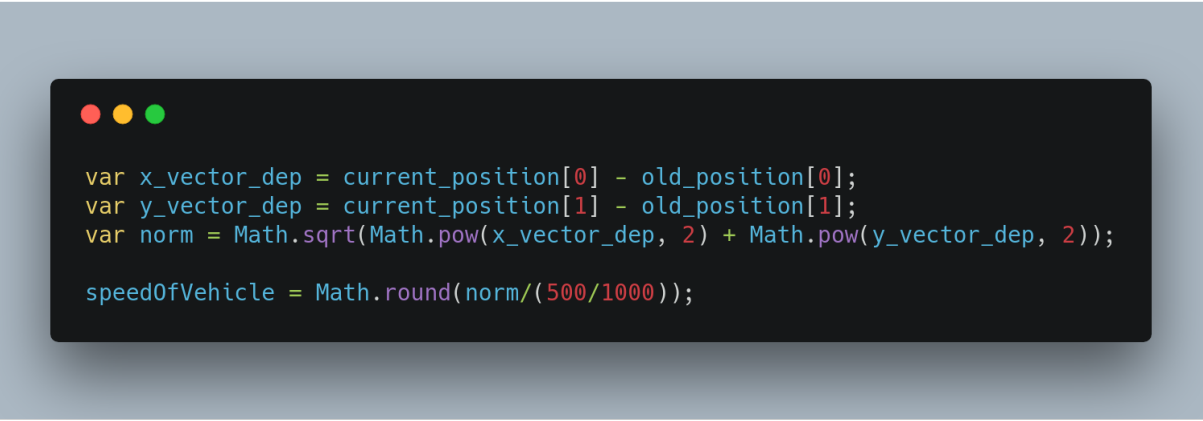
A chaque nouvelle fonctionnalité, il fallait compléter le reset.

Compteur de vitesse

Pour déterminer la vitesse du véhicule, nous avons, à chaque intervalle de temps de 500ms, calculé la norme du vecteur de déplacement du véhicule à l'aide des positions de ce dernier dans le repère du terrain.

La vitesse affichée n'est pas réellement en km/h mais en unité arbitraire par rapport aux positions du terrain (NAV) et en seconde.

Pour l'affichage graphique du compteur de vitesse, nous avons utilisé la librairie "Charts" de Google et notamment les [Gauge Charts](#).



```
var x_vector_dep = current_position[0] - old_position[0];
var y_vector_dep = current_position[1] - old_position[1];
var norm = Math.sqrt(Math.pow(x_vector_dep, 2) + Math.pow(y_vector_dep, 2));

speedOfVehicle = Math.round(norm/(500/1000));
```

Compteur de tour

La validation d'un tour se fait une fois que le véhicule est passé par l'ensemble des checkpoints définis qui correspondent aux différentes parties du circuit. Nous avons choisi

les plans "0, 1, 5, 10, 14, 21, 29". Lorsque le véhicule se déplace sur le circuit, l'application enregistre chaque nouveau plan rencontré, si à l'arrivée l'ensemble des checkpoints ont été rencontrés, on peut considérer que le joueur a bien fait le tour complet du circuit.

Mauvais sens

A partir de la fonctionnalité du "compteur de tour", il était simple de savoir si le joueur allait dans le bon sens du circuit ou non, en vérifiant à chaque changement de plan que celui-ci (son numéro) était bien supérieur au précédent. Dans le cas contraire, le javascript modifie la valeur css "display" du bandeau affichant le message.

Compteur de temps par tour

Une fois la fonctionnalité du compteur de tour réalisée, il était facile d'implémenter un Timer pour afficher les temps pour chaque tour. Il faut juste penser à réinitialiser l'affichage et les variables lors du reset.

Hélicoptère

Construction du graphe de scène

Il a fallu construire le graphe de scène correspondant à l'hélicoptère à partir des différentes assets mis à disposition : corps de l'hélicoptère, axes, pales et turbines.

En attachant avec ThreeJS les différentes parties les unes aux autres :

- Les pales aux axes
- Les axes aux turbines
- Les turbines au corps de l'hélicoptère

Cela nous a permis par exemple de seulement avoir à faire tourner l'axe pour que les 3 pales se mettent à tourner et de pouvoir positionner chaque élément par rapport à son parent.

Rotation des turbines

Nous cherchons à faire pivoter les turbines lorsqu'on tourne à gauche ou à droite. Lorsqu'on ne touche plus les commandes, les turbines doivent se remettre à leurs positions initiales. On réalise ce mécanisme via un *setInterval*.

Tant que l'utilisateur appuie sur l'input pour tourner, on augmente (ou diminue) la rotation Z de l'objet jusqu'à un certain point. Si l'utilisateur lâche l'input, on réduit (ou augmente) cette même rotation jusqu'à la position initiale, c'est-à-dire 0.

La difficulté ici était la combinaison des deux *setInterval* (droite et gauche). Il arrivait souvent que la rotation des turbines sorte de la limite fixée.

Rotation sur l'axe X

Pour l'hélicoptère uniquement, la vitesse fait varier son angle de rotation X afin de donner un effet de déplacement plus réel de celui-ci.

Systèmes à base de particules

Pour mimer la présence de feu / fumée à la sortie des pots d'échappement de la voiture ou des turbines de l'hélicoptère, nous avons eu recours à des systèmes à base de particules.

Dans le cas de la voiture, un seul système a été utilisé mais comprenant deux émetteurs : un pour les pots d'échappement à gauche du véhicule et un autre pour la partie droite. Le système est attaché à la voiture et les émetteurs positionnés de sorte à correspondre aux pots d'échappement.

Pour l'hélicoptère, deux systèmes à base de particules ont été créés avec pour chacun un seul émetteur. Chaque système a été "accroché" à une turbine (gauche ou droite) puis positionné de façon à ce que les particules émises via l'émetteur apparaissent à l'extrémité de la turbine. Cela a permis l'émission des particules suivant le mouvement des turbines, comme par exemple dans une situation de virage.

Problèmes rencontrés

Appropriation et compréhension du code existant : nous avons dû prendre en main le code existant et le comprendre ainsi que découvrir la bibliothèque ThreeJS avant de pouvoir commencer à implémenter de nouvelles fonctionnalités. Le nombre important de variables rendait ce travail de compréhension laborieux au premier abord afin de comprendre correctement le rôle de chacune d'elles.

Lenteur de l'application : en ajoutant les différentes fonctionnalités, nous avons constaté au fur et à mesure des développements une lenteur de l'application qui s'explique par l'augmentation du nombre de calcul que le navigateur doit traiter. Une meilleure gestion des boucles et intervalles pourrait permettre de réduire cette lenteur.

