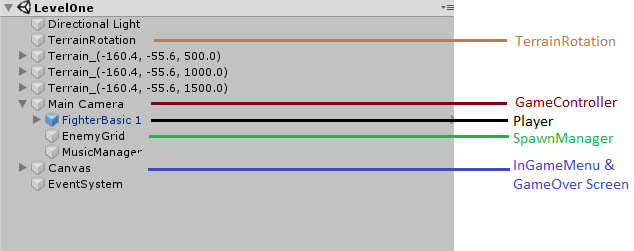
# Doc technique :

Voici d’abord l’inspector Unity avec les GameObject les plus importants :



La plupart des GameObject affichés ont en composant des scripts cruciaux dont voici un résumé :

**TerrainRotation** : Répète le terrain pour que le niveau paraisse infini

**GameController** : S’occupe de contrôler le vaisseau du joueur et d’avancer la caméra

**Player** : Le vaisseau du joueur

**SpawnManager** : Sur le GameObject EnemyGrid, sert a faire apparaitre les ennemis régulièrement.

**InGameMenu & GameOverScreen** : Contient le menu de pause & l’écran de Game over.

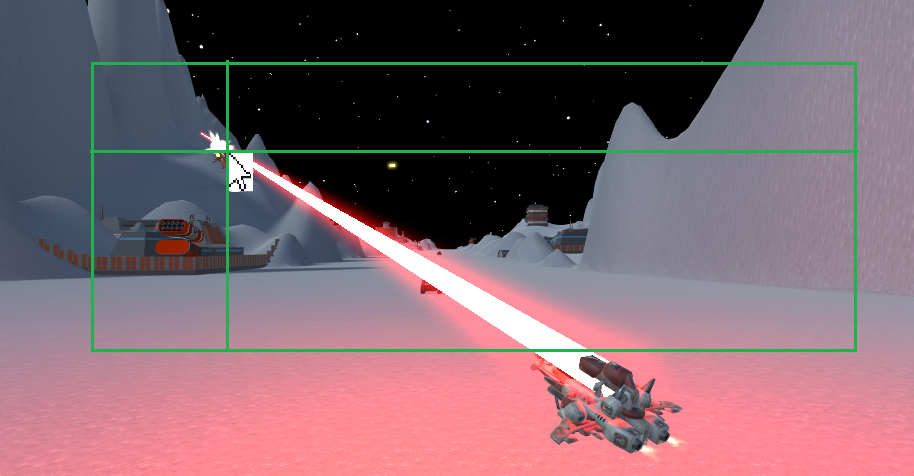
Les classes attachées à ces scripts seront détaillées dans la partie suivante.

Comme vu au-dessus, le vaisseau du joueur ainsi que le GameObject « EnemyGrid » sont enfants de la MainCamera, donc quand cette dernière avance, les autres suivent également.

Les ennemis qui apparaissent restent collés sur l’axe Z de l’EnemyGrid, et ne se déplace que sur les axes X et Y.

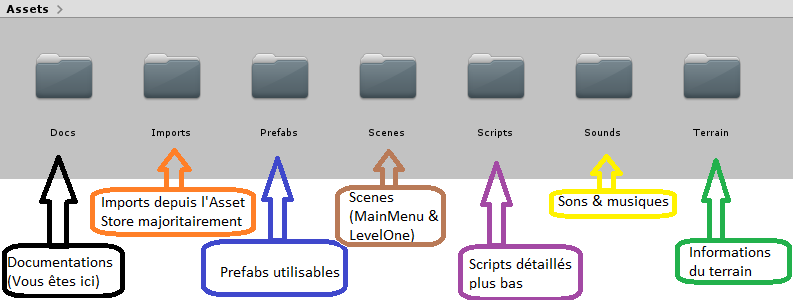
Cela permet, quand on cible à la souris, de n’envoyer des projectiles qu’en direction de cette grille via la méthode : shipGO.transform.LookAt(mainCamera.ScreenToWorldPoint(mousePos));

Ci-dessous un screen avec représenté en vert la fameuse EnemyGrid, et la position de la souris projetée sur cette grille :



Aussi, afin de pouvoir afficher les effets d’impacts à l’endroit de l’impact, j’ai du utiliser les méthodes OnCollisionEnter (et OnCollisionStay). Je n’ai pas réussi à obtenir cet effet avec les OnTriggerEnter. J’ai donc du utiliser les collisions classiques pour se faire. J’ai été confronté a un problème : Les vaisseaux ayant la checkbox ‘IsKinetic’ à true car je souhaitais contrôler les mouvements moi-même et ne pas avoir de collision ni gravité, tout comme les projectiles, la méthode OnCollisionEnter n’était pas appelée. En effet, les collisions sont par défaut désactivées dans ces conditions. J’ai donc du modifier les paramètres de physique de Unity en sélectionnant ProjectSettings => Physics => Contact Pairs Mode => Enable Kinematic – Kinematic pairs.

Enfin, voici une petite explication concernant les répertoires dans /Assets/ :



# Explication des classes

## Classes pour le contrôle du jeu (Dossier Units & Controllers) :

On y retrouve ici quelques classes, mais ce sont celles qui s’occupe de contrôler la majorité du jeu :

* « GameController » : Cette classe à attacher à la Caméra sert notamment a faire avancer la Caméra a une vitesse fixe (Et donc tous les GameObjects enfants, comme le vaisseau du joueur ou la grille d’ennemis), mais contient aussi la référence du GameObject du joueur afin de contrôler ses déplacements et sa rotation. On y retrouve donc quelques spécificités liées joueur : Sa vitesse de déplacement et les variables nécessaires la fonctionnalité de réparation. Cette classe sert également pour afficher (Via OnGui) l’interface en bas d’écran : les barres de Vie et de Ressource (fuel).
* « SpawnManager » : Classe s’occupant du spawn des ennemis sur la grille. Elle contient de nombreuses informations comme les listes de GameObject a instancier (Les escadrons, les véhicules, les boss, ..) ainsi que le timer de chacun. Elle s’occupe aussi de faire apparaitre les vaisseaux à une distance paramétrable avec une réduction de vitesse jusqu’à ce que ceux-ci soient sur la grille, puis les active.
* « TerrainRotation » : Permet que le niveau soit ‘infini’. Elle possède comme variable publique une liste de GameObject pour contenir les différentes parcelles de terrain, et s’occupe de les déplacer à « l’avant » une fois que ces derniers ne sont plus visibles par la Caméra.

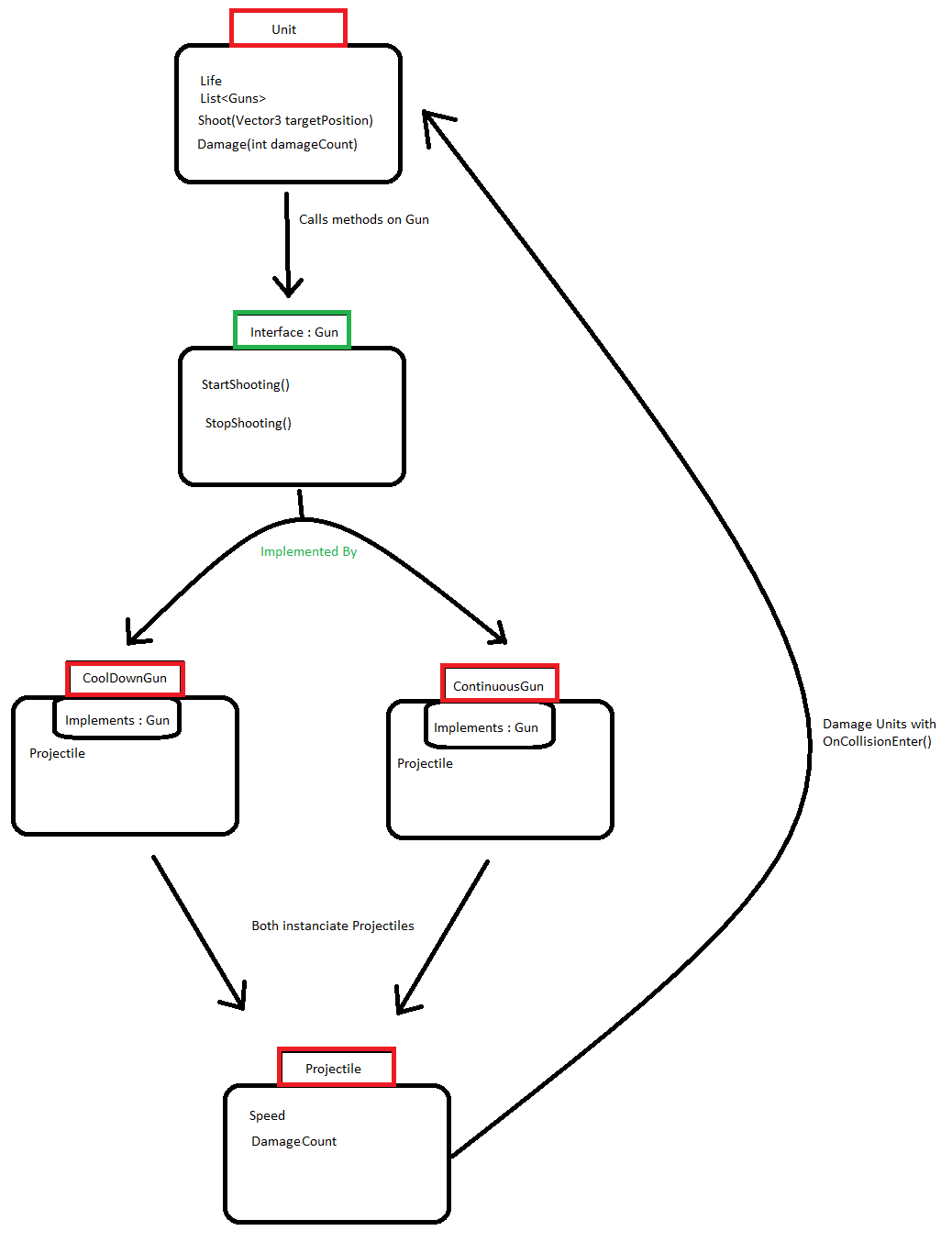
## Classes Utilitaires (Dossier Utils et Sounds)

* La classe la plus importante est sans aucun doute « GameUtils » : Cette classe statique est appellée par différents composants, et permet d’être chargée en mémoire une seule fois, peu importe la scene chargée. Elle retient les informations du joueur comme son nom et son score, et des méthodes récurrentes comme *IsPlayerAlive()* ou *GetPlayerGameObject()*. Elle contient également les méthodes pour charger ou recharger une scene, et aussi les méthodes qui permettent de sauvegarder / charger sur le disque les meilleurs scores (Ou d’en créer des ‘fake’ si aucun meilleur score n’existe pour commencer).
* « Score » : C’est une classe simple qui permet de stocker un score : un nom et un nombre de point. Nécéssaire notamment pour Serializer /Déserializer via [System.Serializable].
* « HighScores » : Contient une *Map<Int,Score>* (ou *Dictionary<Int,Score>*) et s’occupe de garder les 10 meilleurs scores en mémoire. C’est via cette classe qu’on ajoute un score (*AddScore(Score)),* et elle s’occupe de le placer à la bonne position par rapport aux autres.
* « Utils » : Cette classe statique contient quelques méthodes utilitaires à part, comme *GetMainCamera(), GetRandomClip(List<AudioClip>),* ou encore *PlayAllParticleSystemFrom(…)* qui permet récursivement de lancer la méthode *ParticleSystem.Play()*  de tous les GameObject enfants celui passé en paramètre (et lui y compris) avec quelques subtilités : par exemple le scaling a appliquer sur les particules, si on doit le détacher de l’objet parent ou non (Cela permet par exemple aux explosions des vaisseaux de ne pas avancer avec la camera, contrairement à leurs effets d’impact).
* « LimitToBounds » : Script permettant de limiter les mouvements des vaisseaux a la position relative souhaitée (*transform.localPosition*) pour que ceux-ci ne dépassent pas les bornes de la caméra par exemple ou ne s’écrase pas toujours au sol (Cela arrive de temps en temps, c’est fait exprès ! 😊).
* « MusicManager » : Script a attacher a l’objet qui aura le composant *AudioSource* et permet de jouer une musique au hasard quand il n’y a pas de musique en cours parmi la liste paramètrable. On peut aussi modifier la liste en jeu, par exemple pour changer la playlist pour l’écran de GameOver.
* « SoundManager » : Classe statique permettant de jouer ou de stopper un son. Via *PlaySound(…)*, elle ajoute a un GameObject un composant AudioSource, et fait jouer le son désiré au volume choisi. Elle peut faire de même avec une liste de son, et choisira au hasard le son a jouer en utilisant la même méthode combinée a *Utils.GetRandomClip(List<AudioClip>)* vue plus haut.
* « Terrain » : Script attaché au terrain avec comme unique méthode *CreateImpact(…)* qui s’occupe d’instancier la particule désirée au point d’impact avec un projectile.

PS : Sons & musiques éditées avec Audacity

## Classes principales pour le combat (Dossier Units & Controllers et Gun) :

* Unit : Représente une unité, au sol ou aérienne. Possède des points de vie, le son a jouer en cas de mort, la liste de « Gun », la vitesse de déplacement, les effets de particules.
* Gun : Interface dont les implémentations sont utilisées par les armes des vaisseaux. CooldownGun est une arme tirant un « Projectile » avec un temps de recharge entre chaque tir et une dispersion / imprécision paramétrable, et ContinuousGun est une arme tirant un laser direct instantané (uniquement utilisé par le joueur)
* Projectile : Classe représentant le projectile, un laser ou une roquette notamment, contient les informations de dégâts et de vitesse. C’est cette classe qui gère les collision avec les « Unit » et leur inflige les dégâts.



## Classes pour les Menus (Dossier Menu & UI) :

Beaucoup de classes sont à usage unique, notamment pour les boutons. Je ne présenterai pas celles-ci étant donné qu’elles ne contiennent qu’une méthode hormis « HowToPlayButton » : Cette classe est utilisée deux fois, elle prends deux GameObject en variable publique : lors de l’appel a l’unique méthode de la classe, le premier est désactivé et le second activé (GameObject.SetActive(bool)).

* « ScrollingHighScores » : Permet de controller la vitesse de défilement des meilleurs scores dans le menu principal, et la position a partir de laquelle ils doivent revenir à leur position initiale..
* « DeathScreenManager » : Permet d’afficher l’écran de GameOver. Il a en variable publique tous les GameObject contenant les différents textes a afficher comme les scores, le nombre d’ennemi tué,mais aussi le GameObject parent a activer pour que le menu de GameOver apparaisse, la liste de musique à jouer et le temps de ‘fade-in’. Les scores sont récupérés par la classe GameUtils.