Compte Rendu
Informatique Graphique

Nicolas Wallet
L3 Informatique

Mai 2020

Table des matières

Abstract

Les fichiers ci-après évoqués sont disponible sur le lien Git suivant : Github.com Il suffit de cloner le dépot.

Une version en ligne est aussi disponible. (ici)

Par convention les noms des variables et fonctions sont en anglais.

Si vous le souhaitez vous pouvez voir les élement au fur et a mesures, un fois node lancé, en bas de la pages dans la section "Notes" , cliquez sur "Demonstration des composants".

Introduction

Plusieurs objectif pour notre petit jeu : Cote Client Avoir une
— Une voiture qui gère une pyshique de base (accélération, freinage)
— Un route (délimitant les limites)
— Réapritation au dernier point de controle (respawn / checkpoint)
— Génération de carte
Interface graphique (GUI)
— Nombre de tours
— Temps du dernier tour
— Temps au meilleur tour
— Vitesse actuelle
— (M) Liste des joueurs

1 Three JS

Pour l'ensemble des élèment j'ai choisit un style 'low poly', c'est à dire avec peu de faces, c'est simple à réaliser, ça garantie certaines performances.Donc de manière générale j'utilise du flatShading. Pour les ombres , les lumière et element au dessus du sol peuvent castShadow et les autres element reveiveShadow.

1.1 Segement

On pourrai croire que l'élément a réaliser en premier est le joueur (ie : notre voirure) mais vu qu'on veut restreindre dans la plupart des cas le joueur à la route , le mieux est de créer un outils pour créer une route sans trop se prendre la tête, en plus on pourra faire plusieurs circuits différents sans trop de difficulté.

Une 'Map' c'est quoi ?

Une route definit par des coordonée, on va definir plusieurs type de voies (Pour le moment route) Plusieur type de formes 'shape' (ligne droite , intesertion , courbe) Des checkpoints pour verifier que le joueur passe au bon endroit, mais surtout pour respawn.

Le visuels 'route' sera le meme pour chaques type de route.Une bordure couleur beton.Un sol couleur Bitume , et une ligne de signalisation blanche au milieu de la route.On utlisera des PhongMaterial de couleur.

Toute nos route seront de la même largeur pour permettre une unifomrité.

1.1.1 Ligne droite

Notre premier segment est une ligne droite (Linear).C'est le plus simple. Le segment est definie avec une longeur(length).On commence avec un plan pour notre sol.Deux geomtryBox pour les cote et la ligne droite sera un plan.

1.1.2 Virage

C'est le plus compliqué , on pourrait utilisé RingGeometry pour le sol, mais on souhaite gardé nos bordures sur les côtes , or aucune geométry ne correspond.Les seuls qui correspond un peu sont TOrus et Tube. Mais j'ai aussi la possible d'utiliser Shape, qui est beaucoup plus flexible.Et shape fonctionne un peu comme l'API Canvas 2D.Je peux donc utliser exactement les meme forme pour le sol ou les cotes.

Voir figure (1)

1.1.3 Bezier

L'idée c'est d'avoir des lignes droite avec un décalage, voir Figure 2 Annexe A . On reprend le code précends , mais maintenent on a deux points intermedaire. Notre segment bezier est définit par une longueur et un décalage (positif vers la droite et negatif vers la gauche)

Voir figure (2)

```
Shape.bezierCurveTo(x1,y1,x2,y2,x3,y3)
```

1.1.4 Boost

Le boost est une ligne droite.Elle à deux particularité , la prémière visuelle , on utilisé les vertexcolor pour mettre certaine face en jaune. La deuxième c'est qu'on doit ce souvenir que ce segment en particulier permet au joueur d'avancer plus vite.Plus tard on permettera au joueurs de connnaître le numero du segment sur lequel il route, j'aouter donc a une variable globale le numero de segment boost.

1.1.5 Points de contrôle

Mes points contrôles sont surtout des segments utilaires, je vais devoir retenir de ces block.Mais cette gestion ce fait dans la partie suivante.Niveaux viselle c'est une simple route mais avec une ? ? ?

1.2 Carte

On a pas mal de composant nessecaire à la création d'un circuit.On va créer une classe qui va permettre de gérer notre circuit.

```
Class Carte{...}
```

Evidement chaque instance de 'Carte' sera monde à part entière. Une carte sera composé de Segments.Ma 'Carte' doit faire en sorte de placer chaque segment de manière à ce qu'ils forment un circuit (ie : sur j'ajoute deux segments lineaires, un premiers de tailles A et le seconde de taille B , ma carte commence

en (0,0) on obtient segmemet1(0,0) et segment2(0,A)) Ici c'est assez simple à gere , mais ma 'Carte' doit gérere tous mes composants (aussi mes virages) . Ceci implique de calculer le points après un virage , et de changer l'orientation du segments suivants.

Concentrons nous sur nos virages, à partir d'un point (0,0) et d'une taille SIZE , d'une côté SIDE, et d'une orientation FACING.Comment définir le points suivant ? (cf fichier geogebra.html)

$$A(0,0)$$

$$F = (FACING + SIDE)90\pi * /180$$

$$B(SIZE * (cos(F) + sin(F), SIZE * (cos(F) - sin(F))$$

Il y a deux cote (droite ou gauche) et 4 orientation .C'est donc 8 cas mais seulement 4 points

1.3 Terrain

Terrain low poly. Je pars d'un plan.Je vais simplement modifier les positions des vertices en y et les couleurs des faces (et donc le material devra utiliser les vertexColors.). Quand je génère ma carte, je défini des AABB box a partir des segment. (Ce sont les Box3). Je commence par modifier les sommets.Si un sommet se trouve sur un des segment alors je n'y touche pas , si en revanche il et en dehors et change son y. J'ai jouer un peu avec les paramètre pour trouver quelques choses de correct.Pour générer mes montagnes je calculer le sommet 'y' qui est en dehors de tout segment AABB en fonction de sa distance minimal avec les segments. J'ajoute des facteurs d'atténuations et d'aléatoires.

De la même manière je veux sur certains terrains ajouter des lacs , qui sont des montage 'inversé' beacoup plus atténuer.Je passe en paramètre une Box3 et si la vertice et dans ce points on applique les positions pour créer un lac.Pour lac je rajoute une plan bleu translucide à hauteur -1 (ma carte est à 0).

Edit : J'ai essayé de faire un segment tunnel, je me suis qu'en ne l'ajoutant pas le terrain pourrai generer de la montagne.(Voir annexe).Le problème est que on voit certaine face au entré du tunnel lorsqu'il y a une montage. J'ai pensé au clipping mais au mieux c'est avec des plan locaux donc pas simple à gerer.Ou j'avais aussi pensé à retire les faces qui sont à l'interieur tunnels mais ça pose des problème de repositionnement des sommets adjacents. Voir image 3

1.4 Une Voiture

Pour la voiture, un model libre de droit fera l'affaire.Le model est composé d'un .obj (geometry) et un .mtl (material). Les deux fichier sont lié il doivent être chargé en même temps (mtl puis obj mais en réalité) Il y a des exemples dans la documentation pour tout les loaders mais pas de mtl et à la fois obj , j'en ai trouvé un ici : Hofk.de

Pour le déplecement de la voiture , on accéde aux positions et rotation via l'objet (de la scène) et on déclare une variable avec vitesse , accélération , vitesse max etc... On fonction des controles du joueurs on change les valeur de la variable , vitesse ou rotation , puis on calucler la nouvelle position en fonction de la variable.

Pour restraindre les deplecement à la carte , j'avais dans un premier temps utiliser les Box3 mais pour les bloc curve et bezier c'est assez genants.J'ai donc chercher un peu et trouver le raycaster. On definit un liste de mesh qui avec lequelle on cherche une collision.On créer une THRREE.Raycaster() qu'on definit avec un point d'origne à partir de notre voiture.A partir d'une collision on connait un vecteur qui va nous permettre de connaitre l'angle de collision.

Une fois qu'il y a collision je regarde la distance avec la collision précedente :

```
 * Je garde la distance avec la collision calculé précedente
 * Si l'ancienne est plus grand c'est mon joueur essaye de quitter la collision
 * càd il s'eloigne , alors je le laisse repartir
 * sinon je l'arrete
 */
if(old>collisionResults[0].distance)
        car_stats.speed=0
old=collisionResults[0].distance
```

2 Environnement

2.1 Lampadaire

Proche de nos route

Pour la lumière, un THREE.Spolight avec un angle réduit, une distance légèrement supérieur au lampadaires , et en fonction de l'intensité désiré on ajuste de le decay.

2.2 Arbres et Buissons

Vu qu'on veut faire du low poly, on peut utiliser des geometrie très simple fourni pas THREE.JS. On ecrit donc une fonction qui pose un arbre en x,y,z fourni en paramètre, notre arbres est un objet3D composé d'un tronc et de feuilles.Le tronc sera un cylindre marron avec un rayon sup $<$ rayon inf. Les feuilles seront un dodécaèdre car sphere faisait trop boule même avec peu de segments.

En instanciant plusieurs arbres à la suite je me suis apercu qu'on voyait très rapidement qu'il etait tous identiques.J'ai donc ajouter des variations pour que ça soit plus jolie.Plusieur couleur pour les feuilles , dont j'applique aussi une rotation Math.random(). Dans la même idée je fais une souche d'arbre en tant que variation .

Les buissons seront des dodécaèdre avec les variations de couleur et de rotation. Et le nombre de buisson varie aussi.

2.3 Forêt

J'en profite pour faire un fonction 'forêt', qui génère un nombre important d'arbres et de buissons.

Vous pouvez voir les différents resultats dans 'Demonstration des composants' $>$ 'Environnement' $>$ Arbres, buissons et forêts.

3 Login et GUI

3.1 Page de connection

Cette page permet rentrer un pseudo et la couleur de sa voiture.

```
document.querySelector('#log').addEventListener('submit', event => {
    // On initialise le rendu THREE JS
    init()
    animate()

    // Toutes les elements de vont être initialisé pour communiqué avec le serveur.
    // Pour socket.io voir Chapitre 2
    socket=io()
    //...

  })
```

3.2 Statistiques

Comme définit en introduction on veut connaître notre temps au meilleur tour et dernier tour.On créer une qu'on place dans un coin (haut-gauche). On utlisera un systeme de checkoints pour s'assurer que le joueur suit la bonne route (ie : dans le bon sens) et on prendra le temps THREE.clock.

Pour le temps tout les méthodes sont définit dans la doc (Clock

Imaginons notre circuit comme suit : Il nous faut impérativement (pour cette solution) un minimum de 3 checkpoint pour valider de le tour.Cette méthode s'applique dans le cas d'un circuit (ei : depart = arrivé , et départ = checkpoints) .

A chaque fois que le joueur passe un cp on le met à 1 . Si le joueur et au départ qu'il recule au dernier cp alors on ne doit pas le mettre à 1 (car le second cp est encore à 0).

Si tout les cp sont 1 et que nous sommes au départ alors on a fait fait un tour, on peut rester la variable et on recommence si le nombre de tour et différent de celui définit par la carte.

3.3 Contrôles

Si l'utilisateur ne souhaites pas utiliser les touches par defaut il doit pouvoir les changer.On va passer par une variable qui contient les touche associé pour chaque opération.

```
KeyBind = {
    avancer : 'z',
    camera : 'a',
}
//mov est notre tableau de touches qui sont 'keydroum'

if( mov[ KeyBind.avancer ] ) //equivaut a if ( mov['z'] )
    //avancer la voiture
```

Quand notre joueur change de touche, on actualise notre objet KeyBind.En pratique on va garder l'event.key et l'event.code dans keybind.Le code pour les conditions et key pour l'affichage (parce que code ne dépand pas du clavier EN/FR - alors que key oui). Si l'utlilisateur change de carte / refresh il perd sont keyBinding. Pour bien faire , l'idéal serait d'avoir des comptes utlisateurs pour stocker le keyBinging ou d'utiliser le localStorage.

4 Server

Le principe est de mettre le jeu en multijoueur, pour un peu de compétition. Chacun peu rejoindre une carte quand il le souhaite.

Comme cette partie n'est pas demandé et ce n'est pas du THREE.JS je vais expliquer seulement le principe. J'utilie socket.io() pour la connexion client-server.

Nous avons plusieurs carte donc si un joueur se connecte sur la carte 1 il ne doit pas voir un joueur sur la carte 2. Socket.io propose deux solutions pour separer les connexions , les Namespace qui créer un channel de connexions pour chaque namespace et les rooms qui qui permettent de diviser les namespace.Les rooms sont plus adapté à nos 'serveur'. Lorsqu'un joueur lance une carte il se connecte au server (socket.io) en lui donnant le numéro de la carte qu'il veut se connecter.Le server l'ajoute alors a la room.Le server envoie la list des joueur ayant dejà rejoins la room (et le client les affichera).

Lorsque qu'un client se deplace il envoie sa position au server qui la distribue aux autres joueurs de la room ('emit' envoie ajouter les joueur et 'brodcast' l'envoie a tous sauf la personne qui l'a envoyé , et to(X) permet de l'envoie à X, X peut etre une socket ou une room)

Lorsqu'un client se deconnecte , le serveur en informe les joueurs de la room qui le supprimerons.

Voir image 4

4.1 Cacher les joueurs

Ajout de fonctionnalité : On veut pouvoir cacher les joueurs , j'ajouter une touche dans KeyBinding. Étant donnée qu'on a une liste d'id de tous les joueurs présents, on met leur visibilité à l'inverse lors du triggers.

5 Amélioration

Quelques amélioations possible mais pas réaliser par manque de temps.

Améliorer l'editeur de carte de manière à pouvoir ajouter les objets (arbres, foret , etc...) directement (et via un gui). Avoir une base de données pour enregistrer les temps de chaque joueurs.

Il y a aussi du code que j'ai fourni mais pas implémenter.Comme le cube, et la mine,ou encore le lampadaire .Je n'ai pas terminer leur implementation.

6 Sources

Route Center Origine : StackOverflow (pas exactement cette techinque que j'utilise mais ça m'a aider)fa Anton Moek pour la voiture low poly (fichiers OBJ et MTL)

Exemples sur le Raycaster :

— Lee Stemkoski

— Klaus Hofk

Les différents graphiques annexes sont réalisé à l'aide du site GeoGebra et de l'outils path SVG CodePen.io

A Annexe

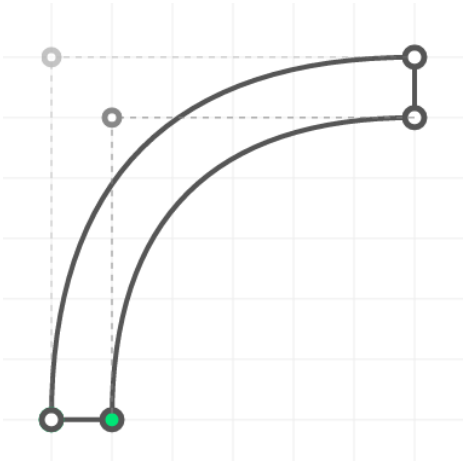

FIGURE 1 – Bezier 1 point (Quadratique)

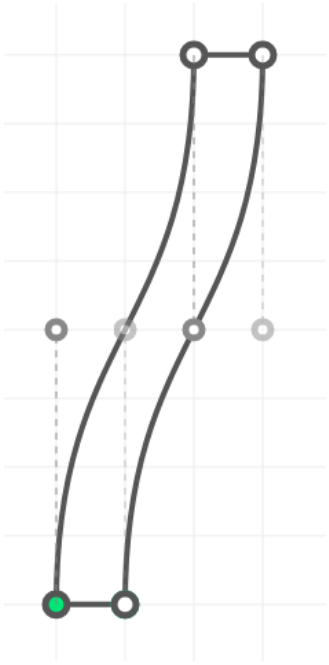

FIGURE 2 – Bezier 2 points

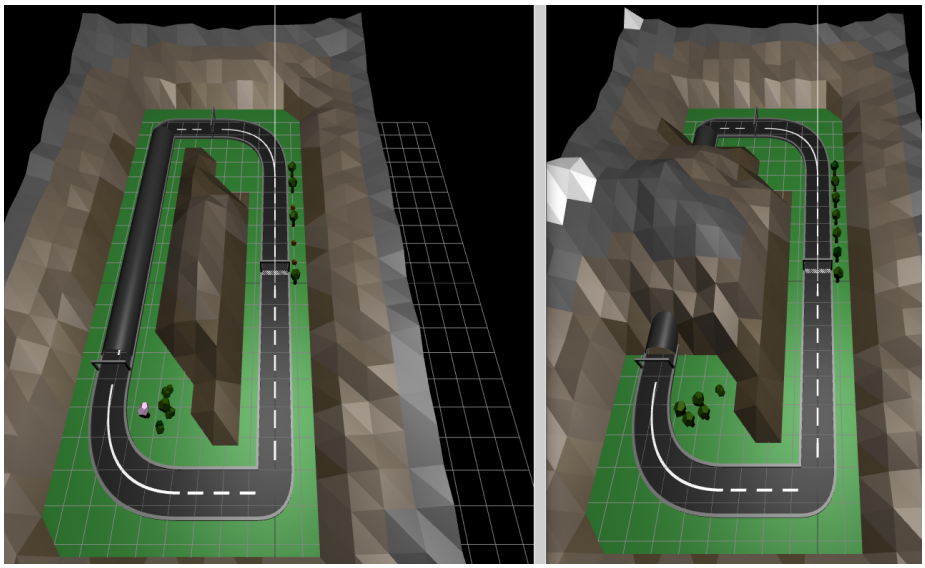

FIGURE 3 – AABB road sur tunnel pour la génération du terrain

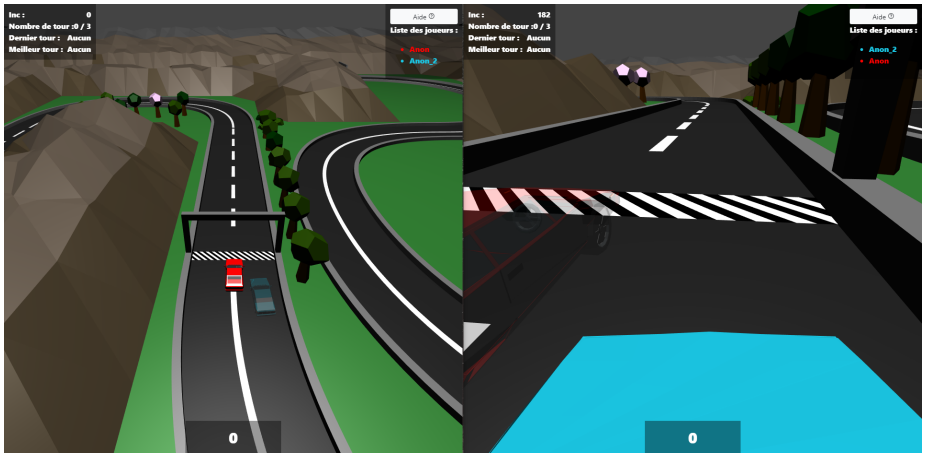

FIGURE 4 – POV de deux joueurs