# 1. Menu

/!\ Yanis et Clément ont réalisé le menu en binôme de leurs côtés. Lors de la mise en commun des projets, le choix a été fait de garder le menu réalisé par Alex et Lucas.

## __ Système solaire __

*Répartition* - 50% Lucas - 50% Alex
*Scène* - Menu/scene/planet.tscn

Pour créer les planètes du système solaire, on créer une scène planète qui peut prendre en paramètre divers attribut (taille, distance du soleil, lumière, ect . . .). Les planètes sont générées selon un fichier de config où toutes les données sont initialisées.

## __ Caméra __

*Répartition* - 40% Lucas - 60% Alex
*Script* - Menu/script/camera.gd
*Scène* - Menu/scene/camera.tscn

La caméra est initialisée sur la vue Soleil mais peut être changée grâce à l'interface sur les différentes planètes.

Les flèches du clavier permettent de faire tourner la caméra au tour de la planète visée. Cette touche en particulier est mise en dur dans le code et n'est pas configurée depuis les paramètres de l'application.

## __ Interface __

*Répartition* - 60% Lucas - 40% Alex
*Script* - Menu/script/interface.gd
*Scène* - Menu/scene/interface.tscn

L'interface a plusieurs fonctionnalitées :

- On peut choisir une planète. Quand on en choisit une, la caméra s'installera vers sa position et le nom / description de la planète est écrit dans les label initialement vide. Ses données sont récupérées depuis l'autoload, grâce au config file. On montre également le bouton permettant de lancer le jeu. Pour déterminer quel jeu est lancé, chaque planète a (dans le conf file) une donnée pour le jeu.

- L'interface permet également d'accéder aux options du jeu et aussi de pouvoir le quitter.

## __ Setting __

*Répartition* - 100% Lucas

*Script* - Menu/script/setting_panel.gd
Ressources/autoload/AutoloadSetting.gd

On peut accéder aux options depuis le menu afin de gérer le volume du son en jeu et des FX. Les données sont stockées dans un fichier de configuration. Pour optimiser les performances du jeu on écrit sur le disque seulement à la fermeture du panneau des options.

# 2. Shooteur

## __ Game _

*Répartition* : 40% Lucas - 60% Alex

**Monde torique**
*Script* : Shooter/script/toreWorldRigidBody.gd

On a réalisé un monde torique en utilisant un objet Area qui englobe l'ensemble du jeu. Si un objet (débris, vaisseau, laser) sort de la zone, il est téléporté de l'autre côté du monde.

**Pause**
*Répartition* : 70% Lucas - 30% Alex
*Scène* : Shooter/scene/GUI/pause_panel.tscn
*Scripte* : Shooter/script/GUI/pause_panel.gd
Ressources/autoload/AutoloadShooter.gd ( a partir L 123)

En jeu, il est possible de passer sur un écran de pause avec la touche "echap" qui a été mappée. un second appuis relancera la partie. En pause, un panel avec différentes options est ajouté au tree.
il est possible de reprendre la partie, sauvegarder, et quitter, et quitter sans sauvegarder. Si l'on sauvegarde avant de quitter, a la prochaine partie les données seront chargées depuis le fichier de sauvegarde et la partie commencera à dans l'état qu'elle était au moment où le joueur à passer le niveau. Il est important de noter que les données (vie, score, énergie) sont stockées de manière temporaire dans un dictionnaire à chaque passage de niveau . a la mort du joueur, le fichier de sauvegarde est vidé.

il est important de noter qu'il est possible dès "tricher" est relançant la partie juste avant de mourir pour ne pas avoir à recommencer.

**Niveau**
*Répartition* : 40% Lucas - 60% Alex

*Script* : Ressources/autoload/AutoloadShooter.gd, ligne 82

Le jeu gère un système de montée de niveau qui devient de plus en plus difficile. Le niveau 1 génère 2 gros débris et le nombre augmentera de deux à chaque montée de niveau. Également la vitesse d'apparition des débris augmentera, ainsi que leur vélocité.

Cependant, un erreur dont nous n'avons pas pu déterminer la source peut subvenir lors d'un passage de niveau. Bien que le joueur ait détruit tous les débris, le jeu ne passe pas au niveau suivant. En réalité, le jeu considère que des débris sont encore en vie. Cette erreur n'arrive pas toujours et est donc compliquée à analyser.

# __ Player __

**Déplacement**
*Répartition* : 65% Lucas - 35% Alex
*Script* : Shooter/script/vaisseau.gd (ligne 34)

Pour déplacer notre vaisseau, des touches ont été configurées et deux configurations sont initialisées à la base : ZQSD et les flèches haut, droit, gauche et arrière.
Pour les mouvements avant et arrière, on a utilisé la méthode apply_force et apply_torque pour la rotation.

Une amélioration possible du déplacement est d'empêcher le mouvement arrière qui d'un physique n'est techniquement pas possible. L'idée serait que la touche "arrière" réalise un freinage du vaisseau mais nous n'avons pas eu le temps de nous pencher sur cette fonctionnalité.

**Laser**
*Répartition* : 30% Lucas - 70% Alex
*Script* : Shooter/script/lazer.gd
*Scène* : Shooter/scene/lazer.tscn

La touche espace est la touche configurée à l'origine pour permettre au vaisseau de tirer un laser. Ce laser partira droit devant en fonction de la position du vaisseau. Il est affecté par le monde torique et interagit avec les débris et le vaisseau pour leur infliger des dégâts. Pour réaliser ses interactions, on a utilisé le layer de collision.

Les laser possèdent une certaine vitesse que le vaisseau peut dépasser à partir d'un certain point. Cela fait que le vaisseau prend ses lasers avant que celui-ci ne parte en avant. On a découvert ce problème tard et nous n'avons pas eu le temps de le régler.

# __ Débris __

On a créé une classe générale de débris qui est étendue par les autres débris plus spécifiques comme Gros Débris.

**Apparition**
*Répartition* : 50% Lucas - 50% Alex

Pour faire apparaître les débris a des positions aléatoires, on initialise un dictionnaire dans l'autoload qui représente un x ou y des 4 coté de l'écran.
→ Ressources/autoload/AutoloadShooter.gd, ligne 47

Ensuite on tire aléatoirement une de ses positions et, on adapte son mouvement (réaliser à l'aide d'apply force), lui aussi générer aléatoirement un vecteur, qui en fonction du 1er résultat, ne fera pas sortir le débris de l'écran
→ Shooter/script/debris/debri.gd, ligne 55

**Destruction**
*Répartition* : 70% Lucas - 30% Alex

Chaque débris possède un nombre de PV. Si celle-ci tombe à zéro, le débris se subdivise en deux débris plus petits (hormis pour le petit débris). On a deux marqueur pour les débris afin de faire apparaître les leur sous-division.
→ Shooter/script/debris/gros_debris.gd, ligne 13
Pour leur mouvement initial, il est générée de façon aléatoire de la même façon de l'apparaition.
→ Shooter/script/debris/debri.gd, ligne 55

Il arrivait que certains débris lors de leur division, apparaissent en dehors du monde et ne soient jamais re téléporté. Pour être sûr de régler ce défaut, on a fait que si un débris à une position trop éloigné du monde, il est quand même re téléporté
→ Script debris.gd, ligne 15.

# __ Bonus __

*Répartition* : 50% Lucas - 50% Alex
*Script* : Shooter/script/bonus/bonus.gd

En jeu, il est possible de débloquer 2 types de bonus en détruisant un débris: extraLife et extraEnergy.
ExtraLife redonne 1 point de vie et extraEnergy remplit entièrement la jauge d'énergie.
Ces deux bonus héritent du composant "Bonus" contenant le comportement de base d'un bonus. Au bout de 5 secondes, s'il n'est pas récupéré, il disparaît.
Leurs taux d'apparition sont gérés à la destruction du débris.

# 3. Survivor

## __ Pourquoi un survivor ? __

C’est un style de jeu qu’on apprécie, mais c’est aussi un type de jeu facilement adaptable à n’importe quel univers et simple à prendre en main.

## __ Joueur __

*Répartition* : Clément: 55% - Yanis: 45%

pas de hurtbox -> le joueur ne peut pas infliger de dégâts au contact, uniquement avec les améliorations
Avec le joueur, on peut récupérer de l’xp, ce qui donne des améliorations, une fois un niveau supérieur atteint.
Le joueur peut se déplacer à l’aide des flèches ou des touches zqsd (ATTENTION! on utilise “up” “down” “left” et “right”, pas des touches…
donc une fois l’input map mise en place, on pourra personnaliser les touches
(on ne sait pas encore si on aura le temps de le faire avant de rendre le projet)

## __ Ennemis __

*Répartition* : Clément: 65% - Yanis: 35%

Les ennemis ont une hurtbox pour infliger des dégâts et une hitbox pour en encaisser implémentation du spawn des ennemis simple à rajouter et à configurer ennemis visuellement différents avec des stats différentes.

Plus le temps passe, plus on ajoute de nouveaux ennemis. Ils seront plus rares, mais plus rapides, avec plus de dégâts et plus de pv. Ils donneront aussi plus d'expérience.
Quand on tire sur un ennemi, il recule sous le choc.
ex:l.52 script T-blue

Les ennemis se dirigent toujours vers le joueur

# __ Items __

*Répartition* : Clément: 60% - Yanis: 40%

Les items ont deux types: “actifs” et “passifs”.
Les items actifs changent la manière de jouer au jeu en créant ou améliorant un nouveau type d’attaque, ce qui pousse le joueur à se déplacer différemment.

Par défaut, le joueur possède une attaque qui vise aléatoirement un des ennemis,
ex: script player l.178

Ce qui rend le joueur faible en début de partie, et l’oblige à éviter les monstres, pour récupérer l'expérience au sol.

**Actif**

*Les drones* : Le joueur doit marcher vers les ennemis pour que les drones attaquent dans sa direction, et transpercent les ennemis, permettant de prendre plus de risque, mais récupérer de l'expérience plus facilement/se frayer un chemin à travers les ennemis.

*Le sabre à lancer* : le sabre (certainement animé par la Force) se déplace aléatoirement vers des ennemis, avant de se rapprocher du lanceur. difficile de récupérer l'expérience dans des gros groupes d’ennemis, mais permet de faire beaucoup de dégâts.

**Passifs**

*Améliorations de stats* : vitesse, cooldown des sorts, réduction des dégâts

*L’anneau* : Augmente le nombre d’attaques de 1 par niveau.
C’est l’objet le plus puissant, et il changera d’effet selon la planète (peut être lié au scénario).

# __ Carte __

*Répartition* : Clément: 50% - Yanis:50%

La carte est composée d’une texture qui se répète, et est pré-générée. la taille est prédéfinie

Force *:* modulable, légère

Faiblesse :
La map n’est pas infinie, il faudrait regarder la courbe d’xp sans amélioration, prendre en compte qu’on ne prend que des améliorations de speed en allant que dans une seule direction, pour avoir la taille maximale réaliste.
On pourrait aussi faire bouger la map avec le personnage.
Aussi, la carte est relativement laide.

## __ Son __

*Répartition* : Clément: 10% - Yanis: 90%
musique de fond
explosion à la mort des ennemis
bruitage quand on ramasse de l'expérience, level-up, choix d'une amélioration, mort du personnage ou victoire

Forces :
Simple et fonctionnel.
Même bus que sur la partie shooter, modifier le son sur le menu ou dans le shooter le modifiera ici aussi.

Faiblesses :
Musique répétitive, pas de bruit de monstres.
Pas éditable depuis le survivor, car on n'a pas mis de menu de pause.

## __ Textures __

*Répartition* : Yanis: 100%

Texture au sol simple, donc modifiable rapidement-> possibilité d'adapter les niveaux aux planètes si nécessaire

Choix d'un thème (très) inspiré de Star Wars pour le personnage et les améliorations -> permet de garder le même design de personnage et améliorations quelle que soit la planète, tout en permettant de rester libre sur le design des ennemis (après tout l'univers est infini)
menus simples

Forces :
Simple, pas trop ancré, permettant de ne pas se limiter si on souhaite poursuivre le jeu.

Faiblesses :
Un peu trop simple, peu de monstres ayant de la personnalité (à part T-blue, il est très beau).

## __ Autres __

*Répartition* : Clément: 50% - Yanis: 50%

**Animation**
Chaque texture de monstre ou joueur est divisée en deux parties, la frame 1 et 2, on les alterne à interval régulier pour simuler la marche, en jouant l'animation walk.
ex: l.25 script T-blue

**Signaux**
on utilise de nombreux signaux pour transmettre les informations:hurt quand une hurtbox entre dans une hitbox, soit quand un ennemi touche le joueur.

ex: l.26 script T-blue

La mort est inévitable. Donc pour gagner, on passe par la fonction death, si on ne voit pas l'écran, c'est que ce sont les ennemis qui sont morts, donc on gagne. death, player l.359
(Dans une partie normale, vous devrez avoir tout juste fini de tuer tous les ennemis au bout de 5 minutes, le temps de fin)

# 4. Autre

## __ Fusion des deux jeux __

*Répartition* : Alex: 80% (Fonctionnel) - Yanis : 20% (Interface)

Les deux jeux ayant été réalisés séparément, un travail a dû être nécessaire afin de les fusionner en un projet. Ainsi, il y a du être revus plusieurs éléments notamment la configuration des touches et autoload, l'interface et la gestion des fichiers.

## __ Problème de comptabilité __

Vers la fin du développement, nous avons remarqué que le projet ne fonctionnait pas sur les versions récentes de Godot (4.2). Après analyse, nous avons remarqué que c'était dû à changement de comportement de fonction (change_scene_to_file). Par manque de temps, nous n'avons pas pu régler le problème mais nous l'avons prise en compte.

### Mot de la fin :

On a été super gentils avec vous!
(on accepte les paiements par chèques, en espèces, ou en points si vous voulez)