ANNÉE 2023

3292.2 Outils d'infographie – Rapport technique – ISC3il-b

Ride-Safe

Projet d'infographie à l'aide de Unity

Étudiants participant à ce travail :

Nicolas Aubert, ISC3il-b
Théo Vuilliomenet, ISC3il-b
Vincent Jeannin, ISC3il-b

Présenté à :

Benoit Le Callennec

Restitution du rapport : **12.05.2023**

Période : **2023**

École : **HE-Arc, Neuchâtel**

haute école arc ingénierie
neuchâtel berne jura www.he-arc.ch

Table des matières

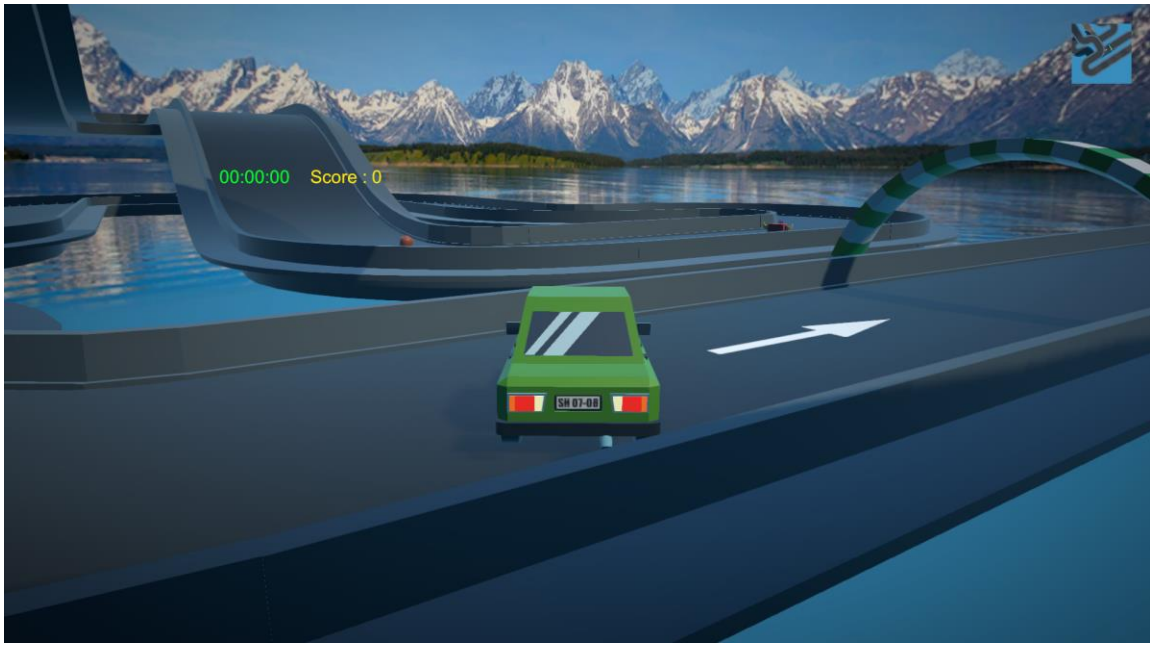
1 - Introduction

1.1 - Description du projet

Nous avons imaginé un jeu de voiture qui sort légèrement de l'ordinaire.

Premièrement, la physique de notre jeu n'est pas habituelle. Contrairement aux autres jeux de voitures, nous avons mis en avant le drift et le manque d'adhérence sur la route. La conduite sera donc plus délicate. De plus, le chronomètre n'est pas le seul objectif. Vous devrez aussi obtenir un score en écrasant un maximum d'entités placées sur votre chemin.

Et pour finir, vous aurez le plaisir de conduire dans un panorama merveilleux qu'offrent les alpes suisses.



1.2 - Objectifs

Un cahier des charges a été établi en début de projet et se trouve sur notre wiki. En voici le lien : Ride Safe CDC. Voici la liste des objectifs principaux :

- Contrôler une voiture à la troisième personne ;
- Circuit à parcourir ;
- Chronométrage du temps de course ;
- Minimap ;
- Paysage sur les côtés du parcours ;
- Génération d'entités sur le parcours ;
- Collision entre entités et voiture.

2 - Implémentation

Ce chapitre détaille les différents aspects techniques de notre projet et donne ainsi un aperçu du travail effectué.

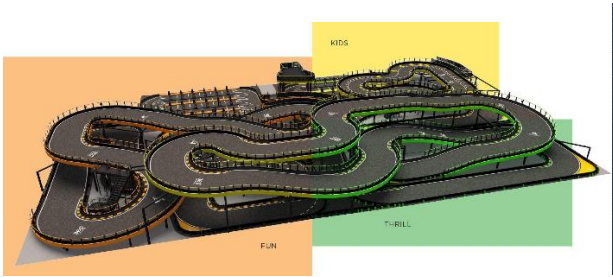
2.1 - Prefab / Assets

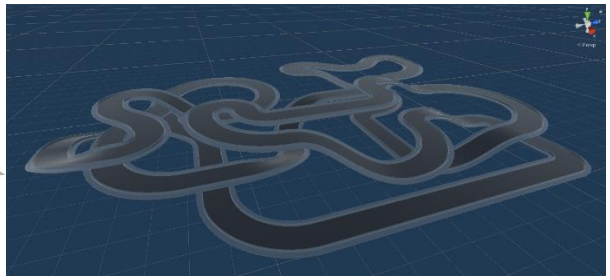
2.1.1 -Circuits

Les circuits sont totalement faits à la main. Pour cela, nous avons été cherchés un set de prefabs sur l'asset store de Unity[1], ressemblant à l'image ci-dessous, puis nous avons construits à la main les différents circuits, et mettant bout à bout les différents morceaux de route.

Malgré une inspiration débordante des chefs bétonneurs que nous sommes pour construire des circuits complexes et originaux, nous avons préféré « perdre » le moins de temps possible à la création des circuits, et c'est pour cela que nous sommes partis sur des circuits existants, trouvés sur internet, et adaptés avec les différents prefabs proposés par l'asset Unity.

Par exemple, une comparaison entre un circuit trouvé sur internet sur la gauche, et sa reproduction adaptée dans Unity sur la droite.





Une fois les morceaux de routes mis ensemble, nous en avons fait un prefab personnel, afin de pouvoir l'inclure beaucoup plus facilement dans la scène principale.

[1] https://assetstore.unity.com/packages/3d/environments/roadways/simple-roads-212360

Nous y avons ensuite ajouté les différents petits personnages et monstres, ainsi qu'une arche et une flèche au sol afin d'y montrer le départ, l'arrivée et le sens de navigation.

Une particularité s'ajoute au dernier circuit. Voulant ajouter une difficulté supplémentaire au dernier niveau, il s'agit en réalité de deux circuits différents, reliés par une intersection à un seul et unique endroit. C'est-à-dire que si par malheur vous choisissez le mauvais chemin à l'intersection, vous ne pourrez explorer que la moitié du circuit, car vous n'aurez aucune autre possibilité de rejoindre la seconde moitié.

2.1.2 -Voiture

Tout comme pour le circuit, nous sommes partis d'un asset trouvé sur l'asset store de Unity pour réaliser notre voiture.

Toutes les modifications liées à la physique sur la voiture ont été réalisées dans le prefab afin de pouvoir facilement la dupliquer sans avoir à modifier ses propriétés en cascade.

2.1.3 -Entités

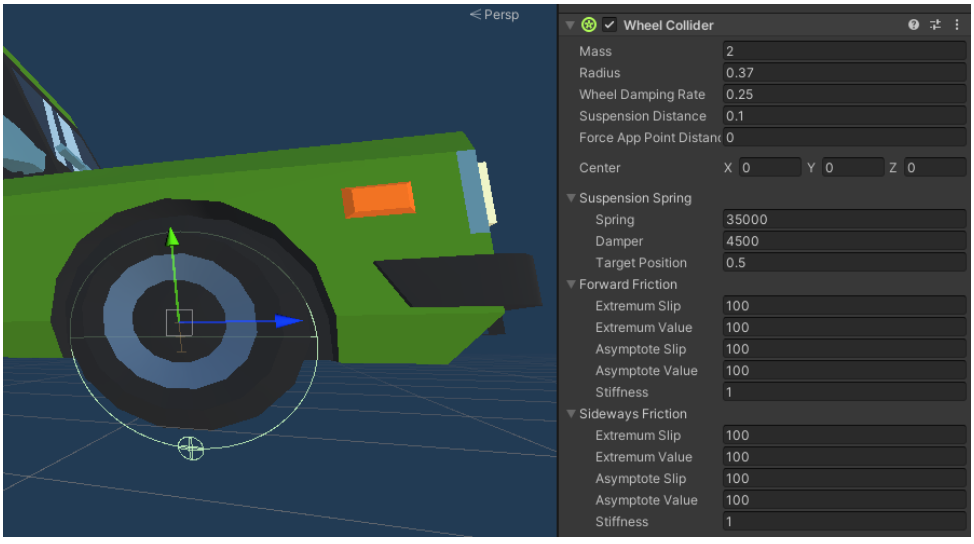
Comme la voiture et les morceaux de route, les différentes entités, qui peuvent aller du père Noël au cyclope de feu, ont été récupérées sur l'asset store de Unity.

Cependant, quelques modifications ont aussi été appliquées aux entités. Nous avons notamment modifié leurs animations 3D et ajouté la gestion des collisions.

2.2 - Physique sur la voiture

2.2.1 -Conduite

Afin d'être au plus proche de la réalité, ce sont les pneus de la voiture qui vont faire avancer la voiture par une force de friction entre les roues et la route. Ceci est possible grâce au Component ***Wheel collider.*** Ce composant permet de gérer la géométrie de la roue et sa physique par le biais de sa masse, ses suspensions et son adhérence.



Pour ce qui est de la conduite, le script ***WheelController*** a été réalisé et considère les entrées utilisateurs pour faire accélérer, freiner et tourner les roues. À nouveau, c'est grâce au composant ***Wheel collider*** que toute la logique des roues est gérée. Voici les trois propriétés utilisées :

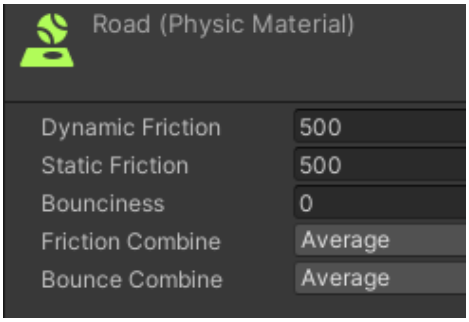
- ***WheelCollider.motorTorque*** : Accélération de la roue ;
- ***WheelCollider.brakeTorque*** : Freinage de la roue ;
- ***WheelCollider.steerAngle*** : Rotation latérale des roues.

Cependant, le composant ***WheelCollider*** n'est pas visuel. Si la roue possède une forte accélération, ou est totalement arrêtée, il n'y a aucune différence visible. C'est pareil pour la rotation latérale.

C'est pourquoi il faut mettre à jour le ***Transform*** de la roue en fonction du ***WheelCollider*** à chaque Update.

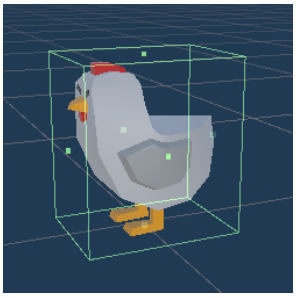
La méthode ***WheelCollider.GetWorldPose*** permet de récupérer la position et la rotation physique de la roue afin de mettre à jour son ***Transform*** dans un second temps.

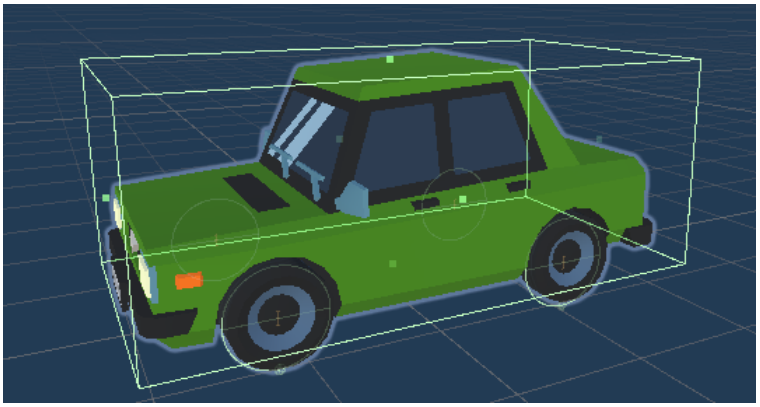
De plus, la roue n'est pas la seule qui doit posséder un coefficient de frottement statique et cinétique. Il y a aussi la route. Et pour ce faire, nous avons assigné à toutes les routes un Material ***Road*** dans le ***Mesh Collider*** des routes.



2.2.2 -Collisions

Afin que notre voiture puisse entrer en collision avec des objets et non passer à travers, il faut lui assigner un ***box collider***. Par la suite, si deux ***Box collider*** se touche, la méthode par défaut ***OnCollisionEnter*** est appelée et permet de gérer la collision.





C'est notamment dans cette fonction que l'on va mettre à jour le score suite à l'écrasement d'une entité et aussi démarrer l'animation 3D de la mort de l'entité dans une coroutine.

```
void OnCollisionEnter(Collision collision)
{
    // Check if the collision is with the car
    if (collision.gameObject.name == "Sedan" && !hit)
    {
        rigidBody.AddForce(new Vector3(0, rigidBody.mass * force, 0));
        hit = true;

        if(scoreScript != null)
        {
            scoreScript.increaseScore(10);
        }
    }

    if (collision.gameObject.name != "Sedan" && hit)
    {
        StartCoroutine("Die");
    }
}
```

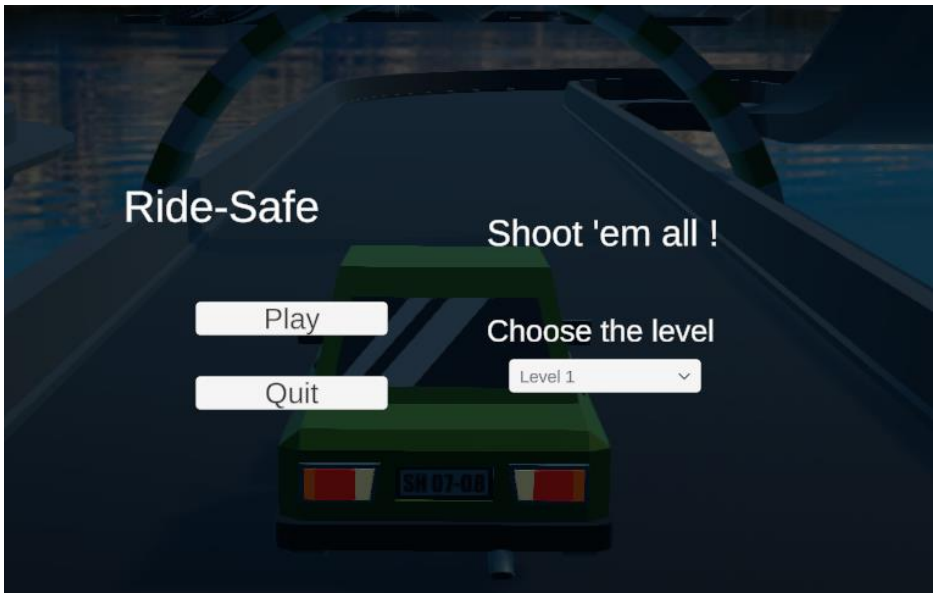
2.2.3 -Phares

Afin de rendre plus crédible notre voiture, nous avons ajouté des lumières rouges au niveau des phares arrières. Ces dernières s'allument lorsque la voiture freine. Pour se faire, il suffit de récupérer les lumières dans un script et d'augmenter la propriété ***Light.intensity*** pour que l'illusion de phares arrières soit bonne.

2.3 - Logique du jeu

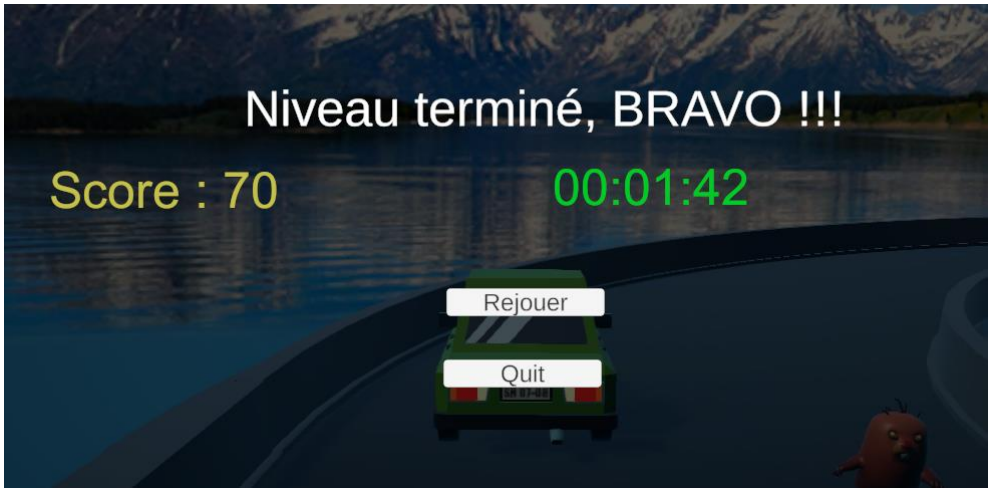
2.3.1 -Menu

Nous avons plusieurs circuits de différents niveaux. C'est pourquoi il nous faut logiquement un menu pour choisir le circuit que l'on souhaite faire.



Un canevas a été utilisé avec des widgets pour le texte, des boutons et un dropdown afin de choisir le niveau. Lors du clic sur le bouton « Play », plusieurs méthodes sont appelées afin de placer la voiture sur le bon circuit et configurer correctement la ligne d'arrivée et fonction du circuit.

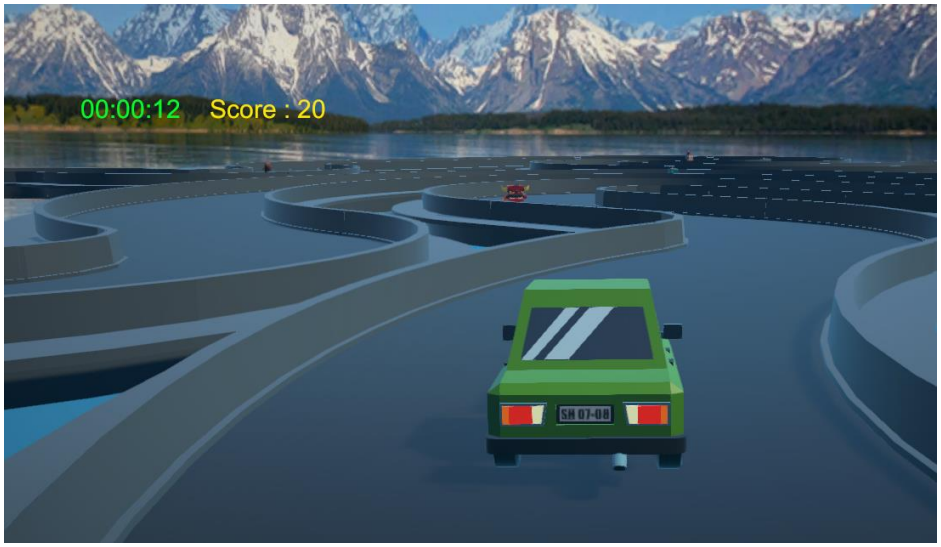
Lorsque vous franchirez la ligne d'arrivée, un autre menu apparaîtra pour vous féliciter et vous proposer de rejouer ou de quitter. À nouveau, une méthode sera appelée pour quitter ou relancer le jeu.



Le script ***GameManager*** a été créé afin de coordonner les différents états du jeu. Nous utilisons notamment la propriété des canevas ***Canevas.enabled*** afin d'afficher ou de cacher un menu.

2.3.2 -Chronomètre / score

Le score et le temps sont deux métriques mises à jour par l'avancée dans le circuit. Tout comme la minimap, ces deux informations sont visibles dans le canevas affiché lors de la partie.



Ils possèdent chacun un script qui s'occupe de l'affichage et de la mise à jour.

Étant donné que le temps doit s'arrêter lors du franchissement de la ligne d'arrivée, c'est le ***TimerScript*** qui déclenche la fin d'une partie avec le menu de félicitation. Pour ce faire, nous testons à chaque instant si le ***Transform*** de la voiture se trouve à une distance inférieure à 4 de la ligne d'arrivée. Ce calcul est possible avec la méthode ***Vector3.Distance()***.

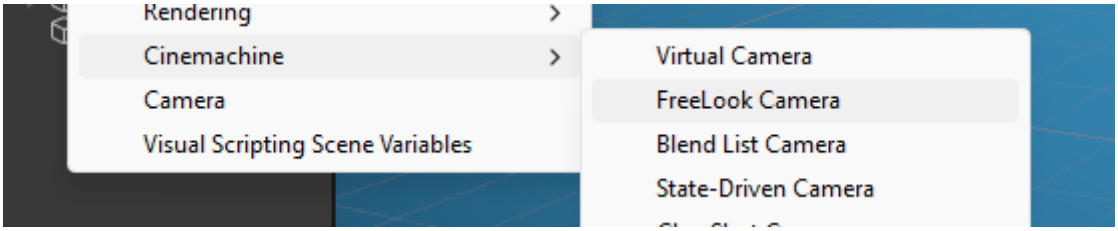
Le score est incrémenté lors de chaque collision avec des entités. Les entités possèdent donc toutes un lien direct sur le script ***ScoreScript*** afin de pouvoir appeler sa méthode ***ScoreScript.increaseScore()***.

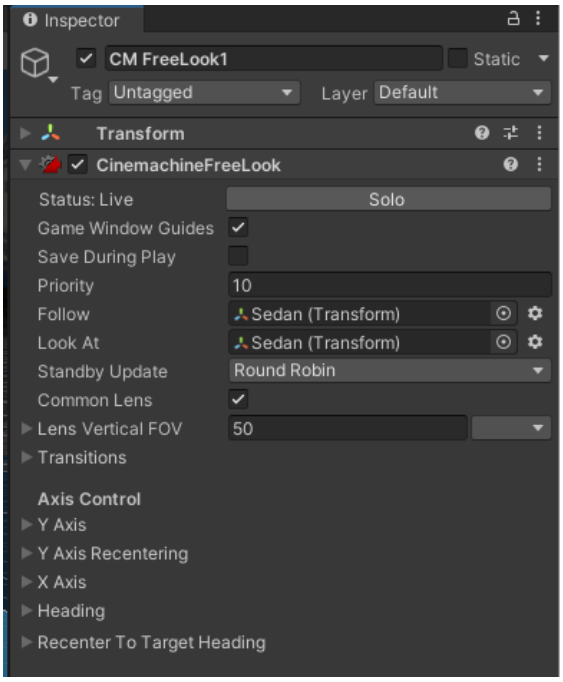
2.3.3 -Caméra avec Cinemachine

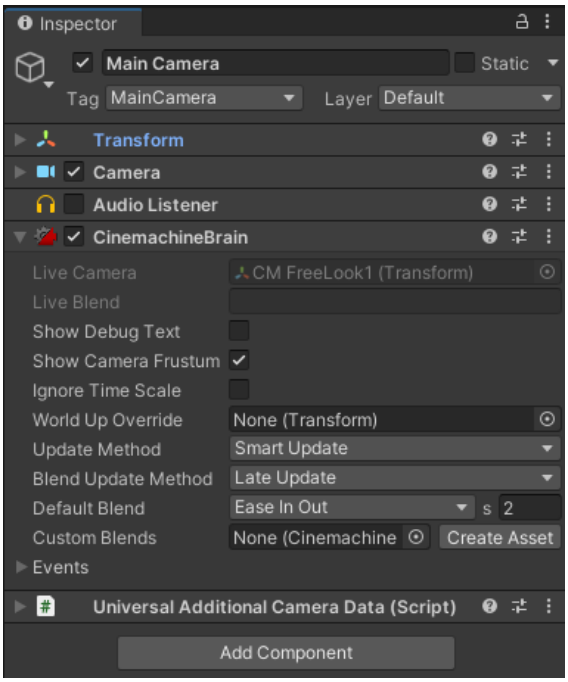
Cinemachine est un package de Unity qui permet de rendre notre caméra plus intelligente et par exemple capable de suivre une voiture à la troisième personne.

C'est tout à fait possible de la faire à la main sans Cinemachine, mais les secousses de la voiture, la voiture qui se retourne et le virage brusque se voient rapidement et le rendu n'est pas bon. Cinemachine gère automatiquement toutes ces problématiques et offre plusieurs types de suivis intéressants.

Pour utiliser Cinemachine, il suffit de créer une ***FreeLook Camera*** et d'y assigner notre voiture dans les propriétés ***Follow*** et ***Look At*** de la ***FreeLook Camera***. Et finalement, lier la ***FreeLook Camera*** à notre caméra principale par le biais d'un Component ***CinemachineBrain*** dans la caméra.







2.4 - Minimap

Dans Unity, la minimap est généralement réalisée à l'aide d'une deuxième caméra qui suit la voiture et affiche une vue du dessus. Voici les étapes à suivre pour mettre en place une minimap dans Unity.

Tout d'abord, nous avons créé une deuxième caméra nommée "Minimap Camera". Cette caméra doit être rendue fille de la voiture pour qu'elle suive automatiquement la voiture lorsqu'elle se déplace. Sa la position et sa rotation ont été adaptées pour obtenir une vue du dessus de la voiture.

Ensuite, pour rendre cette caméra un élément de l'interface utilisateur (UI) Unity, nous avons ajouté une Render Texture nommée "Minimap Render Texture". Nous avons assigné ensuite la Target Texture de la Minimap Camera avec la Minimap Render Texture.

Pour afficher cette texture, il est nécessaire de l'appliquer à un objet Raw Image se trouvant dans un Canvas. Sa taille et sa position peuvent être ajustées en modifiant les propriétés de l'objet Raw Image.

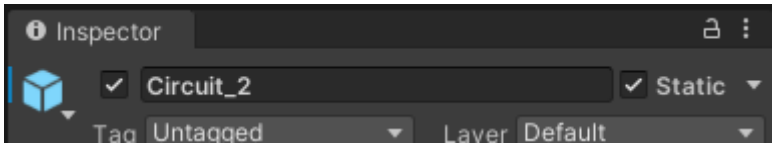
La minimap étant mise en place, nous avons décidé d'afficher un rond vert à la place de la voiture. Nous avons ajouté un layer supplémentaire nommé "Minimap", puis créé une sphère verte, enfant de la voiture. Nous avons modifié légèrement la position de la sphère pour la surélever et avons changé son layer pour le layer Minimap nouvellement créé.

Ensuite, nous devons modifier les paramètres de culling mask de la caméra principale et de la caméra de la minimap. Il est nécessaire de désélectionner le layer Minimap dans le culling mask de la caméra principale, et d'effectuer l'action inverse pour le culling mask de la caméra de la minimap : sélectionner uniquement le layer Minimap.

2.5 - Autres fonctionnalités techniques

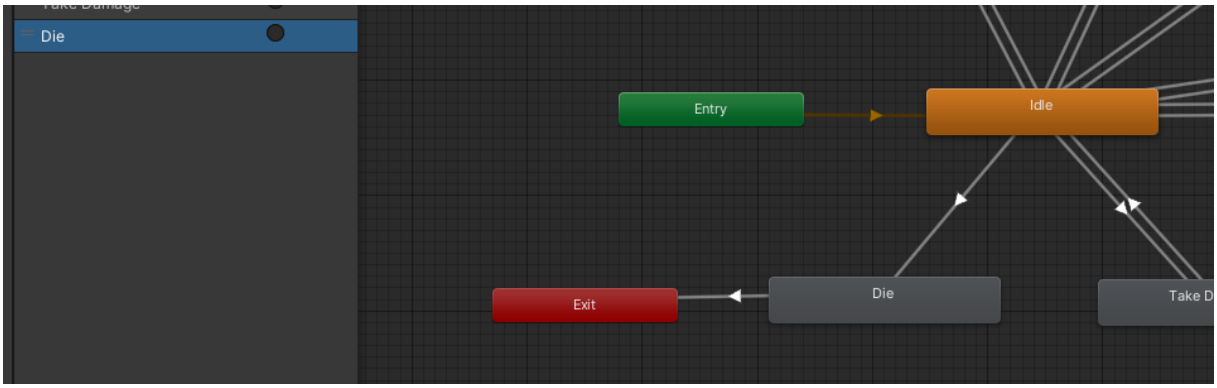
2.5.1 -Ombres et Lightmaps

Étant donné qu'un grand nombre de GameObject sont statiques dans notre scène, il est intéressant d'améliorer le rendu des ombres par un précalcul. Il y a notamment tous les circuits qui peuvent être précalculés en cochant le « static ».



2.5.2 -Animation 3D

Toutes les entités disposées sur les circuits sont animées soit par leurs animations respectives avec des Animation Controller prédéfinis soit animés avec des Animation Controller modifiés pour bouger légèrement en permanence et posséder une animation de mort. L'animation de mort possède un trigger ***Die*** commun à toutes les entités pour faciliter l'appel à cette animation à la suite d'une collision.



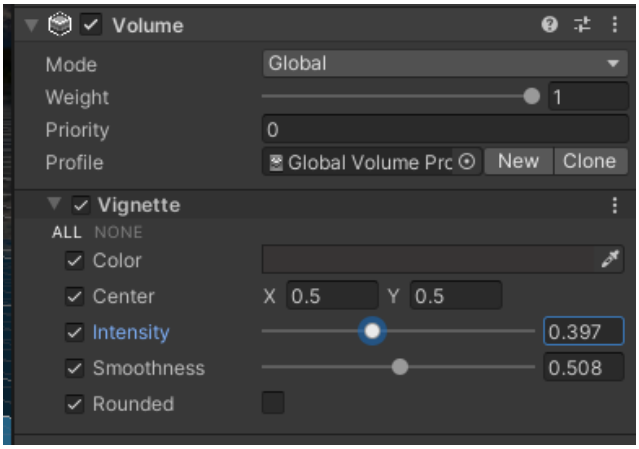
```
IEnumerator Die()
{
    if(animator != null)
    {
        animator.SetTrigger("Die");

        yield return new WaitForSeconds(1f);
    }

    Destroy(gameObject);
}
```

2.5.3 -Post-processing

Afin de pratiquer la mise en place du post-processing et par la même occasion améliorer le rendu de notre jeu, nous avons ajouté un petit effet de vignettage à toute notre scène.



3 - Résultats

3.1 - Vidéo de démonstration

Lien sur la vidéo de démonstration : https://www.youtube.com/watch?v=w5o1gN8bGEQ

3.2 - Jouabilité

La voiture est difficile à maitriser au début et il est difficile de prendre de la vitesse. Cependant, une fois le drift maitrisé, il n'y a pas de problèmes majeurs dans la conduite ou dans la physique de la voiture. Certes, elle se retourne facilement lorsque l'on prend de la vitesse ou que l'on plante trop fort les freins, mais ça fait partie de la difficulté désirée du jeu.

3.3 - Rendu global

L'ombre de la voiture qui est calculée en temps réel n'est pas de très bonne qualité malgré des réglages particuliers pour essayer de l'adoucir.

La skybox combinée avec un léger effet de vignetting donne une impression de clarté et de profondeur de la scène.

Les collisions avec les entités fonctionnent, mais ne donnent pas toujours un joli saut en arrière, mais restent plutôt bloquées sur la voiture.

4 - Limitations et perspectives

Il semble évident qu'un jeu de voiture abouti ne se réalise pas en 2 mois. Cependant, nous nous sommes concentrés sur les aspects principaux d'un jeu de voiture que sont les circuits, la conduite et la logique du jeu.

Une de nos idées de base était de se concentrer sur une simulation réaliste d'écrasement d'entités avec notre voiture. Nous n'avons malheureusement pas eu assez de temps pour approfondir cette facette du projet. À la place, nous avons utilisé les animations 3D par défauts d'entités que nous avons importées et ajouté une force verticale à chaque entité qui entre en collision avec la voiture pour tout de même simuler une violente percussion.

Nous aimerions donc améliorer cet aspect et pourquoi pas modifier les textures pour simuler des saignements et tout ce qui va avec une violente percussion de voiture. Pourquoi pas même aller jusqu'au démembrement des entités plus la collision est violente.

5 - Annexes

5.1 - Où se trouvent quels fichiers

- **Tous nos scripts** : /Script/…
- **Prefab circuits** : /Prefabs/…
- **Prefab routes** : /UnityStore_Prefab/Roads/Prefabs/…
- **Prefab voiture** : /UnityStore_Prefab/Cartoon cars/Prefabs/Flat/Sedan
- **Prefab entités** : /UnityStore_Prefab/*/Prefabs/…
- **Minimap** : /Scenes/MiniMap/MinimapTexture… & MainScene
- **Camera** : MainScene
- **Game logic** : MainScene

5.2 - Bibliographies et références

5.2.1 -Sites Web

Technologies, U. (n.d.). *Unity - Manual: Unity User Manual 2021.3 (LTS)*.

https://docs.unity3d.com/Manual/

The best place for answers about Unity - Unity Answers. (n.d.).

https://answers.unity.com/index.html

[Tutorial] Implementing a Minimap in Unity – The Knights of Unity. (n.d.). The Knights of Unity - Blog of Knowledge. https://blog.theknightsofunity.com/implementing-minimap-unity/

5.2.2 -Assets Unity provenant de l'asset store

Track environment, https://assetstore.unity.com/packages/3d/props/props-for-track-environment-lowpoly-free-211494

Santa claus, https://assetstore.unity.com/packages/3d/characters/meshtint-free-santa-claus-106193

Boximon cyclope, https://assetstore.unity.com/packages/3d/characters/meshtint-free-boximon-cyclopes-mega-toon-series-154436

Boximon fiery, https://assetstore.unity.com/packages/3d/characters/meshtint-free-boximon-fiery-mega-toon-series-153958

Chicken, https://assetstore.unity.com/packages/3d/characters/animals/meshtint-free-chicken-mega-toon-series-151842

Metalon, https://assetstore.unity.com/packages/3d/characters/creatures/meshtint-free-polygonal-metalon-151383

Burrow, https://assetstore.unity.com/packages/3d/characters/creatures/meshtint-free-burrow-cute-series-184837

Roads, https://assetstore.unity.com/packages/3d/environments/roadways/simple-roads-212360

Cars, https://assetstore.unity.com/packages/3d/vehicles/land/4-low-poly-toon-cars-205608