TD/TP 3 : Fonctionnalités / Game Play / Equilibrage Scriptables Objects / Manager / Events /

A. Avant propos : dernières séances

- proposer de nouvelles fonctionnalités et de permettre leur équilibrage simple sans toucher la moindre ligne de code.
- Optimiser en utilisant les LOD
- Centraliser les données et traitements avec les managers
- Recherche d'algorithmes
- Aborder l'enregistrement de données
- Communication entre classes avec les événements
- Qualité logicielle , robustesse

B. Que reprendre du projet initial (TP1 et TP2) ?

Ce que vous voulez ! Par exemple, pour le déplacement du véhicule Joueur, différentes techniques ont été abordées. Retenez celle que vous maîtrisez ou idéalement celle qui semble la plus adaptée à vos idées de GamePlay. (Préférence pour l'approche physique)
Idem pour la gestion des collisions, la création procédurale de la route, etc …

C. Projet à rendre / evaluation / Groupe

Des niveaux d'exigences ou de finition sont proposés pour chaque exercice. Il vont conditionner en partie l'évaluation. ***Must*** 0 à 6, ***Should*** 0 à 9 , ***Could*** 0 à 14 .
Attention : la plupart du temps, le Should est un couche supplémentaire du Must , le Could est un couche supplémentaire du Could. Autrement dit , faire le CC du Could implique d'avoir fait celui du Must et ajouter les fonctionnalités supplémentaire du niveau suivant.

L'évaluation portera en partie sur la **qualité du code** (0 à 5) : lisibilité, séparation IO et traitements, fps et physique, nomenclature, commentaires.
L'évaluation portera en partie via des tests sur la **robustesse et le gameplay** (0 à 3)
Vous réalisez le tp à plusieurs, **groupe de 2 minimum à 5 maximum**

D. Un deuxième mode de jeu : Chrono [algorithmique / sauvegardes]

Le mode classique est celui développé jusque maintenant : parcourir le maximum de distance sans limite de temps.
Ce nouveau mode Chrono proposera de parcourir le maximum de distances en un temps limité (e.g. 10 sec)
La course du meilleur score actuel sera représentée sous forme d'un fantôme (véhicule transparent). Les positions (à intervalle de temps réguliers, période du mateur physique, méthode FixedUpdate) du meilleur score seront donc enregistrées . Elles seront exploitées dans les courses chronos à venir pour visualiser les positions du meilleur chrono.

Must : Enregistrer et mettre à jour les 3 meilleurs temps (PlayerPrefs) **ET** afficher le fantôme

Shoud : En plus des scores enregistrés, enregistrer la course(position, orientation, intervalles) du meilleur chrono pour pouvoir afficher son fantôme d'une exécution à l'autre (fichier json ou fichiers binaires)

Could : pour plus de fluidité et exploiter au mieux les FPS, enregistrer la position et temps à chaque appel à Update (au lieu de FixedUpdate). Lors de la simulation du véhicule fantôme topChrono, il faudra interpoler les positions/orientations entre deux positions car les fps de l'enregistrement du meilleur chrono seront différents des fps du nouveau chrono en cours.

E. Obstacles et actions [Collisions/ Manager / Events]

Au moins 1 obstacle doit entrainer un bonus (à définir) pour le joueur.
Au moins 1 obstacle va le pénaliser ou rendre plus difficile sa progression (à définir).

Ces actions auront un effet limité dans le temps.

Must : Utiliser les détections de collision (OnTriggerXXXX , OnCollisionXXX) et un manager GameManager pour gérer les propriétés/caractéristique du véhicule du joueur (e.g. vitesse max, VitesseMin, bonus / malus en cours)

Should :
SOIT Utiliser les events pour gérer les effets des collisions avec ces bonus/malus
SOIT intégrer la fréquence d'apparition de ces bonus/malus dans les paramètres d'équilibrage du jeu (fréquence/ probabilité, en fonction du temps ou de la progression)

Could : Utiliser un manager Sons et l'exploiter dans le jeu pour ces bonus malus, et aussi pour d'autres aspects (autres collisions, freinage, drifts , fin et début de jeu)

F. Plusieurs Niveaux de difficulté : [équilibrage / scriptableObject]

Chaque *niveau de difficulté* définit :

- Un niveau de difficulté *initial* (voir paramètres ci-dessous)
- Une *évolution* spécifique de la difficulté ce niveau pendant le temps de jeu

L'évolution du niveau de difficulté pendant le jeu peut être définie en fonction du temps, de la distance parcourue, des bonus/malus, etc. à votre choix et appréciation.

Les *paramètres de difficultés* peuvent être la vitesse, la largeur de la route, le nombre d'obstacles le type d'obstacles, le type de pattern de route (sécurisés ou non), l'influence des malus/bonus, etc …
Ces paramètres sont donc initialisés en fonction du choix de difficulté initial puis , idéalement (could), sont modifiés en fonction de l'évolution du joueur dans le niveau

La complexité de cette tâche n'est pas algorithmique ni technique mais nécessite de :
- Avoir un code permettant d'initialiser et modifier *simplement* ces paramètres de difficultés.
- Trouver pour chaque niveau un ensemble *équilibré* de valeurs de ces paramètres permettant de proposer un jeu intéressant.

L'idéal sera de pouvoir équilibrer le jeu sans avoir à modifier le code ni même à le consulter. Tout devrait se faire depuis l'IDE d'unity via les paramètres publics d'entrée des scripts et managers et scriptableObjects (Should) .

Must : Proposer 2 niveaux de difficultés et garder l'application facilement équilibrable. Proposer une UI pour le choix de la difficulté

Should : Utiliser les scriptables Objets pour définir facilement plusieurs (au moins 3) niveaux de difficultés

Could : Utiliser les scriptables objets pour définir plusieurs évolution possible de la difficulté pendant la progression du joueur, difficulté progressive en fonction d'un temps de jeu ou d'une distance parcourue

G. Proposer un obstacle (ou décor) LOD [optimisation]

Le principe du LOD (Level of details) permet d'utiliser une résolution de maillage d'un objet en fonction de sa taille sur l'écran. En vulgarisant, plus il est loin de la caméra, plus sa surface sur l'image de rendu sera petite, moins on a besoin de détails.
Pour cela, il faut donc plusieurs maillages de résolution différentes pour un même objet et un component LOD Group qui permet d'ajuster les choix de maillage et transitions.
Plusieurs maillages ? le faire avec votre modeler habituel ou récupérer un objet disponible avec plusieurs résolutions.
A priori pas de code à fournir mais un component à paramétrer correctement.

Remarque : il existe aussi cette optimisation pour les animations : plus l'objet ou le personnage à animer est petit (ou loin) , moins il est utile de l'animer.

Imposé : préfixer le nom de votre objet de LOD_ (e.g. LOD_MonObjet) et suffixer chaque maillage de _LODx (LOD_MonObjet_LOD0, LOD_MonObjet_LOD1, LOD_MonObjet_LOD2)

Must : ajouter 2 obstacles LOD

Should : Utiliser ou non les LOD en fonction d'un paramètre de l'application (Priorité Rendu/ Priorité FPS)

Could : Pour les obstacles avec une comportement (e.g. n déplacement) , n'activer l'animation en fonction de l'importance de l'objet à l'écran.

DOC : Component LOD Group ImportLOD in Unity
Tutorial : https://learn.unity.com/tutorial/working-with-lods-2019-3
Youtube : How to set Automatic LOD (Blender & unity)

H. Changement de caméra [UI / UX]

Proposer deux caméras, l 'une proche d'un mode 3ème personne, l'autre plutôt 1ère personne (devant le véhicule ou à l'intérieur)

Must : Utiliser la touche C pour passer d'une caméra à l'autre (ou d'une position de caméra à l'autre)

Should : Ne gérer qu'une seule caméra ; proposer une transition progressive de la position (et paramètre) de 3ème personne à celle de 1ère personne , et inversement. La transition ne sera pas brutale (pas un changement de position) mais progressive et travaillée (utiliser curve animation ou algo équivalent pour déplacer la caméra sur un temps donné)

Could : en mode 3ème Personne, faire tourner les roues en fonction de la vitesse, tourner les roues avant lorsque le véhicule tourne, ajouter un effet sur les roues lorsque le véhicule freine.

DOC : https://docs.unity3d.com/Manual/MultipleCameras.html
Tutorial : https://learn.unity.com/tutorial/the-camera#