

Compte-rendu Space Survival

Compte-rendu Space Survival



Réalisé par :

- Brinis Halim

Table des matières

I.	Introduction.....	3
II.	Outils	3
i.	Unity.	
ii.	Visual Studio.	
iii.	Sqlite Manager.	
III.	Présentation des interfaces et du jeu.....	
IV.	Réalisation.....	4
i.	Préparation (Quelques notions)	
ii.	L'environnement.	
iii.	Le joueur.	
iv.	Ennemis.	
v.	Audio.	
vi.	Scores.	
vii.	Menu principale.	
viii.	Quelques Scripts en plus.	
ix.	Construction du jeu.	
IV.	Conclusion	7

I. Introduction :

« Space Survival » est un jeu vidéo 2.5D que réalisé grâce Unity qui est un environnement de développement multiplateforme dédié à la réalisation de jeu vidéo, j'ai utilisé du C# afin de mener cette tâche a bien, durant la réalisation du jeu j'ai dû télécharger des « Assets » qui sont eux des éléments gratuits fournis sur le site officiel de Unity, ces « Assets » contiennent la totalité des éléments graphiques et sonores qu'on peut voir dans le jeu, tel les modèles 3D, les textures, les son, les animations ainsi que les effet spéciaux, tout ce pack est fournis gratuitement sur Unity et est à libre disposition pour tous les utilisateur, le jeu vidéo en soit est un « Shoot 'Em Up » autrement dit un jeu de tire arcade a scrolling vertical (vu du dessus) ou le joueur pilote un vaisseau et fait face à des vagues d'ennemies, le jeu en soit comporte qu'un seul niveau et le but y est de survivre le plus longtemps, à la fin le joueur peu sauvegarder son score dans une base de donnée ou j'ai dû utiliser du SQLite afin de stocker les scores.



Figure 1 – Icône du Jeu Space Shooter

II. Outils :

Comme mentionné plus haut j'ai dû utiliser Unity, et Visual Studio pour la réalisation de ce jeu :

- A. Unity** est un moteur de jeu multiplateforme (smartphone, Mac, PC, consoles de jeux vidéo et web) développé par Unity Technologies. Il est l'un des plus répandus dans l'industrie du jeu vidéo, du fait de sa rapidité aux prototypages pour les très gros studios, aussi pour la sphère du jeu indépendant qui développe directement dessus pour sortir leurs applications sur tout support
- Le moteur est capable de supporter deux différents langages le C# ainsi que le Java Script.



Figure 2 : Logo officiel d'Unity.

B. Visual Studio : La célèbre suite de logiciels Microsoft pour Windows et Mac, j'ai donc utilisé cet environnement pour toute la partie code du jeu « les scripts », l'environnement étant associé directement avec Unity lors de l'installation de ce dernier j'ai donc pas eu besoin de faire une quelconque manipulation vis-à-vis de ce dernier.



Figure 3 : Logo officiel de Visual Studio.

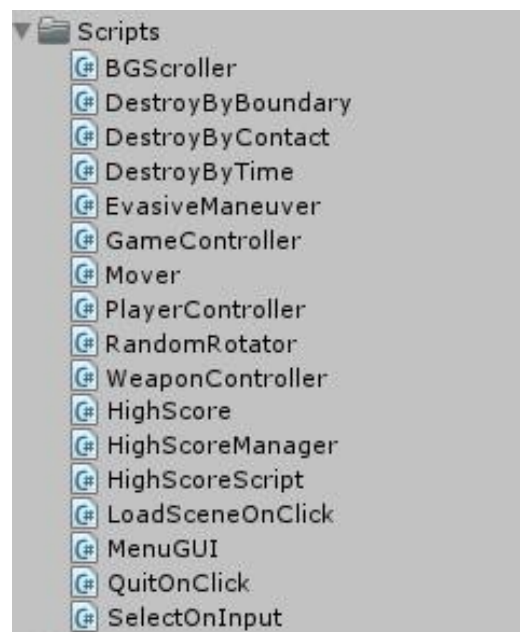


Figure 4 : Liste des scripts écrits sous Visual Studio.

C. Sqlite Manager : J'ai dû utiliser aussi une base de données en SQLite3 afin de stocker les scores, Sqlite Manager est quant à lui un « Addon » (un plugin) pour Firefox qui apporte la possibilité de gérer les fichiers Sqlite sous Firefox, connecter SQLite3 à Unity n'a pas été une mince affaire pour ce dernier j'ai dû utiliser quelques fichiers dll : sqlite3.dll (ou sqlite3.dll et sqlite3.def pour les builds 32 bits du jeu) ainsi que Mono.Data.Sqlite.dll et System.Data.dll qui sont eux deux des dll fournies par Unity, la base de données quant à elle dispose d'une table HighScores qui elle aussi dispose de 5 champs : une ID en entier qui sert de clé primaire, un champ Nom en texte, un champ score en entier, un champ wave en entier pour voir à quel vague le joueur est arrivé et finalement un champ Date afin de mieux positionner les scores si ils sont pareils et que l'un soit plus ancien que l'autre.

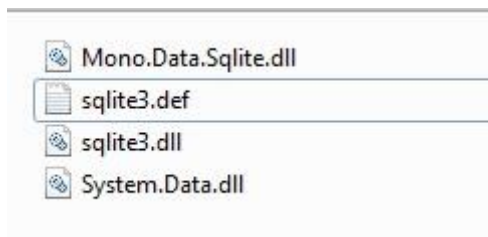


Figure 5 : Liste des fichiers dll utilisées.

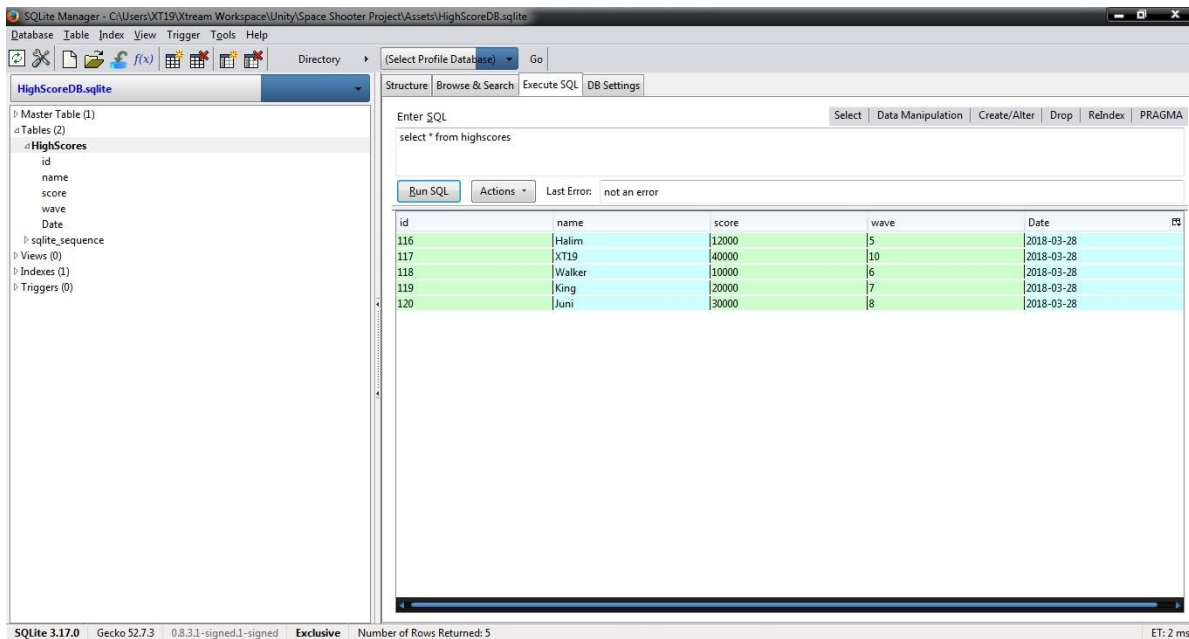


Figure 6 : Sqlite manager, la base de données.

III. Présentation des interfaces et du jeu :

Lors du lancement de Space Survival on a donc droit a cette interface :

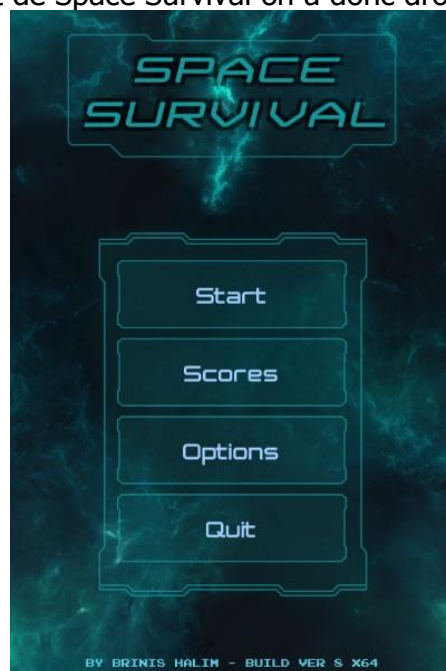


Figure 7 : Menu Principale de Space Survival.

De bas en haut on a donc un bouton « Quit » qui nous permet comme son nom l'indique de fermer le jeu, un bouton Option qui nous permet d'être redirigé avec la configuration des options ou on aura droit à la gestion de volume du jeu ainsi qu'à la remise à zéro de la base de donnée, le bouton « Scores » quant à lui va nous permettre de voir le tableau des scores, ainsi que « Play » qui permet de lancer une partie.



Figure 8 : Menu Options.

Je tiens à dire que Les fonctions du menu options jusque-là par contre ne sont que visuels, car à ce moment-là j'avais pas encore implémenté le contrôle du volume, on a donc droit à deux bouton fonctionnelles, « Reset Game » qui permet de remettre la base de donnée à zéro, et d'effacer le tableau des scores, par contre une autre fenêtre de confirmation va apparaitre lorsque on clique sur « Reset Game », Back quant à lui permet simplement de revenir sur le menu principale.



Figure 8 : Fenêtre de confirmation du Reset.

Ici on a droit à deux boutons « Cancel » et « Confirm » qui comme leur noms l'indique le premier permet de revenir sur le menu options, et l'autre de confirmer le vidage de la base de donnée.

De retour sur le menu principal, n'oublions pas le bouton Scores qui va nous afficher le menu suivant :



RANK	NAME	SCORE	WAVE
#1	XT19	40000	10
#2	JUNI	30000	8
#3	KING	20000	7
#4	HALIM	12000	5
#5	WALKER	10000	6

Back

Figure 9 : Tableau des Scores.

Le tableau des scores est un menu à défiler soit grâce à la barre qui est à droite soit directement en cliquant et en glissant sur le menu lui-même, en bas ce tableau on a aussi un bouton « Back » tout comme options qui nous permet de revenir sur le menu principale du jeu.

Arrive finalement le cœur du programme le jeu en soit, ce dernier se lance en cliquant sur le bouton, Play, le joueur contrôle le vaisseau grâce aux flèches directionnelles et peut aussi tirer grâce à la touche Ctrl, le but du jeu est de survivre le plus longtemps possible afin de faire le plus gros score possible et ce en survivant a des vagues d'ennemies de plus en plus nombreuses et de plus en plus rapide.



Figure 10 : Space Survival

IV. Réalisation :

La réalisation de Space Survival fut longue et complexe, je vais ici essayer de détailler de mon mieux la réalisation du jeu depuis ces débuts

A. Préparation (Quelques notions): L'une des premières difficultés à la quels j'étais heurté était la prise en main de Unity, ce dernier proposant des concepts propres à lui, et n'ayant pas spécialement beaucoup travaillé avec des environnements dédiés à la réalisation de jeu vidéo, il m'aura fallu apprendre de certaines notions apportées par Unity.

En bas on peut voir un aperçu de la fenêtre de Unity lorsque je travaille sur le projet, cette fenêtre peut être divisée en 5 Composantes principales (La Scène, la Hiérarchie, L'inspecteur, le projet, et le Jeu) la scène ou on peut y importer des objets tel le vaisseau qu'on voit sur la photo, sur cette scène on peut simuler en temps réel ce que donnera la propriété de chaque qu'on ajoute à la scène et ce en lançant le jeu grâce au bouton Play en haut, tous les objets présents sur la scène le sont aussi dans la Hiérarchie ou on peut plus facilement voir la liste des objets que la scène actuelle comporte, on peut y former des groupes d'objets aussi, et de mettre des objets en tant que parent d'autre ce qui va leur permettre d'avoir des propriétés communes, en bas de la Hiérarchie on peut voir le Projet qui comporte tous les fichiers utilisés afin de réaliser le jeu, que ce soit des fichiers audio, des images, des textures, des modèles ou même les scripts qu'on écrits tous seront à l'intérieur du dossier « Assets » du jeu, à droite on a l'inspecteur, qui nous montre les propriétés de l'objet qu'on a sélectionné actuellement dans la scène ou dans la Hiérarchie, ici par exemple on peut voir l'objet Joueur étant sélectionné et se dernier disposant de certaines « Composantes » qui sont mentionnées à droite dans L'inspecteur, on ici par exemple la composante Transform qui définit la position, la rotation et la taille actuelle de l'objet dans la scène, Le Rigidbody qui permet d'utiliser la physique par défaut de Unity, qui par exemple dans ce cas on a le case « Use Gravity » décoché, car on veut pas que le vaisseau soit influencé par la gravité et tombe dans le vide, tout en bas aussi on peut voir Done_Player_Controller qui est aussi une composante attachée à l'objet du joueur, par contre cette dernière est un Script que j'ai écrit sous Visual Studio et qui va permettre au joueur de se mouvoir grâce aux touches du clavier, et de tirer.

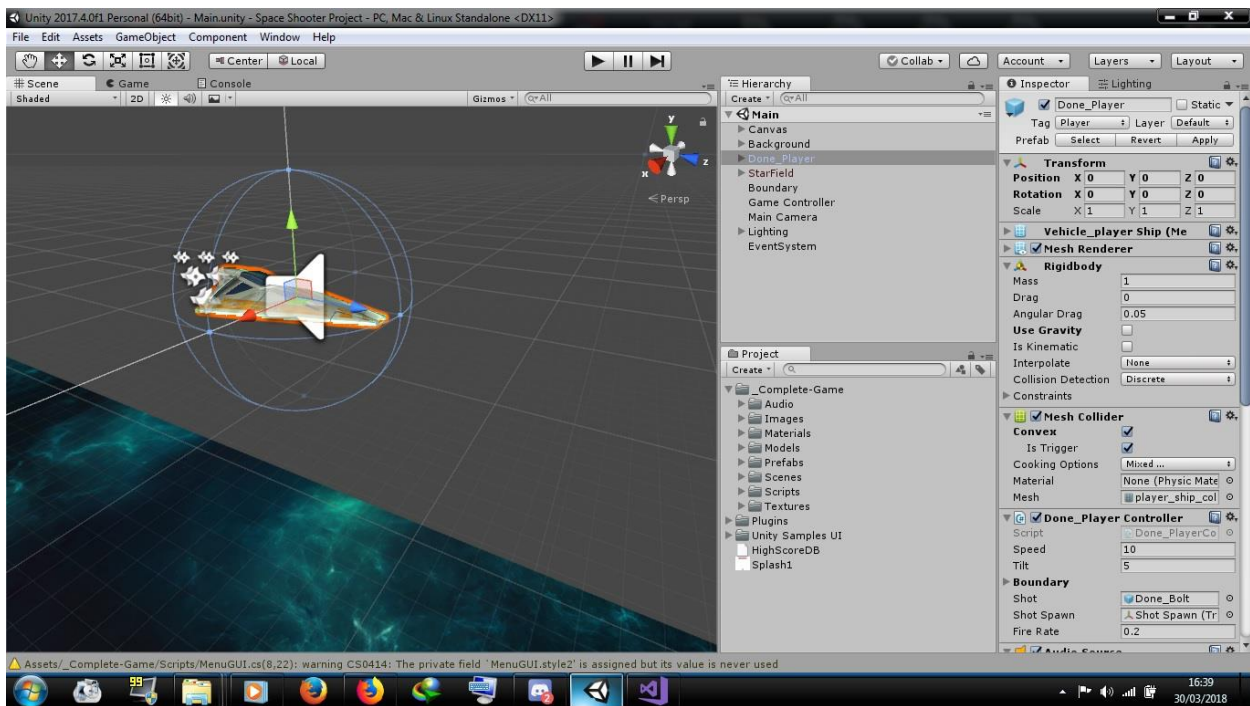


Figure 11 : Fenêtre Unity.

Durant ce projet j'ai utilisé des textures et des modèles ainsi que des musiques qui m'appartiennent pas, Unity étant une plateforme très riche elle offre aussi une grande variété de choix par rapport au Assets qu'on utilise, Unity dispose donc d'un « Store » où on peut télécharger certains modèles gratuit.

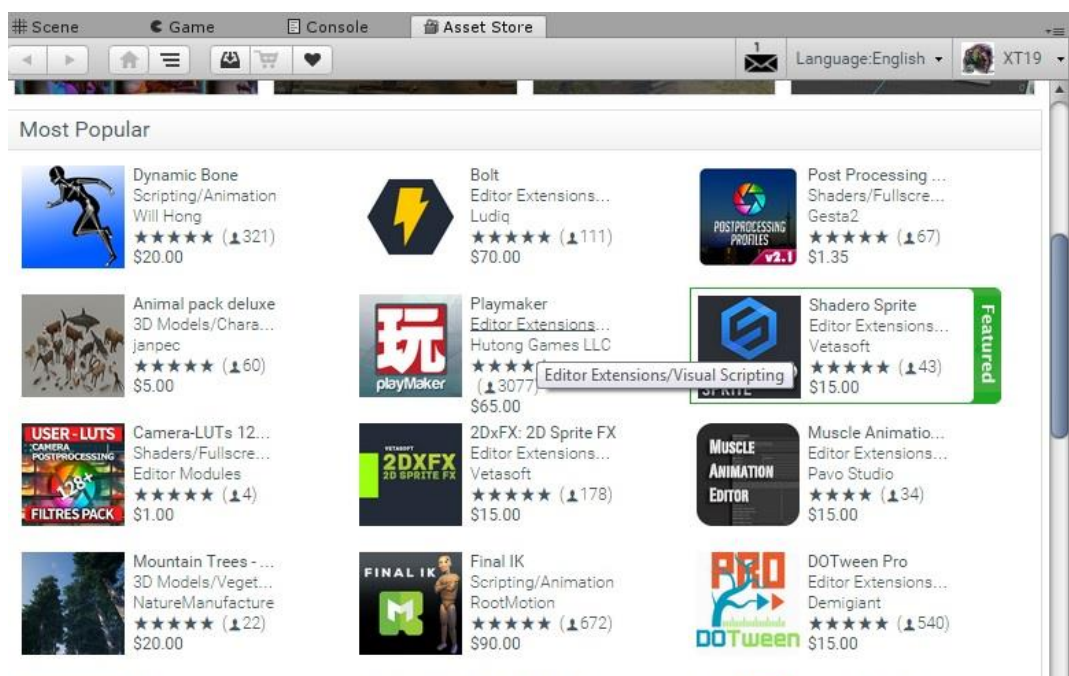


Figure 12 : Asset Store Unity.

B. L'environnement :

La première étape avec Unity était de poser les bases même de l'environnement du jeu, c'est-à-dire l'arrière plant, les lumières, ainsi que la camera du jeu, sur l'image on peut voir la toute première scène du jeu avec seulement la camera de base et sans aucune lumière dans la scène, j'ai changé la résolution de la camera pour avoir un rendu coupé sur les coté (comme beaucoup de jeux de tire arcade) en bas on peut voir le fichier qui comporte la scène actuelle

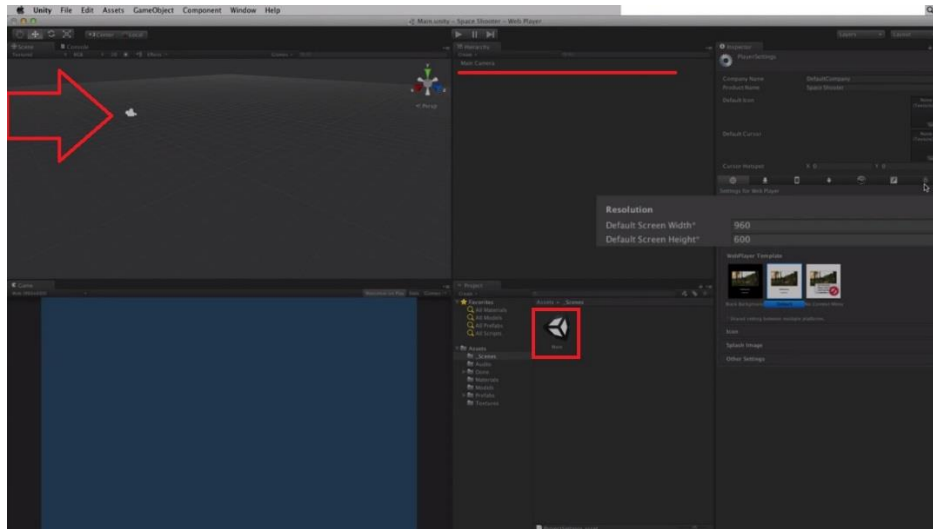


Figure 13 : Camera Unity

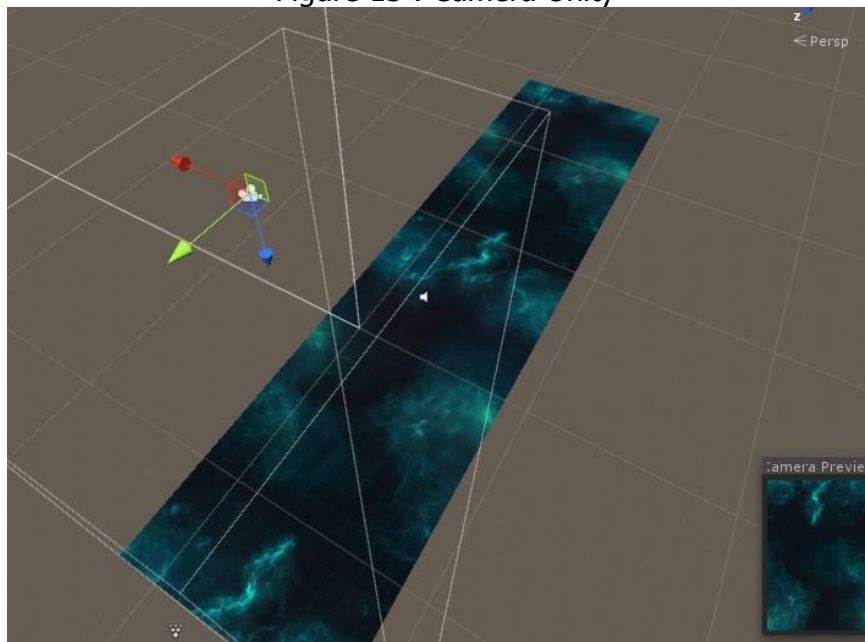


Figure 13 : Camera Orthographique et arrière plant.

Comme on le voit dans la Figure 13 j'ai utilisé ce qu'on appelle une Caméra Orthographique, à la différence de la camera de perspective, celle-ci ne voit pas la distance et de ce fait quel que soit la distance de l'arrière plant de cette dernière elle n'y verra aucune différence, (en bas à droite on peut voir le rendu de la camera).

En dernier lieu il m'aura fallu ajouter une frontière, un objet dont j'ai enlevé la visibilité et juste laissé la boîte de collision afin qu'il prenne en compte tout ce qui entre en contact avec lui et le détruit comme les tirs, et les ennemis qui sortent de l'écran afin que la scène et le

jeu ne se retrouve va surcharger en objets inutilement instancié, c'est ce que Destroy_by_boundary fait.

```
using UnityEngine;
using System.Collections;

public class Done_DestroyByBoundary : MonoBehaviour
{
    void OnTriggerExit (Collider other)
    {
        Destroy(other.gameObject);
    }
}
```

Figure 14 : Code DestroyByBoundary.

C. Le joueur :

Pour le joueur donc comme on l'a vu plus haut on pose un vaisseau pour le représenter la texture ainsi que le model de ce vaisseau étant été importé depuis l'asset store, le model étant directement disponible dans le projet une fois importé on a donc droit à un dossier Models ou sont stocké tous les modèles du jeu, les modèles dépendent aussi d'un autre dossier ou sont stocké les textures de chaque objet car y'a d'un côté le model (le squelette) ainsi que la texture (sa peau).

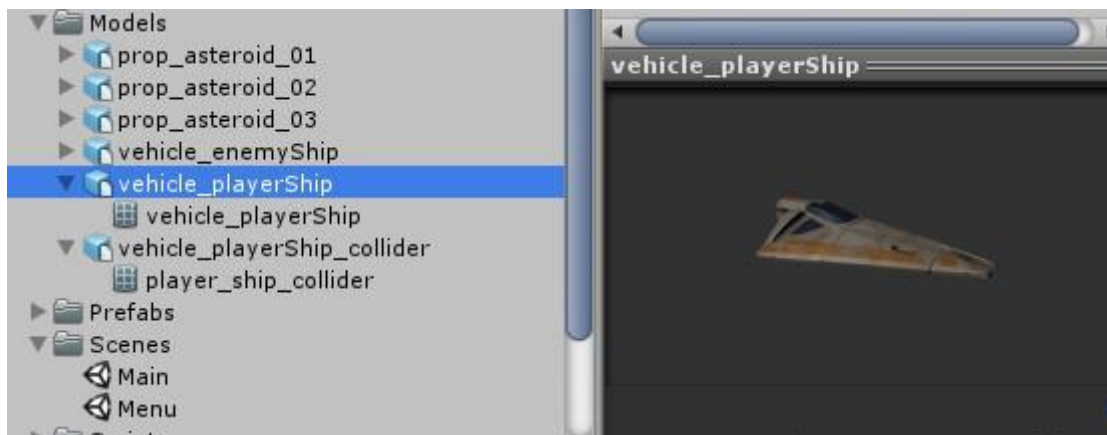


Figure 15 : Model du joueur.

On peut voir ici dans le dossier Models tous les modèles que j'ai pu utiliser que ce soit le vaisseau du joueur, les ennemies, ou les astéroïdes (je rappelle que ces derniers sont directement ostensibles via l'asset store d'Unity ainsi que beaucoup d'autres ressources.

En glissant donc cet objet on pourra interagir avec lui en changeant ces propriétés et en lui attachant de nouvelles composantes.

Je prends à titre d'exemple l'objet du joueur et j'essaierai de le détailler au maximum afin de souligner toute les subtilités dont il est question, l'objet du jouer donc une fois intensité sur la scène et une fois qu'on y a cliqué dessus on a droit aux propriétés suivantes :

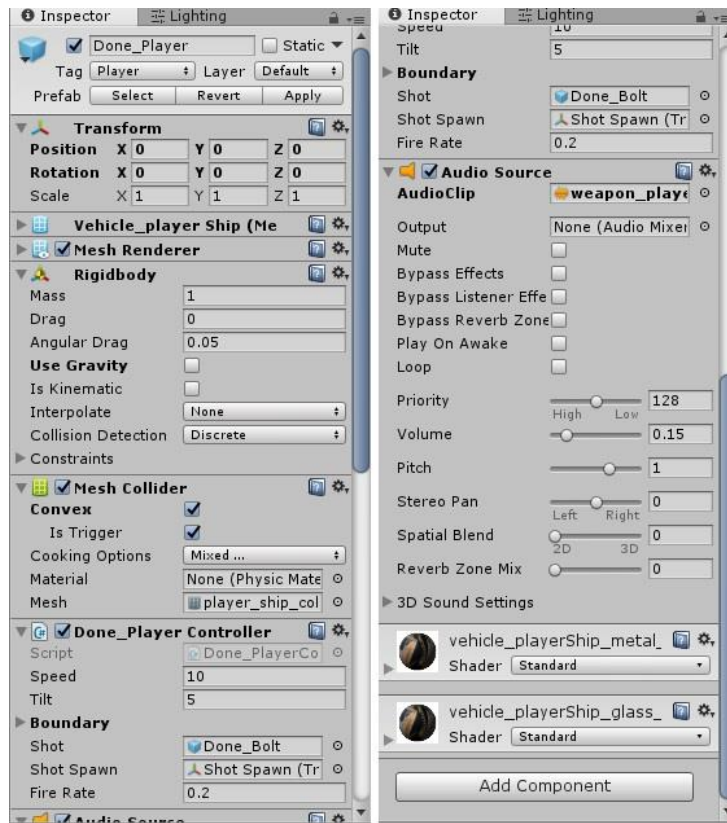


Figure 16 : Composantes du joueur.

Donc à partir de la gauche tout en haut on a la propriété de l'objet même du Joueur qui si décoché le désactive tout simplement, juste en dessous on a la composante Transform qui définit la position de l'objet actuelle dans la scène sur un repère cartésien, sa rotation ainsi que sa proportion, la composante juste en bas est la texture utilisé dans l'objet et le Mesh Renderer juste en dessous sert à l'instancier sur la scène, Le Rigidbody quant à lui gère la physique de l'objet ou on voit la Mass de l'objet , son frottement par rapport a l'air et aux autres surface, si il est régis par la gravité ou pas , et sa solidité (Kinematic), en dessous on a le Mesh Colider qui quant à lui gère tout ce qui est collision par rapport aux autres objets, ce qu'il faut savoir c'est qu'il existe sur Unity plusieurs types de Clodiers, certains plus simple et moins demandant en performance de calcule que d'autres, dans ce cas par exemple j'ai utilisé un Colider fournis avec l'objet du joueur qui s'applique à la forme du vaisseau, on peut y voir aussi la case Convexe et Is Trigger coché ce qui veut dire que l'objet en soit est susceptible de déclencher quelques chose lorsque il entre en contact avec un autre objet, et ce en faisant appelle à une fonction contenu dans un script par exemple, plus bas on peut voir le Done_Player_Controller qui est un script que j'ai écrit sous Visual studio afin de donner au joueur la possibilité de se mouvoir et de tirer, on peut y voir un champ speed qui est une au fait une simple variable public dans le script qui définit comme son nom l'indique la vitesse du joueur, Tilt quant à lui et le degré d'inclinaison du joueur, Boundary (qui est fermé pour le moment mais qui est en fait un classe interne disposant de variable globales déterminant la zone sur la quel le joueur a le droit de se déplacer on verra sa en détaille sur une photo un peu plus en bas) on a aussi un champ Shot ou y'a une objet Done_Bolt c'est au fait le Prefab d'un tire, ce Prefab je l'avais déjà créé et là je donne le Prefab au script afin qu'il puisse en instancier des copies sous une certaine condition, on a aussi Shotspawn ou j'ai donné une composante transforme d'un autre objet que j'ai créé sur la scène cette objet se trouve au front du vaisseau, l'emplacement de cette objet servira de point d'apparition aux

balles et donnera l'impression de tire, en dernier lieu dans le script on a une variable public Fire Rate qui quant à elle gère la cadence de tire.

Sur l'image à droite on a en plus une composante Audio Source ou on donne un fichier audio à jouer, la par exemple j'y ai décoché la case on Awake qui veut dire que le son va être joué dès que l'objet est instancié sur la scène, Loop qui veut dire que le son va se répéter, pour le moment l'audio source ne sert à rien je l'ai juste attaché, cette audio source contient le son de tire du vaisseau et sera appelé par le script du vaisseau à chaque fois que ce dernier instancie un objet de laser sur la scène, voyons voir en détail donc le script ainsi que ce qu'il contient



Figure 17 : Script du Joueur depuis Unity.

```
using UnityEngine;
using System.Collections;

//System serializable pour que unity prenne en compte la classe comme classe interne
[System.Serializable]
//Classe possédant le groupe de variable pour limiter les mouvements du joueur
public class Done_Boundary
{
    public float xMin, xMax, zMin, zMax;
}

public class Done_PlayerController : MonoBehaviour
{
    //Ces variable seront visible dans la composante du script de unity
    public float speed;
    public float tilt;
    public Done_Boundary boundary;

    //ces variables servent a accueillir l'endroit ou va apparaitre le tire et l'objet du tire en soit
    public GameObject shot;
    public Transform shotSpawn;
    //cadance de tire
    public float fireRate;

    private float nextFire;

    void Update ()//s'exécute a chaque frame aka tout le temps
    {
        //Bouton Fire1 est predifinit c'est le bouton Ctrl
        //ne tire que quand le temps actuelle depasse nexFire
        if (Input.GetButton("Fire1") && Time.time > nextFire)
```

Figure 18 : Script du Joueur 1/3.

Comme on peut le voir dans l'image au-dessus, on a donc deux classes la première étant une classe interne ou on ne fait que rassembler plusieurs variables faisant la même chose afin d'avoir une idée plus claire et que Unity puisse les rassembler plus proprement, (System.Serializable) sert à ce que Unity puisse reconnaître la classe à qu'on puisse modifier les variable depuis ce dernier (à condition qu'elle soit Public), en second lieu on a la classe Player_Controller qui contient 5 variables globales le Speed et le Tilt, qui eux seront visible à partir de Unity et seront modifiés durant les tests afin de donner un bon résultat.

```

void Update ()//s'exécute a chaque frame aka tout le temps
{
    //Boutton Fire1 est predéfinit c'est le bouton Ctrl
    //ne tire que quand le temps actuelle dépasse nexFire
    if (Input.GetButton("Fire1") && Time.time > nextFire)
    {
        //nextFire est égale au temps actuelle + la cadence de tire
        nextFire = Time.time + fireRate;
        //Fait apparaître Shot dans la position du shotSpawn avec sa rotation ses deux objets étant les var public en haut
        Instantiate(shot, shotSpawn.position, shotSpawn.rotation);
        //Récupère la son contenu dans la composante AudioSource de l'objet actuelle et le joue
        GetComponent().Play ();
    }
}

void FixedUpdate ()//S'exécute soit une fois, soit plusieurs fois soit pas du tout chaque frame
{
    //GetAxis Horizontal est prédéfinit et veut dire les fleches directionnelles
    float moveHorizontal = Input.GetAxis ("Horizontal");
    float moveVertical = Input.GetAxis ("Vertical");

    //Declaration d'un nouveau vecteur carthesien de mouvement sur l'axe X et Z seulement
    Vector3 movement = new Vector3 (moveHorizontal, 0.0f, moveVertical);
    //Modification de la vitesse de l'objet selon le vecteur de mouvement multiplié par une vitesse
    //( car le get axis génère seulement une valeur float entre 0 et 1 et faut l'amplifier
    GetComponent<Rigidbody>().velocity = movement * speed;

    //Récupère la position actuelle de l'objet et la compare par rapport a deux valeurs
    GetComponent<Rigidbody>().position = new Vector3
    (
        //Mathf.Clamp force la première valeur a rester dans le champ des deux autres

```

Figure 19 : Script du Joueur 2/3.

La photo suivante montre la fonction Update du script, sachant que les scripts sous Unity fonctionnent un peu de la même manière que le code Kotlin sous Android, ou on a des fonctions OnStart, OnCreate, ...etc. ou les fonctions s'exécutent sous certaines conditions par exemple ici sous Unity on a droit à Start() ou la fonction s'exécute dès l'objet auquel le script est attaché est instancié sur la scène, la fonction Update() s'exécute à chaque frame du jeu tandis que FixedUpdate() n'est pas obligé de s'exécuter, elle peut par contre s'exécuter plusieurs fois en une seule frame si le code le lui demande, dans la fonction Update() on a donc un If() qui vérifie si le bouton Fire est actionnée, après quoi il additionne le temps actuelle avec la cadence de tire qu'on va déclarer depuis Unity et à chaque If() il compare si le temps actuelle dépasse bien le temps précédent + la cadence, ce qui va créer un décalage du timing de la cadence entre chaque tire, après ça on Instancie l'objet Shot, avec la position de shotSpawn et sa rotation (ce dernier étant juste un objet vide se trouvant au front du vaisseau, en dernier lieu dans le If(), on joue le son qui est déjà attaché à l'objet du vaisseau ce qui donnera l'illusion d'un son de tire.

Dans le FixedUpdate() on reçoit deux Input par défaut qui sont « Horizontal » et « Vertical » ces deux désignant les flèches directionnelles, avec ces deux inputs on déclare un vecteur3 qui sera la composante de mouvement du joueur, après quoi on modifie la Vitesse de l'objet actuelle (le joueur) vise à vis de ce vecteur qu'on va amplifier par la variable public (accessible via Unity) Speed, vu que les valeurs générées par L'Input sont des Floats dont la valeur est comprise entre 0 et 1.

```

void FixedUpdate ()//S'exécute soit une fois, soit plusieurs fois soit pas du tout chaque frame
{
    //GetAxis Horizontal est prédefinit et veut dire les fleches directionnelles
    float moveHorizontal = Input.GetAxis ("Horizontal");
    float moveVertical = Input.GetAxis ("Vertical");

    //Declaration d'un nouveau vecteur carthesin de mouvement sur l'axe X et Z seulement
    Vector3 movement = new Vector3 (moveHorizontal, 0.0f, moveVertical);
    //Modification de la velocity de l'objet selon le vecteur de mouvement multiplié par une vitesse
    //( car le get axis genere seulement une valeur float entre 0 et 1 et faut l'emplifier
    GetComponent<Rigidbody>().velocity = movement * speed;

    //Recupère la position actuelle de l'objet et la compare par rapport a deux valeurs
    GetComponent<Rigidbody>().position = new Vector3
    (
        //Mathf.Clamp force la première valeur a rester dans le champ des deux autres
        Mathf.Clamp (GetComponent<Rigidbody>().position.x, boundary.xMin, boundary.xMax),
        0.0f, //le Y importe pas
        //meme chose pour le Z on donne une frontière pour le limiter
        Mathf.Clamp (GetComponent<Rigidbody>().position.z, boundary.zMin, boundary.zMax)
    );
    //Finalement pour plus d'esthetique on modifie la rotation de l'objet actuelle selon son mouvement
    //le Quaternion.Euler modifie l'inclinaison selon l'axe Z par rapport a la velocity de l'objet
    //et ce en l'amplifiant fois un nombre negatif afin qu'il tourne dans le sens inverse et qu'il donne
    //l'impression de se pencher dans le sens de la direction, le tilt definit de combien de degre ce dernier
    //va se pencher
    GetComponent<Rigidbody>().rotation = Quaternion.Euler (0.0f, 0.0f, GetComponent<Rigidbody>().velocity.x * -tilt);
}

```

Figure 20 : Script du Joueur 3/3.

Dans la figure 19 si dessus, on peut voir en dernier lieu qu'on modifier directement la position de l'objet actuelle et ce en mettant une fonction `Mathf.Clamp` pour la composante X ainsi que Z qui, la fonction `Mathf.Clamp` force une valeur à rester dans le champ de deux autres, la par exemple pour la position X je la force à rester en `boundary.xMin` ainsi que `boundary.xMax` ces deux valeurs je les ai implémenté depuis Unity (elles sont publique) en ayant effectué des tests.

Donc voilà ce fut le l'objet joueur dans ces moindres détails, par contre je ne pense à expliquer tous les autres Scripts car certains sont vraiment long (et là je pense particulièrement à `GameController` qui régit le system d'apparition aléatoire des monstres sur un certain axe, augmentant leur nombre à chaque fois ainsi que leur vitesse d'apparition, le `GameController`, gère aussi la fonction de `GameOver()`, l'affichage des boites de dialogues à l'écran et l'enregistrement du score.

D. Enemies :

Avant tout chose je tiens à souligner une différence entre l'objet du joueur et celui de l'ennemie, faut noter que seul le joueur est déjà instancié sur la scène dès le départ contrairement à l'ennemie qui lui est un objet Prefab, c'est une sorte de classe à la quel `GameController()`, va faire appel afin d'instancier des copies de ce dernier sur la scène.

Y'a en somme donc trois 4 types d'ennemies pour les 3 premiers ils ont la même fonction, ce sont les astéroïdes, la seule chose qui change en eux à chaque fois c'est leur texture, ils ont une trois script le premier leur permet de tourner dans un sens aléatoire avec une vitesse aléatoire compris entre deux valeurs qu'on lui donne, le second scripte fait qu'il bouge d'un dans un certain axe aussi afin que le joueur ai l'impression de vitesse, le dernier script quant à la lui est commun à tous les ennemis, c'est le script qui gère les collisions entre soit le joueur et l'ennemie, soit le tire du joueur et l'ennemie, selon ces deux premier une animation d'explosion sera jouée, ainsi qu'un son d'explosion.

```

void OnTriggerEnter (Collider other)
{
    if (other.tag == "Boundary" || other.tag == "Enemy")
    {
        return;
    }

    if (explosion != null)
    {
        Instantiate(explosion, transform.position, transform.rotation);
    }

    gameController.AddScore(scoreValue);

    if (other.tag == "Player")
    {
        Instantiate(playerExplosion, other.transform.position, other.transform.rotation);
        gameController.GameOver();
    }

    Destroy (other.gameObject);
    Destroy (gameObject);
}

```

Figure 21 : Script de collision.

Comme on peut le voir dans le code au-dessus on introduit la notions de tag ou on peut appeler un objet ou un groupe d'objets par un tag qui va les designers, on vérifie si Other.tag = Player par exemple si c'est le cas il fait apparaître l'animation d'explosion à l'emplacement de other, et appelle la fonction GameOver de gameController qui va arrêter le jeu et faire afficher l'écran de game over à la fin.

L'autre ennemie autre que les astéroïdes sont les vaisseaux ennemies, ces derniers en plus du script de mouvement, et du script de collision, ils ont un script IA qui leur permet de bouger aléatoirement et a fréquence aléatoire entre certains points.

L'ennemie a la possibilité d'instancier un tire d'une couleur différente aussi sur la scène et ce de manière aléatoire aussi.

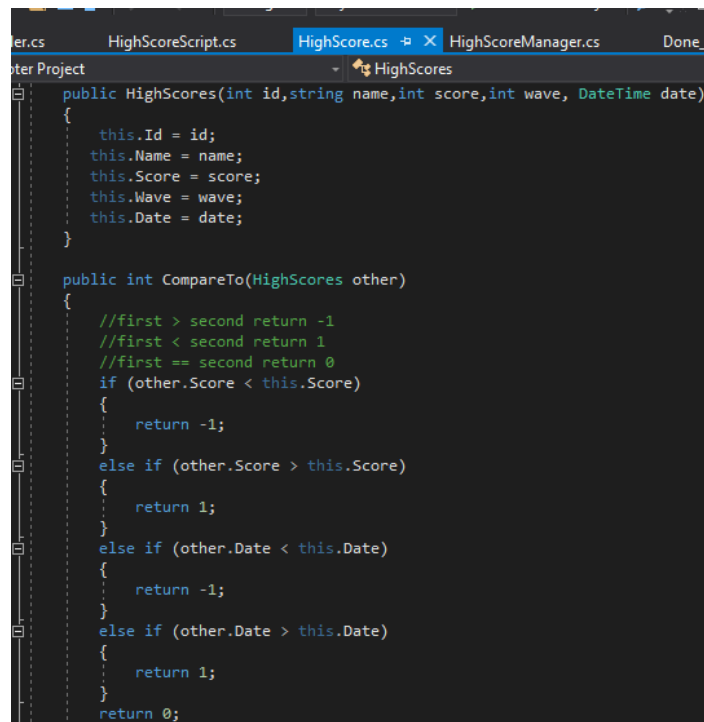
E. Audio :

Pour ce qui est des musiques dans le jeu elle sont géré par GameController(), GameController() quant à lui étant trop long, il faut juste noter qu'il y'a un objet instancié invisible sur la scène au quel GameController() est attaché, cette objet dispose aussi des musiques dans le jeu, et GameController() ne fait que les jouer, par contre dans le menu c'est pas GameController() qui prends le relais mais HighScoreManager() qui joue les musiques, ce dernier lorsque on appuie sur un bouton ce dernier fait appelle à une fonction de HighScoreManager qui elle change de scène et passe à la Main scène et donc le jeu en lui-même.

F. Scores :

Les scores sont donc sauvegardée, dans une base de donnée SQLite3, pour ce faire j'ai dû utiliser quelques fichiers DLL fournis par Sqlite, ainsi que Unity, l'enregistrement du score s'effectue sur l'écran de GameOver ou le joueur aura la possibilité d'écrire son nom afin de sauvegarder son score sans quoi il le sera pas, le score contiens donc le nom du joueur, le

score lui-même et le numéro de la vague à la quel le joueur été arrivé, les scores seront aussi trié afin que le premier de la liste soit le plus grand score et ce automatiquement, on peut aussi limiter le nombre de scores pouvant être enregistré et faire une liste des top 10 scores par exemple, la récupération de scores s'effectue en tout grâce à trois scripts, l'un récupérant les scores dans la base de donnée, l'autre définissant l'objet Score dans une liste, et enfin le dernier écrivant les scores dans les champs, les champs afin d'écrire les scores sont aussi un objet Prefab qui sera placé toujours de la manière dont il a été construit la seule chose qui changera chez lui chaque fois est son contenu.

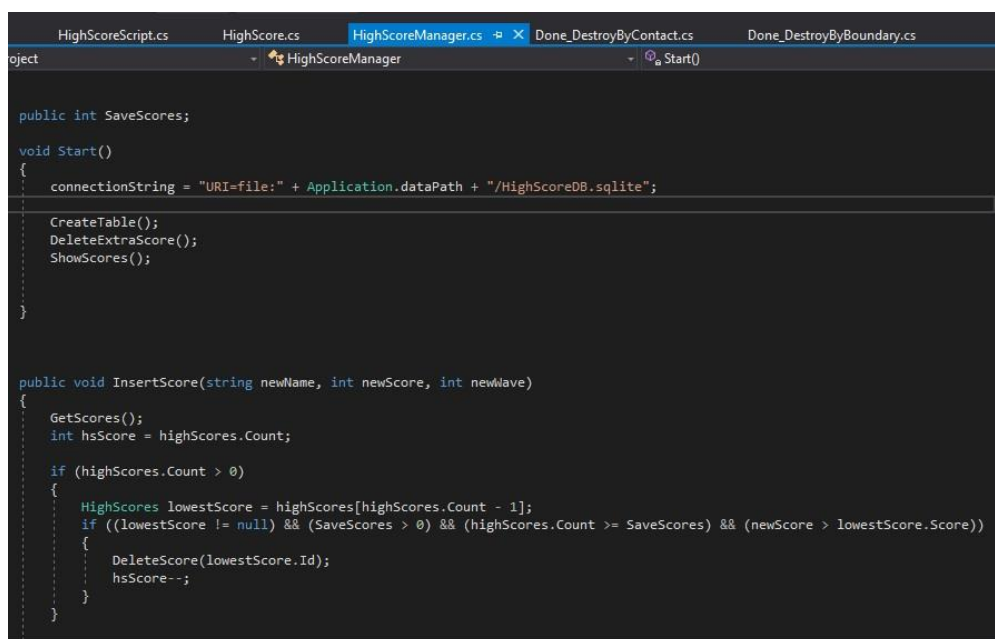


```
public HighScores(int id, string name, int score, int wave, DateTime date)
{
    this.Id = id;
    this.Name = name;
    this.Score = score;
    this.Wave = wave;
    this.Date = date;
}

public int CompareTo(HighScores other)
{
    //first > second return -1
    //first < second return 1
    //first == second return 0
    if (other.Score < this.Score)
    {
        return -1;
    }
    else if (other.Score > this.Score)
    {
        return 1;
    }
    else if (other.Date < this.Date)
    {
        return -1;
    }
    else if (other.Date > this.Date)
    {
        return 1;
    }
    return 0;
}
```

Figure 22 : HighScore Script.

Ce dernier fait la comparaison des scores et c'est grâce à lui qu'ils sont stockés dans l'ordre.



```
public int SaveScores;

void Start()
{
    connectionString = "URI=file:" + Application.dataPath + "/HighScoreDB.sqlite";

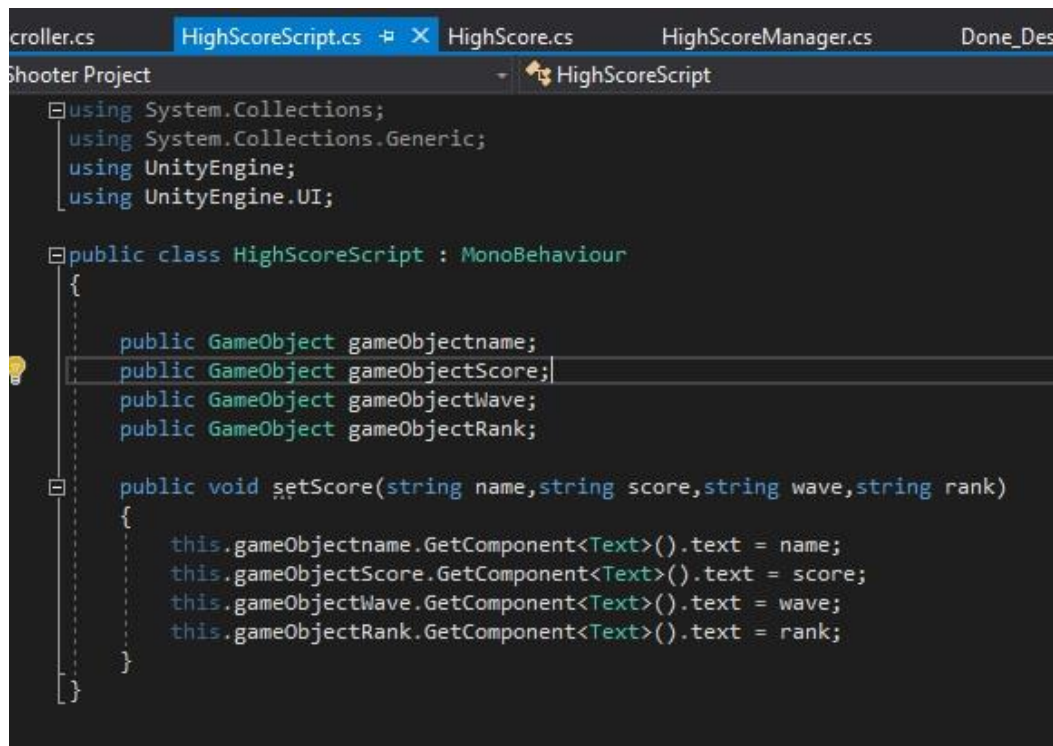
    CreateTable();
    DeleteExtraScore();
    ShowScores();
}

public void InsertScore(string newName, int newScore, int newWave)
{
    GetScores();
    int hsScore = highScores.Count;

    if (highScores.Count > 0)
    {
        HighScores lowestScore = highScores[highScores.Count - 1];
        if ((lowestScore != null) && (SaveScores > 0) && (highScores.Count >= SaveScores) && (newScore > lowestScore.Score))
        {
            DeleteScore(lowestScore.Id);
            hsScore--;
        }
    }
}
```

Figure 23 : HighScoreManager Script.

Je montre brièvement HighScoreManager car état vraiment trop long pour tout détailler dedans, le script en soit étant généralement des connexions Sqlite et de l'exécution de certains Query, on voit dans l'image qu'il trouve la base de données selon l'emplacement actuelle de l'application et qu'il interroge directement le fichier Sqlite.



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class HighScoreScript : MonoBehaviour
{
    public GameObject gameObjectName;
    public GameObject gameObjectScore;
    public GameObject gameObjectWave;
    public GameObject gameObjectRank;

    public void setScore(string name, string score, string wave, string rank)
    {
        this.gameObjectName.GetComponent<Text>().text = name;
        this.gameObjectScore.GetComponent<Text>().text = score;
        this.gameObjectWave.GetComponent<Text>().text = wave;
        this.gameObjectRank.GetComponent<Text>().text = rank;
    }
}
```

Figure 24 : HighScoreScript.

Ce dernier scripte est appelé par HighScoreManger lui aussi et ne fait que coller les résultats actuels dans la composante Text qu'on va lui donner depuis Unity vu qu'elle est Public.

G. Menu :

Tout ce qui est interface sur Unity est géré par ce qui est appelée des Canvas, une sorte de system qui régit tout ce qui est UI.

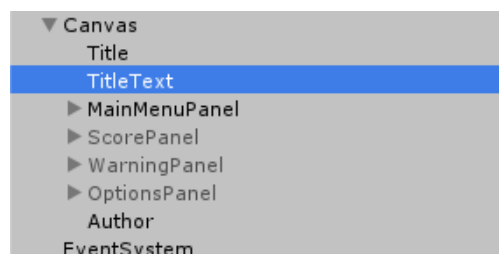


Figure 25 : Canvas.

Les boutons quant à eux ont un script qui leur est propre, ce script leur permet de soit effectuer des fonctions de base sur Unity tel que désactiver ou activer un objet, ou d'appeler une fonction particulière contenu dans un certain Script

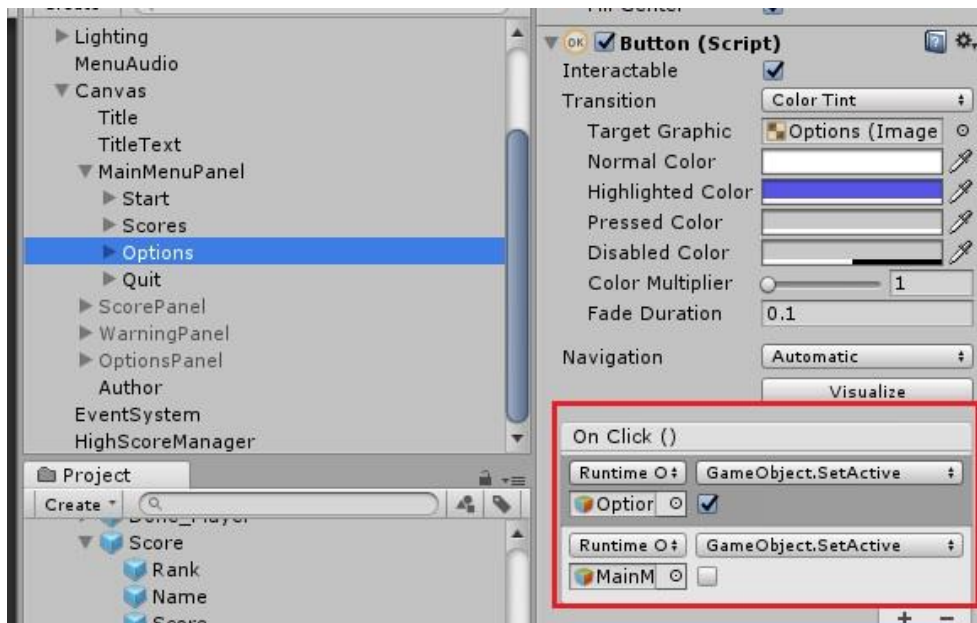


Figure 26 : Fonctions Bouton Options.

H. Quelques scripts en plus :

En plus donc de ces scripts nécessaire au fonctionnement du jeu j'ai dû ajouter certains scripts qui n'ont été que esthétique, on a par exemple le défilement de l'arrière-plan grâce BGScroller qui fait défiler une copie de l'image de l'arrière plant afin de donner une illusion de mouvement.

```

using UnityEngine;
using System.Collections;

public class Done_BGScroller : MonoBehaviour
{
    public float scrollSpeed;
    public float tileSizeZ;

    private Vector3 startPosition;

    void Start ()
    {
        startPosition = transform.position;
    }

    void Update ()
    {
        float newPosition = Mathf.Repeat(Time.time * scrollSpeed, tileSizeZ);
        transform.position = startPosition + Vector3.forward * newPosition;
    }
}
  
```

Figure 27 : BGScroller Script.

RandomRotator qui fait tourner les astéroïdes dans des sens aléatoires et a des vitesses aléatoires.

```

using UnityEngine;
using System.Collections;

public class Done_RandomRotator : MonoBehaviour
{
    public float minTumble;
    public float maxTumble;

    void Start ()
    {
        GetComponent<Rigidbody>().angularVelocity = Random.insideUnitSphere * Random.Range(minTumble, maxTumble);
    }
}

```

Figure 28 : RandomRotator Script.

DestroyByTime du quel je n'ai pas parlé mais qui fait la destruction de l'animation d'explosion elle-même et ce en la faisant disparaître avec un certain temps limite

```

using UnityEngine;
using System.Collections;

public class Done_DestroyByTime : MonoBehaviour
{
    public float lifetime;

    void Start ()
    {
        Destroy (gameObject, lifetime);
    }
}

```

Figure 29 : DestroyByTime Script.

QuitOnClick qui ferme le jeu, ce script est appelé par le bouton Quit

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class QuitOnClick : MonoBehaviour {

    public void Quit()
    {
        Application.Quit();
        //UnityEditor.EditorApplication.isPlaying = false;
    }
}

```

Figure 30 : BGScroller Script.

LoadSceneOnClick qui est appelé par le bouton Play lorsqu'on clique sur ce dernier.

I. Construction du jeu :

C'est là qu'intervient toute l'ingéniosité d'Unity, ce dernier nous permet de compiler le jeu en destination de plusieurs plateformes et ce juste en téléchargeant les bibliothèques nécessaires afin de mener cette tâche à bien.

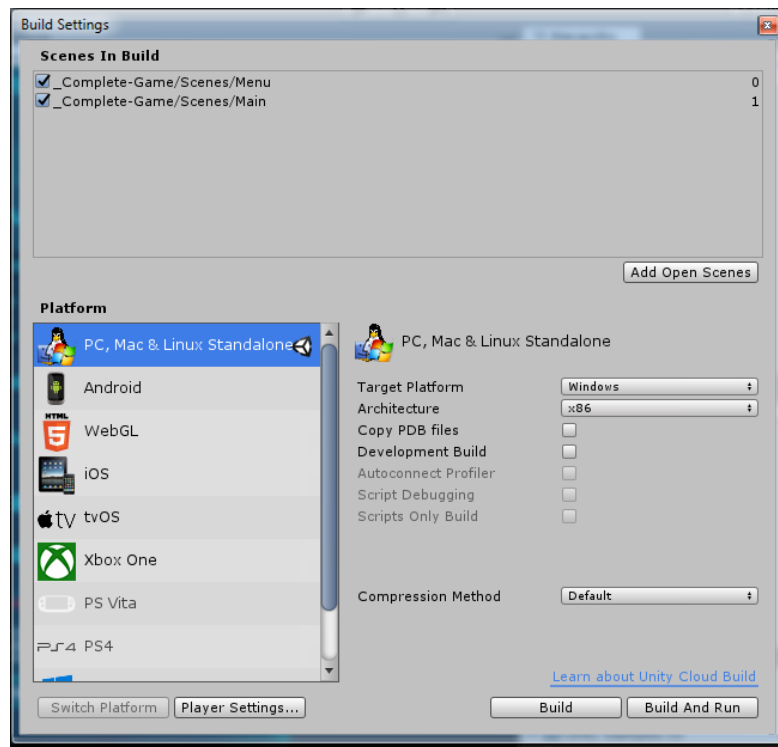


Figure 31 : Build Game.

On se retrouve donc si tout marche bien (et si tout ce compile bien) à la fin ce Build avec un Exécutable ainsi que quelques autres fichiers et on est donc prêt à lancer le jeu.

V. Conclusion :

A la fin de ce projet qui m'aura quand même pris beaucoup de temps, j'aurai acquis beaucoup d'expérience par rapport à l'utilisation de Unity, et du codage en C#, je me rends compte aussi à quel point Unity est puissant et est très complet c'est pourquoi je le conseil a tout le monde, en ce qui concerne le C# je trouve pas avoir eu beaucoup de difficulté vis-à-vis de ce dernier (ressemble au Java) je trouve qu'au contraire c'était la partie la plus aise à condition de bien sur trouver les bonne fonctions, sinon je compte améliorer le jeu afin d'y ajouter de nouvelles fonctionnalités et de contenu afin d'obtenir le résultat que je désire de ce dernier.