Game Document

Equipe

Guillaume
QUERTAIN

Lead Game Designer
Sound Designer
Level Designer - Artist

Noé
LYON

Direction Artistique
Technical Artist
Game Artist

Gregory
DOS SANTOS

Game Programmer
Game Design
Integration Sound Design

Table des matières

OVERVIEW

Contraintes

La contrainte imposé pour ce projet est de créer un noyau système fonctionnel et ludique puis de le convertir en jeu.

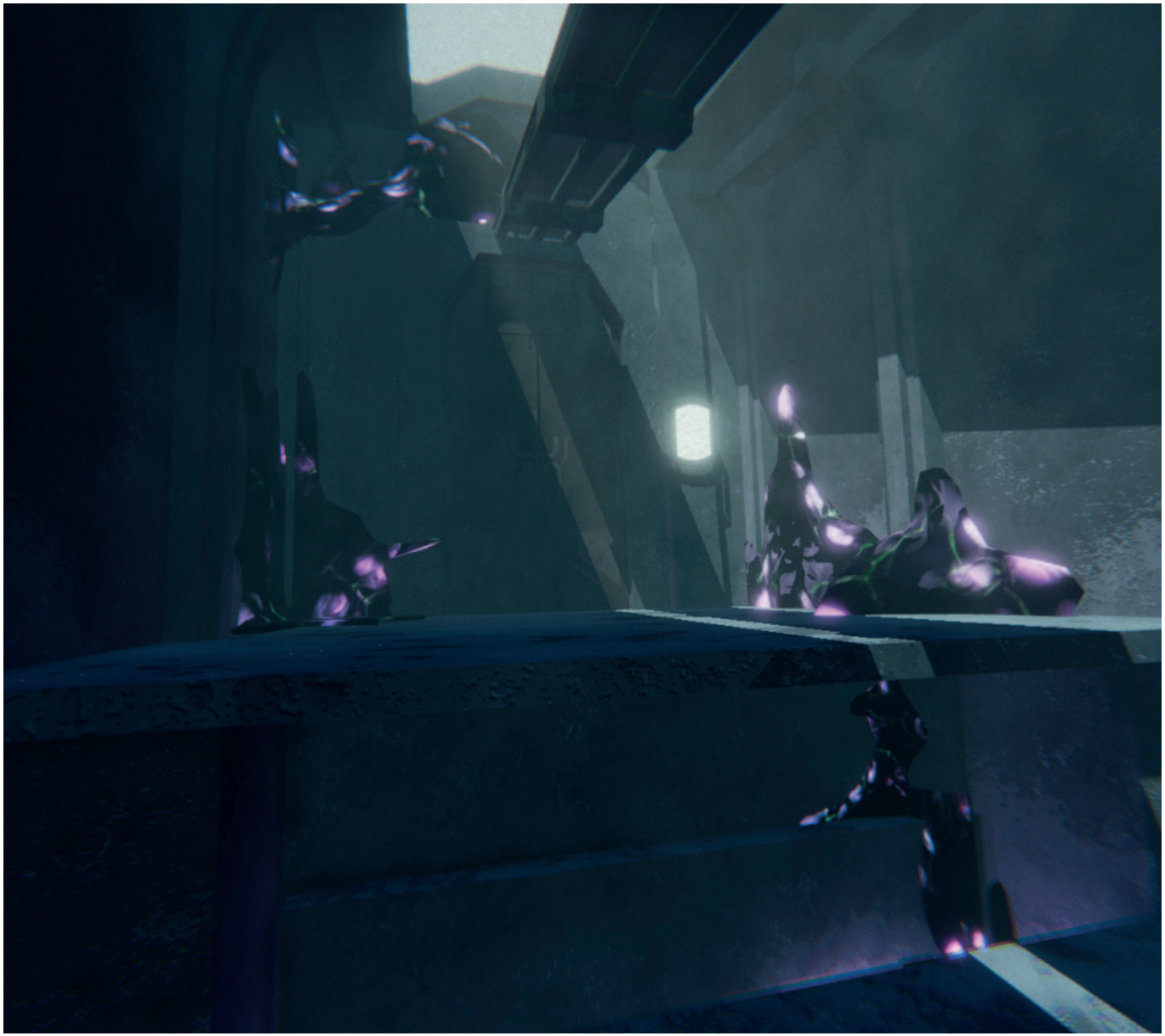
Cette façon de fonctionner permet de s'assurer que le Core Gameplay est efficace avant de se lancer dans la production d'un jeu.

Sujet

Le sujet étant **Etat Critique**, c e dernier est assez large et permet de s'exprimer assez librement.

L'état critique est défini par l'instant où un système est à la limite de passer d'un état à un autre sans que de retour en arrière soit possible.

Cette bascule peut être autant positive que négative mais dans tous les cas, elle vient créer une tension impliquant le joueur



Expérience

Nous souhaitons proposer une expérience frénétique, rapide, satisfaisante et évolutive. Le joueur doit appréhender, comprendre et maîtriser les mécaniques pour pouvoir profiter au mieux du jeu mis à sa disposition.

L'idée est que le joueur se rende rapidement compte de ce que les outils mis à sa disposition lui permettent de faire et qu'il les exploitent pour finir au plus vite les niveaux dans lesquels il évolue.

Il y a peu de mécaniques mais ces dernières permettent une grande liberté de mouvement aux joueurs. Chaque joueur peut donc exploiter les mécaniques comme bon lui semble pour terminer chaque niveau.

Enchaînement des actions

Pour que l'expérience souhaité puisse être atteinte, il est important de mettre au centre du gameplay enchaînement entre les actions du joueur. Si les enchaînements sont fluides et rapides, alors l'expérience le sera aussi.

Le centre du gameplay est basé sur l'enchainement entre les actions. Lancer, tirer, lancer, viser, se déplacer, viser... Tout est une succéssion d'action bien executé. Le droit à l'erreur est faible. L'enchainement est donc centrale et essentiel pour progresser.

Lien entre les mécaniques

Pour que le gameplay soit fluide, il faut que les mécaniques soient cohérente et intuitives. Il est donc important que le joueur comprenne rapidement comment fonctionne chacune d'entre elle et leurs intéractions.

Même si le jeu ne met pas a disposition un multitude de mécaniques complèxes, il est important que les mécaniques présentent soient cohérentes et liés entre elles.

Maitrise du Gameplay

L'enchaînement des actions et le coté dynamique de notre jeu viennent rapidement demander une compréhension et une maitrise des outils. Le but n'est donc pas de donner des outils trop complexe au joueur.

Le choix de s'orienter vers un jeu de speedrun axe le gameplay autour de ses mécaniques et de leur bonne utilisation.

Pitch

Lancez- vous dans une course effrénée contre la mort pour nettoyer des zones des parasites intrusifs.
Prenez de la vitesse, conservez la et téléportez vous pour atteindre des zones innaccéssibles normalement.

Type de Jeu

Fast FPS / Fast Platformer

Cible

Hard Core ayant une tendance complétionniste et / ou compétitive.

Univers

Shift prend place dans un univers froid et épuré inspiré du brutalisme. Ce monde est corrompu par des parasites étranges et inquiétant.

Game Concept

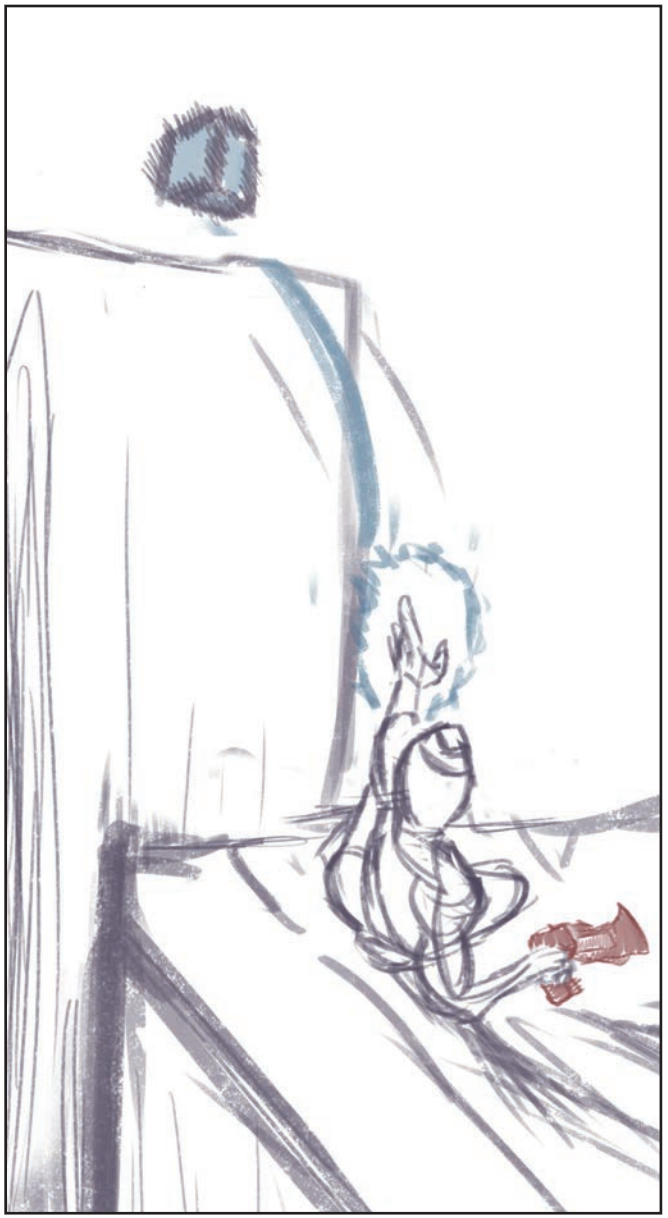
Dans Shift, le joueur incarne cyborg agile et rapide employé par une entreprise de nettoyage. Son but est de débarasser des zones des parasites occupant les lieux.

L'avatar à la capacité de **courir**, **glisser**, **sauter** et lancer un projectile sur lequel il peut se **téléporter**. Cette dernière action consomme des charges rechargeable en tirant sur les parasites présents dans les niveaux. Toutes ces actions lui permettent de prendre de la vitesse pour atteindre au plus vite la fin de la zone.

Présentation du gameplay

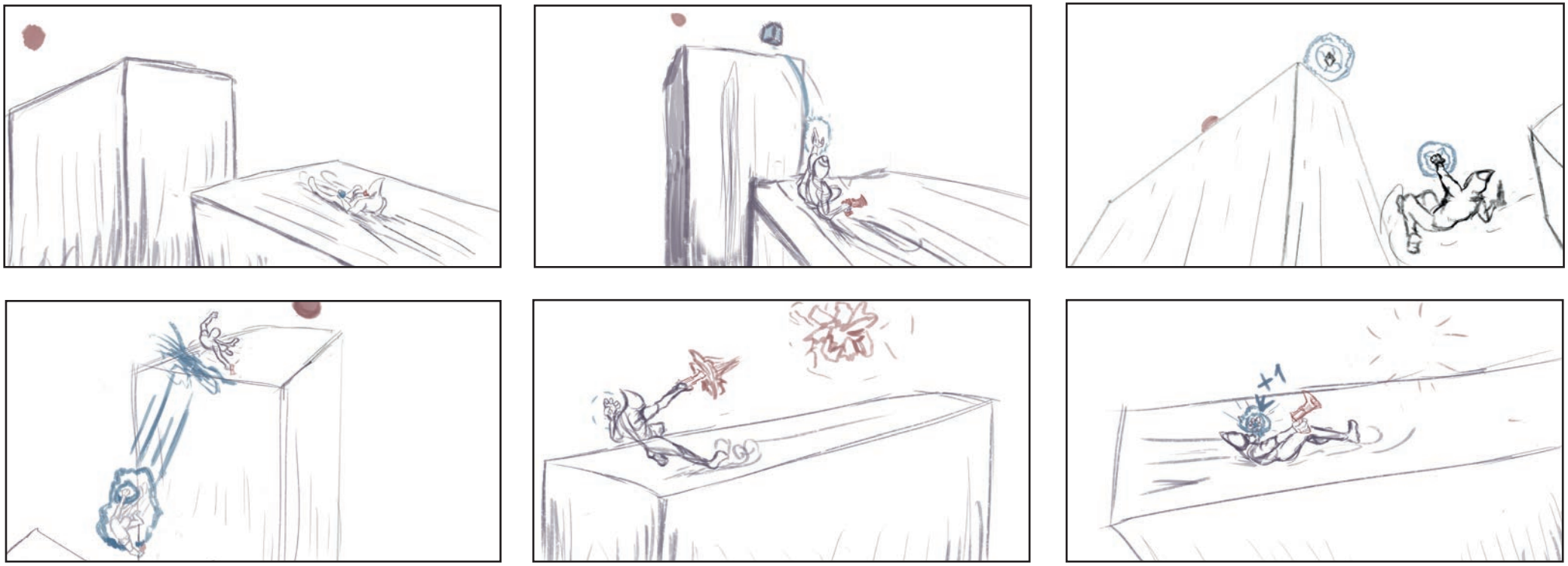
Dans Shift, nous cherchons à créer un jeu de speedrun dans lequel le bon enchainement des actions mène à la réussite.

Le jeu se déroule à la première personne. L'avatar a des outils de déplacement, un saut et une glissade qui permet de conserver de la vitesse et un téléporteur qu'il peut lancer et dont il prend la vélocité lors de la téléportation.



Gameplay

Prenez de la vitesse et conservez la pour traverser les niveaux le plus rapidement possible. Téléportez vous ou tirez sur des cibles pour récupérer des charges de téléportation et accéder à de nouvelles zones. Toute chute sera fatale.



Caméra

Plus de détails dans la partie Game Design du document.

La caméra est en point de vue première personne. Cela signifie que l'on voit ce que notre avatar perçoit. Ce point de vue permet une plus grande immersion et un meilleur controle de sa position dans l'espace.
Ce choix donne entièrement la caméra au joueur et il faut donc forcer, d'une manière ou d'une autre, a voir les choses.

La caméra de Shift est réactive à la vitesse et à l'état dans lequel l'avatar se trouve (téléportation, glissade, course). Le FOV, les contrastes, l'abération chromatiques sont des effets sur lesquels appuyer pour donner cet impressions de vitesse.

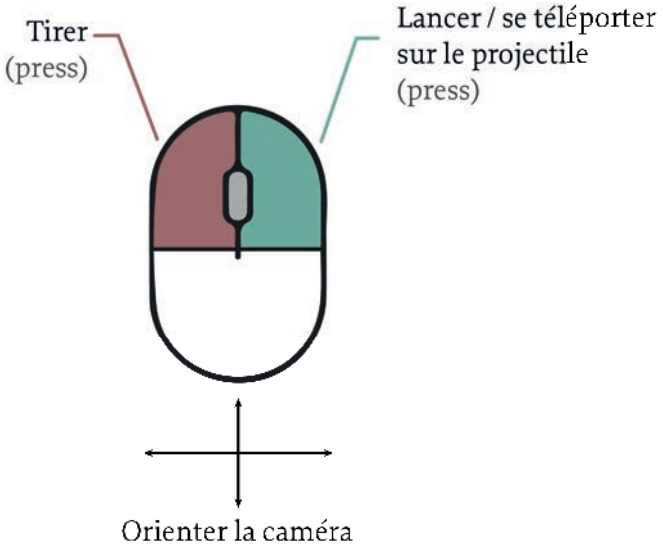
Le joueur doit pouvoir voir les bras de son avatar pour constater dans quel état système il se trouve, autrement dit, combien de charge il dispose. Il doit aussi pouvoir voir combien de temps il lui reste pour terminer le niveau.

Contrôles

Souris

Pour les contrôles souris, nous avons adopté les conventions du genre. Clique droit pour le tir et clique gauche pour la capacité.

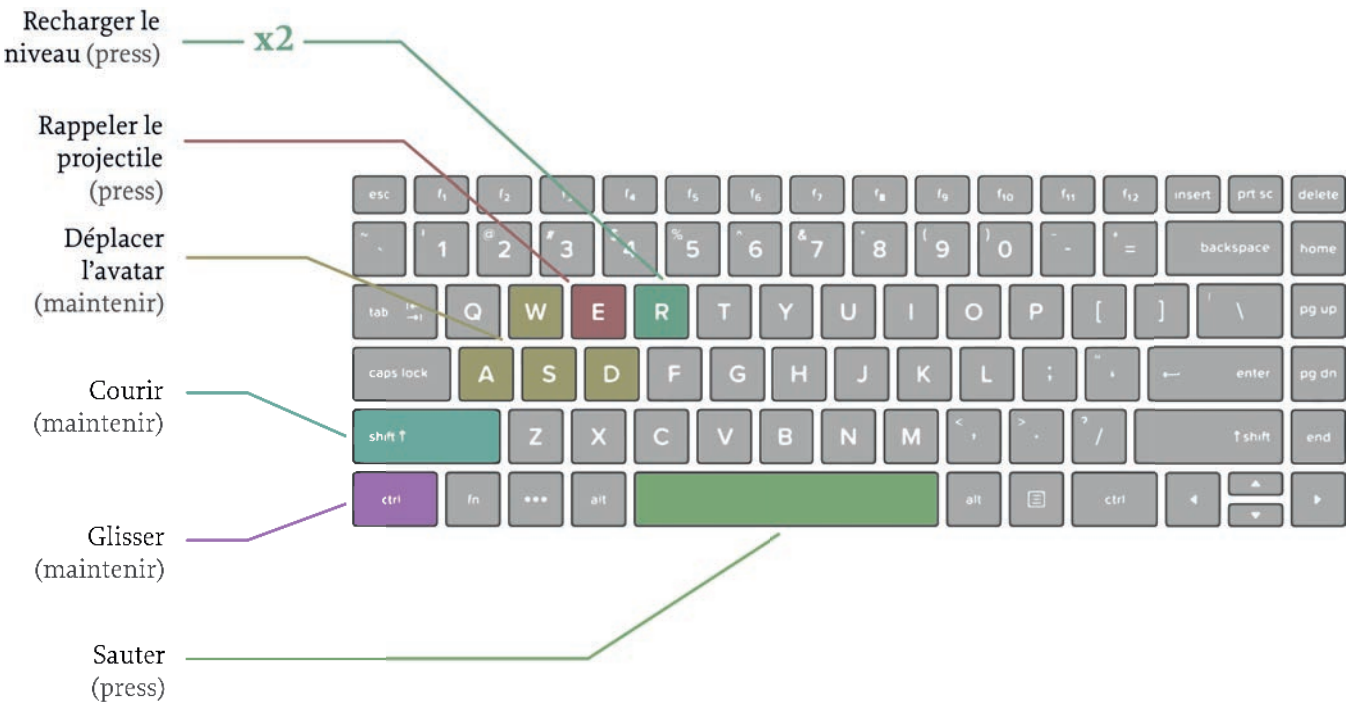
Comme nous ciblons des Hardcores, il ne faut pas essayer de réinventer les habitudes.



Clavier

Comme pour la souris, nous avons adopté les contrôles conventionnels de FPS.

Pour recharger le niveau, nous avons choisis un appuie double sur le bouton R pour s'assurer que le joueur ne relance pas par erreur.

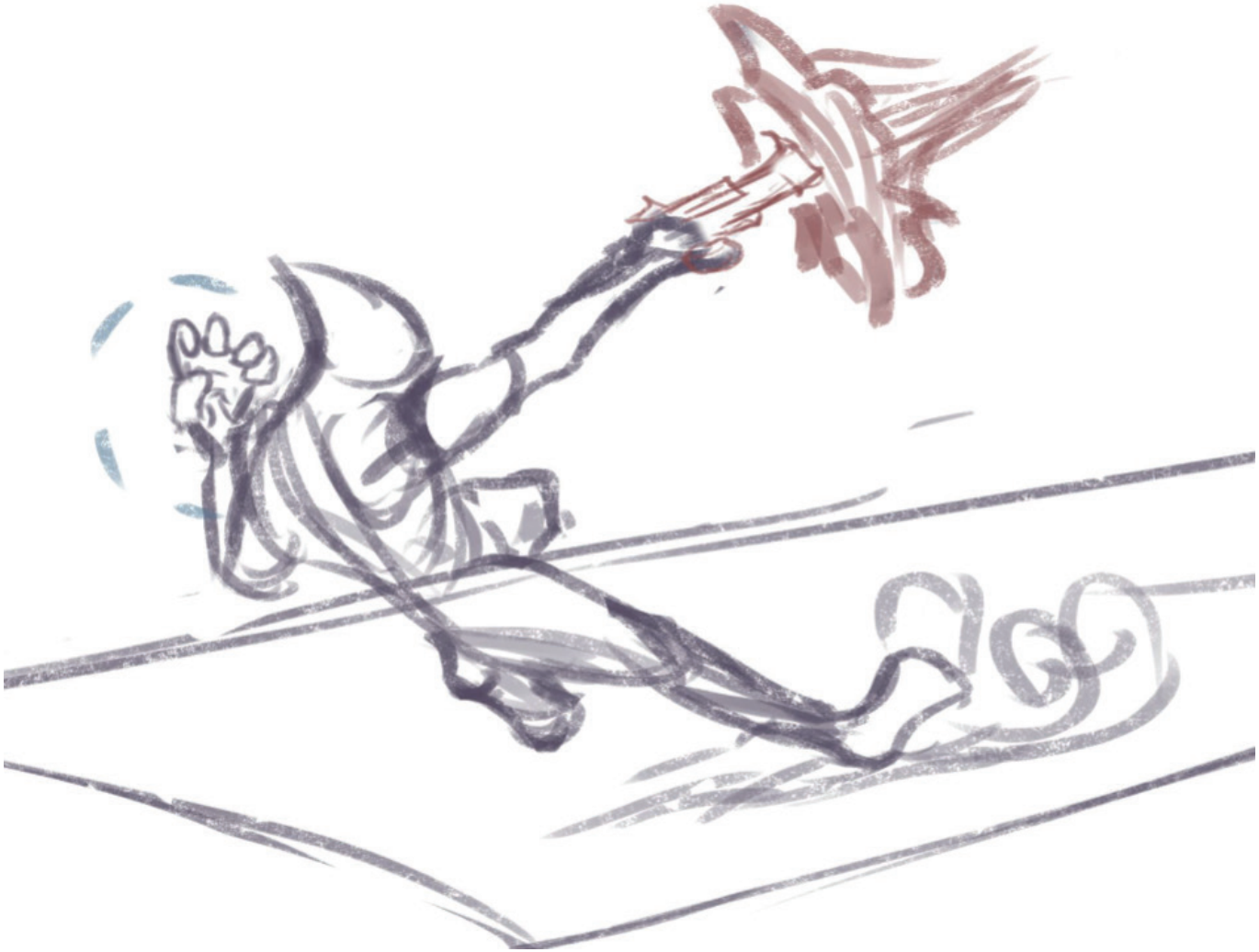


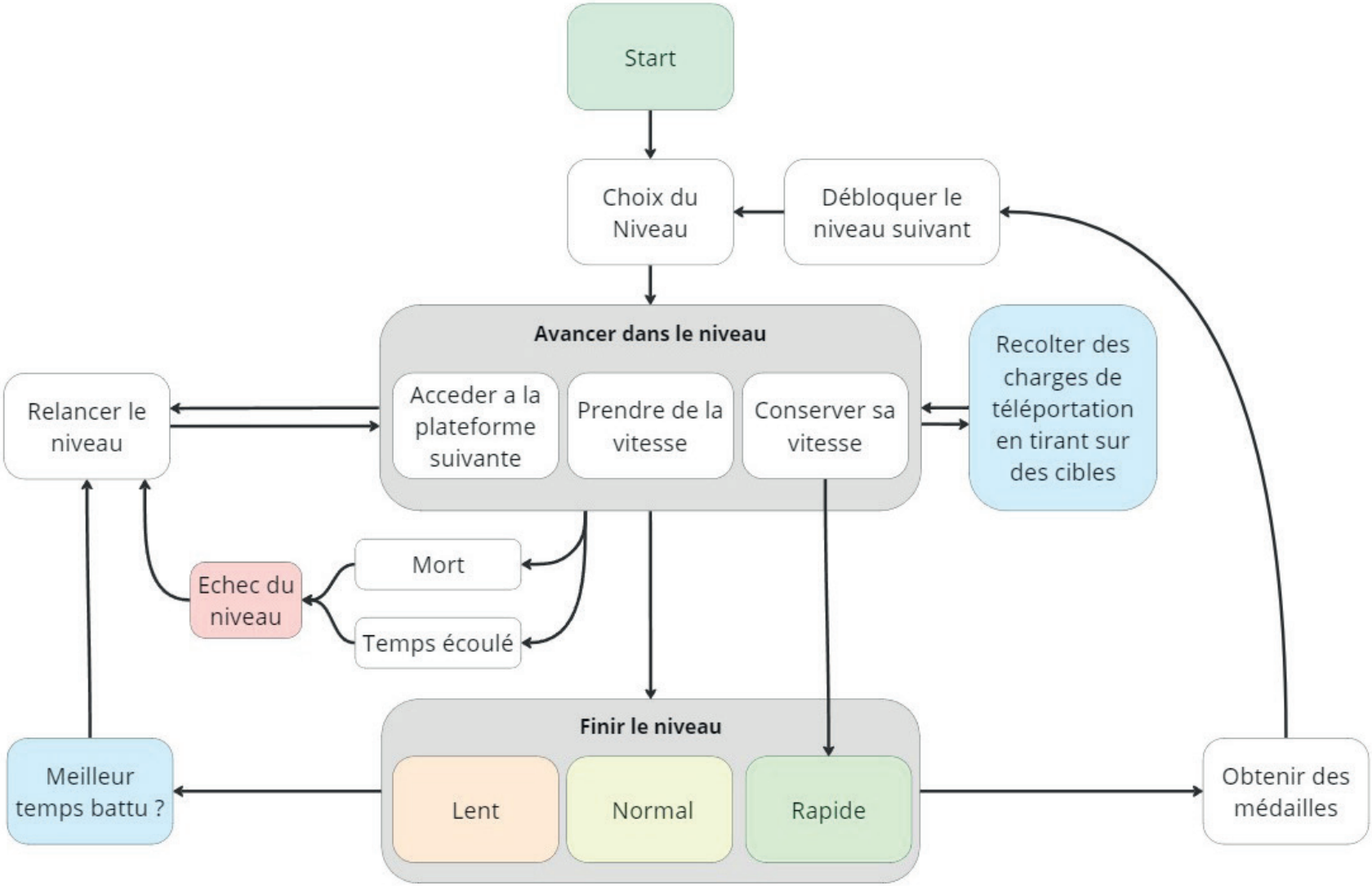
Character

Le joueur contrôle un cyborg humanoïde évoluant dans de grand espace. Il subit la gravité, les collisions et les frottements.
De part sa nature, l'avatar est très rapide et agile.

Il peut se déplacer en avant, arrière, gauche et droite, il peut marcher ou courir, sauter, glisser. Glisser lui permet de prendre de la vitesse en descente et de la conserver sur un sol plat.

L'avatar possède des projectiles qu'il peut lancer dans la direction de sa caméra. Une fois l'un d'entre eux lancé, il peut soit le ramener dans sa main, soit se téléporter dessus. Lors de la téléportation, l'avatar conserve toute sa vélocité
Il n'a qu'un nombre limité de projectiles et peut en récuperer en tirant sur certaines cibles.




Start
Choix du Niveau
Débloquer le niveau suivant
Avancer dans le niveau
Acceder a la plateforme suivante
Prendre de la vitesse
Conserver sa vitesse
Recolter des charges de téléportation en tirant sur des cibles
Relancer le niveau
Echec du niveau
Mort
Temps écoulé
Finir le niveau
Lent
Normal
Rapide
Meilleur temps battu ?
Obtenir des médailles

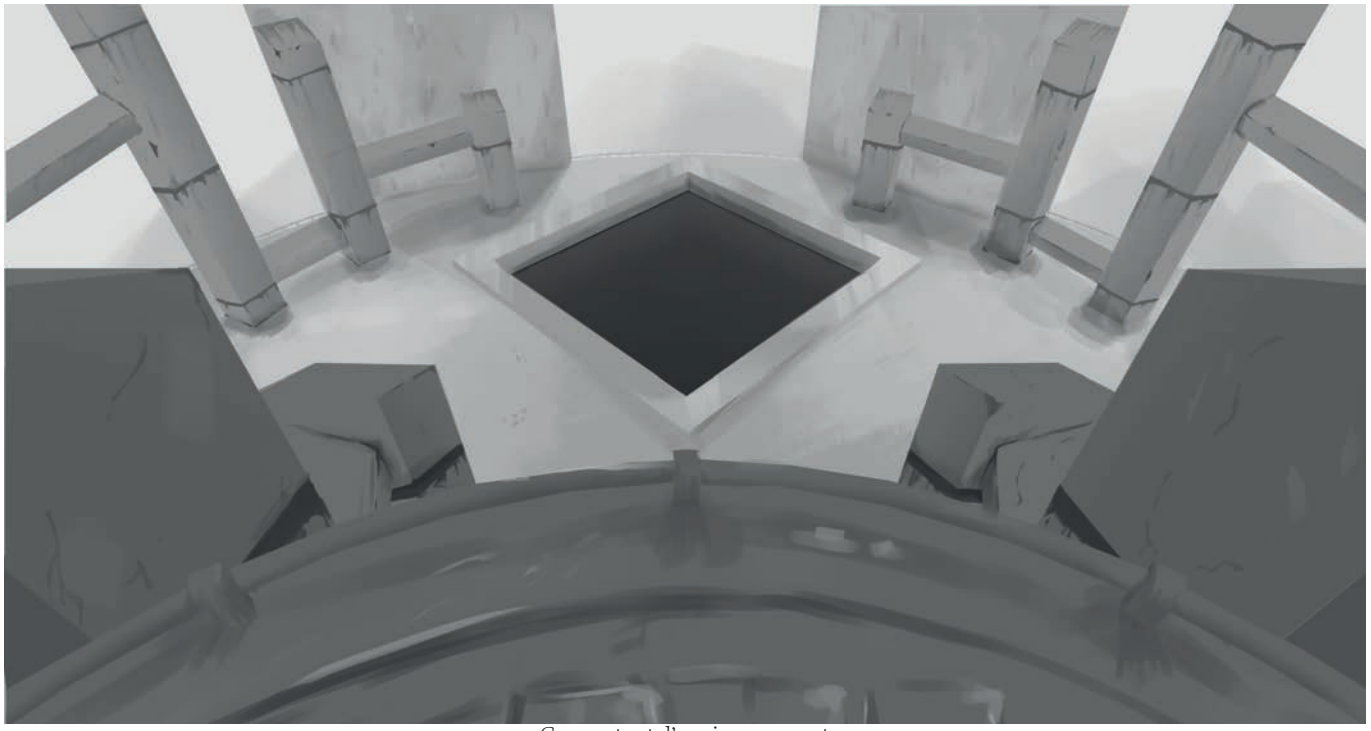
Univers

Le joueur évolue dans un environnement imposant et inquiétant dans le style brutaliste. Cette architecture est corrompue par un étrange organisme.

C'est de cet organisme que l'avatar tire ses pouvoirs. Il représente donc les cibles.

Cet environnement permet un bonne lisibilité qui est nécessaire a cause de la grande vitesse de l'avatar.

Le contraste entre l'environnement et l'organisme doit être très appuyé pour rapidement identifier les cibles.



Concept art d'environnement

Concept art de l'oganisme

Tableau d'incrémentation

FEATURES	KERNEL		INCREMENT 1		INCREMENT 2	
Mouvements de l'avatar	Mouvements fluides, toutes les features classiques implémentées (walking, running, jumping, sliding)		Vitesse au sol conservable et ground control amélioré, Grimpe sur le bord de plateform, Plus de feedback et de polish			
	Classic	Medium	SotA	High		
Projectile	Interractions simple avec l'environnement (rebond surface plate,)		Interactions plus complèxes avec l'environnement (cylindre, sphere, ...) et avec l'avatar			
	Classic	**Low**	**KSP**	**High**		
Tir	Tir classique		Meilleurs feedback		VFX travaillé	
	Classic	**Low**	SotA	Medium	KSP	High
Level Design	Peu de niveau avec quelques ingrédients interactibles		Peu de niveaux mais niveaux plus immersifs avec plus d'ingrédients interactibles		Beaucoup de niveaux plus immersifs avec plus d'ingrédients interactibles. Lobby Playground	
	Classic	Medium	SotA	High	SotA	High
Direction Artistique	Univers défini, au moins un concept art d'environnement, d'ambiance, de personnage et de mécanique		Concepts arts spécifiques, plusieurs par domaines.			
	Classic	Medium	KSP	High		
Environnemental Art	Level Art basique, Au moins une texture par matériau		Textures plus détaillé, meilleur travaille sur les lumières		Texture procédurale, level art détaillé.	
	Classic	Medium	SotA	High	KSP	High
Character Art	Modèlisation bras et jambes, animations simples		Juiciness		Animations détaillées.	
	Classic	Medium	SotA	High	KSP	High
Shaders & VFX	VFX essentiels, shader simple		Polish des effets. Compositing simple		Compositing Avancé	
	Classic	Medium	KSP	High	KSP	High
UI	UI basique, Menu		UI Intradiégétique, reglages des touches			
	Classic	**Low**	SotA	Medium		
Sound Design	Sound design basique et fonctionnel, musique non réactive		Sons plus immersifs et plus variés, musique réactive		Travail détaillé sur chaques interactions et multitude de sons. Plusieurs musiques et ambiances sonores	
	Classic	**Low**	SotA	Medium	SotA	Medium
Programmation	- sauvegarde de la progression		- optimisation - parametrer ses propres commandes		Leader Board ? - commande manette - encodage des sauvegardes	
	Classic	**Low**	SotA	Medium	SotA	Medium

GAME DESIGN

Expérience

Le but est de proposer une expérience frénétique, rapide, satisfaisante et évolutive. Le joueur doit appréhender, comprendre et maîtriser les mécaniques pour pouvoir profiter au mieux du jeu mis à sa disposition.

L'idée est que le joueur se rende rapidement compte de ce que les outils mis à sa disposition lui permettent de faire et qu'il les exploitent pour finir au plus vite les niveaux dans lesquels il évolue.

Il y a peu de mécaniques mais ces dernières permettent une grande liberté de mouvement aux joueurs. Chaque joueur peut donc exploiter les mécaniques comme bon lui semble pour terminer chaque niveau.

Enchaînement des actions

Pour que l'expérience souhaité puisse être atteinte, il est important de mettre au centre du gameplay l'enchaînement entre les actions du joueur. Si les enchaînements sont fluides et rapides, alors l'expérience le sera aussi.

Lien entre les mécaniques

Pour que le gameplay soit fluide, il faut que les mécaniques soient cohérente et intuitives. Il est donc important que le joueur comprenne rapidement comment marche chacune d'entre elle et leurs interactions.

Maitrise du Gameplay

L'enchaînement des actions et le coté dynamique de notre jeu viennent rapidement demander une compréhension et une maitrise des outils. Le but n'est donc pas de donner des outils trop complexe au joueur.

Le choix de s'orienter vers un jeu de speedrun axe le gameplay autour de ses mécaniques et de leur bonne utilisation.

Si une mécanique est mal utilisé, le joueur sera grandement impacté.

Gameplay

Le but est de créer une expérience rapide et dynamique dans laquelle toutes les actions s'enchaînent de façon fluide et cohérente.

Le joueur doit instinctivement comprendre ce qu'il doit faire et comment il doit le faire.

Pour ne pas aller a l'encontre de la fluidité, toutes les actions doivent s'enchaîner de façon fluide et il ne doit pas y avoir de temps mort entre chacune de ces actions.

Direction Artistique

La direction artistique doit être au service de la lisibilité et de la compréhension des mécaniques avant de servir un aspect esthétique.

Processus de Design

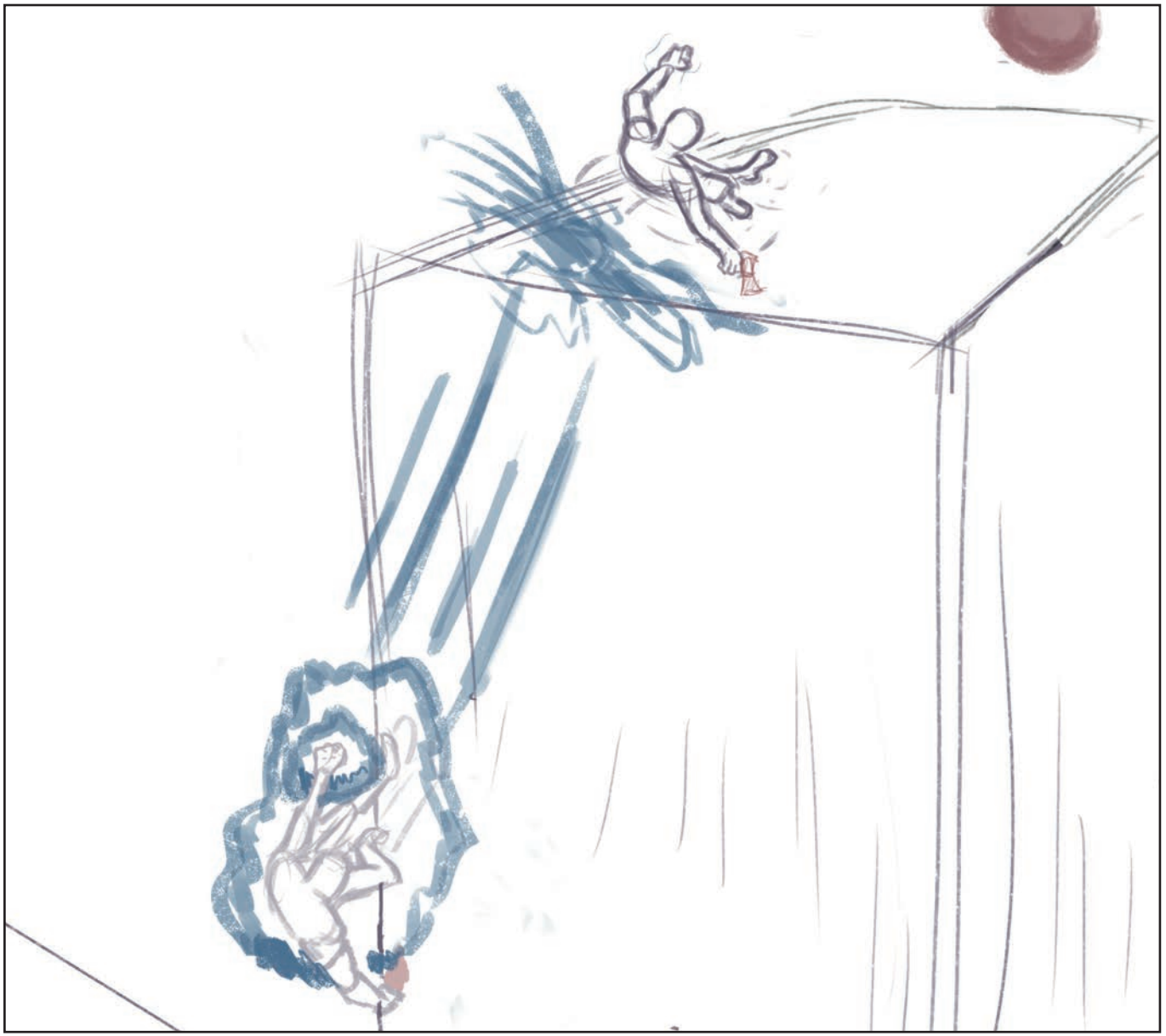
Le sujet étant très large, nous avons fait choisi de trouver des concepts pour un seul genre de jeu. Cela à facilité et cadré la création de mécanique.

L'état critique nous à rapidement orienté vers le genre du FPS. Beaucoup d'états critiques sont trouvables dans ce genre de jeux. Par exemple, un manque de ressources, un manque d'endurance ou de point de vie.

Une fois le choix du genre de jeu choisi, nous avons décidé d'inclure une notion de manque de ressource.

Notre choix s'est porté sur un outils de téléportation que le joueur peut lancer et sur lequel il peut se téléporter. Il n'en possède qu'un nombre limité et doit donc viser des cibles pour en récupérer.

Cette mécanique assez libre nous a permis de nous poser des question quant au genre du jeu. Comme tout le gameplay est axé autour de la visé, nous avons décidé de conserver le genre du FPS.



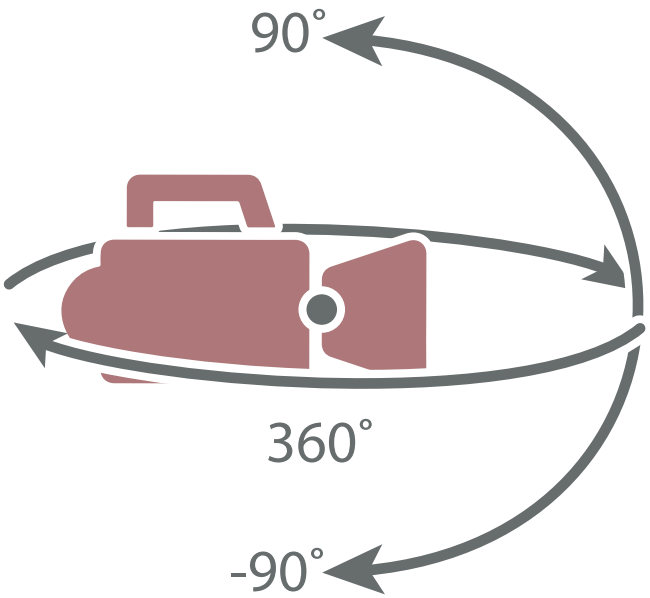
Choix de la caméra

Les raisons du choix de la caméra a la première personne sont les suivantes :

- Une simplification de la visée.
- Une bonne réactivité. La caméra à la première personne permet de suivre exactement les mouvements que le joueur effectue avec sa souris. Il n'y a pas de latence. La caméra ne sert pas un aspect cinématique mais elle sert le gameplay.
- Une meilleure immersion. La caméra est le point de vue de l'avatar contrôlé. On voit au travers de ses yeux. Si l'avatar a des informations, le joueur les a aussi.
- Un sentiments de vitesse plus appuyé. Grace à la meilleure immersion, il est plus simple de faire ressentir la vitesse. Le FOV, la distorsion et d'autres effets sont plus exploitables en FPS qu'en TPS.

Caméra - Générale

A l'arrêt et en déplacement, la caméra peut se tourner à 360° horizontalement et est bloqué entre 90° et -90° verticalement.



Le jeu est axé autour de la visé, il faut donc que la caméra ne soit pas limité pour ne pas limiter les actions du joueur.

La caméra est basé sur une caméra classique de FPS. Notre cible est censé être déjà habitué ou sensibilisé à ce controller.

Caméra - En mouvement

Comme dit précédemment, en FPS, le joueur à besoin de référentiel en plus que ce du terrain sur lequel il se trouve. C'est pour cela qu'en mouvement, la caméra doit indiquer au joueur la direction dans laquelle il se dirige.

Pour cela, la camera s'incline vers la gauche ou vers la droite en fonction du mouvement de l'avatar. L'inclinaison avant et arrière n'est pas nécessaire et dessert la lisibilité globale du jeu.

En plus d'ajouter à la lisibilité des mouvements, cet effet vient améliorer l'immersion et faire comprendre que l'avatar est un biped d'une certaine taille.

De même, pour l'avatar qui avance, la caméra doit très légèrement monter et descendre. Cet effet doit être faible pour ne pas nuire a la lisibilité du jeu. Cela permettra aussi de renforcer le fait de comprendre que l'avatar est biped et qu'il marche.

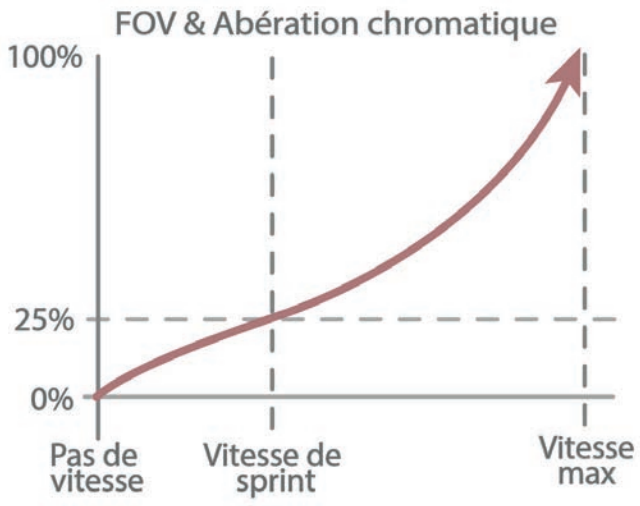
Caméra - Vitesse

Dû à la haute vitesse que l'avatar peut atteindre, les référentiels visuels (textures et repères visuels) ne sont pas suffisant pour comprendre la vitesse de déplacement.

Il faut donc ajouter des effets visuels pour indiquer artificiellement au joueur sa vitesse.

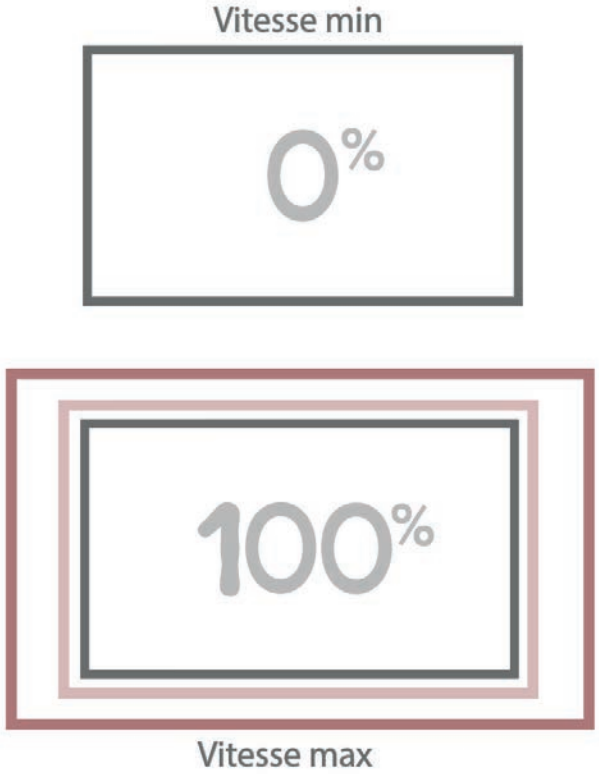
L'augmentation de FOV permet d'altérer la perception du ressenti de la vitesse.

L'abération chromatique vient ajouter une distorsion de la vision appuyant l'effet de vitesse.



Le changement ne doit pas être trop brutal et trop grand. Si l'avatar se déplace déjà rapidement, il ne faut pas un trop grand FOV, les deux se cumule et le tout devient illisible.

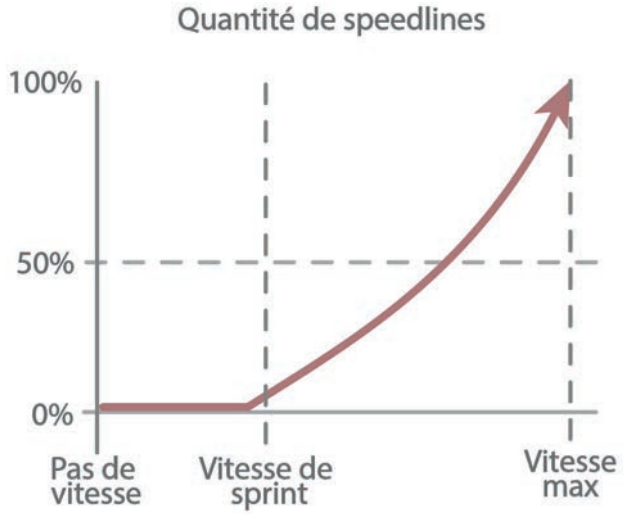
De même pour les changement de vitesse brutaux, la transition entre la marche est la course n'est pas trop grande pour ne pas perturber la lisibilité.



Les speedlines sont un moyen visuel quantifiable par le joueur pour comprendre la vitesse à laquelle l'avatar va. Elles ne sont pas obligatoirement de la forme classique (des lignes).

Pour compenser le manque visuel de jambes, les speedlines doivent être directionelles, c'est à dire suivre la direction de l'avatar.

En plus d'indiquer une vitesse, les speedlines indiquent donc la direction de l'avatar ajoutant encore un référentiel améliorant la lisibilité du jeu.

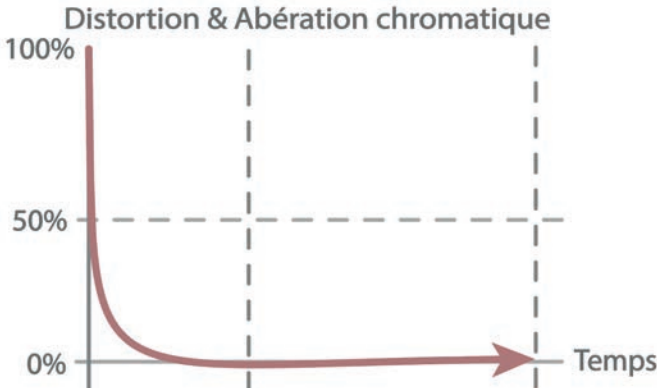


Caméra - Téléportation

La problématique est que si il n'est pas assez conscient de son environnement, il ne va pas forcement se rendre compte de son changement de position.

Il faut donc un feedback montrant ce déplacement. Lors de la téléportation, le joueur doit avoir un feedback montrant qu'il s'est téléporté. Cet effet est caractérisé par une distorsion de la vision du joueur. L'utilisation de distortion d'écran, de bloom, d'abération chromatique donne cet effet recherché.

Ces effet ne doivent pas rester longtemps à l'écran, il doivent être comme un flash (ci dessous la courbe d'intensité des effets).



Caméra - Informations à l'écran

L'UI doit donner des informations essentiels au gameplay. Le joueur doit donc savoir les choses suivantes :

- La vitesse à laquelle l'avatar va (cela passe par les effets visuels et par un compteur).
- L'information si le projectile est lancé ou en main (cela passe par le visuel du projectile en main ou non).
- Le nombre de charge qu'il lui reste (cela peut passer par une UI intra ou extra diégétique).
- Le temps global du niveau.

- La hauteur à laquelle il se trouve par rapport au sol en dessous de lui.
- Le centre de l'écran : vers où l'avatar va tirer et lancer son projectile.
- Le temps avant que le projectile revienne dans la main de l'avatar.
- Toutes les informations liés au téléporteur (en l'air, en main, vient d'être détruit) .
- La position du téléporteur lorsqu'il est lancé
- Le nombre de cible à détruire pour terminer le niveau. (si niveau à nettoyer)
- La position des cibles à l'écran pour améliorer la lisibilité global.

Contrôles

Clavier

De par la nature du jeu et du support (PC), le jeu propose un menu pour reconfigurer les touches et modifier la sensibilité de la souris.

Chaque joueur à des contrôles différents et il faut que le jeu soit jouable par le plus grand nombre possible.

Orienter la camera

Déplacer la souris à droite et à gauche pour tourner horizontalement et avant arrière pour regarder vers le haut et le bas.

Orienter l'avatar

Appuyer sur A & D pour aller a droite et a gauche en fonction de où regarde la caméra.
Appuyer sur W & S pour avancer et reculer en fonction de la caméra.

Tirer

Appuyer sur le clic gauche de la souris permet de tirer qu'il y ait une cible ou non.

Courir

Appuyer sur SHIFT en avançant ou en allant à droite ou à gauche permet de courir.

Glisser

Appuyer sur CTRL en allant au dessus d'une certaine vitesse permet de glisser dans la direction de la vélocité de l'avatar.

Sauter

Appuyer sur ESPACE permet de sauter.

Interaction avec le projectile

Le projectile se lance en appuyant sur le clique droit de la souris, dans la direction de la caméra, vers le centre de l'écran.

Appuyer sur le clique droit de la souris lorsque le projectile est lancé permet de se téléporter à la position du projectile.
Appuyer sur E permet de le rappeler sans consommer de charge.

Relancer le niveau

Appuyer deux fois sur la touche R rapidement permet de relancer le niveau depuis le début.

Support Manettes

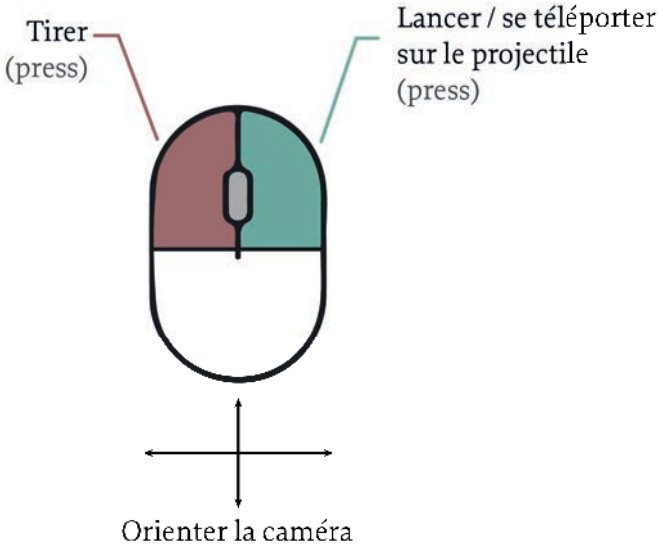
Le jeu n'est pas designé pour être joué à la manette, il n'y a donc pas de support optimisé prévu pour ce périphérique.

Contrôles

Souris

Pour les contrôles souris, nous avons adopté les conventions du genre. Clique droit pour le tir et clique gauche pour la capacité.

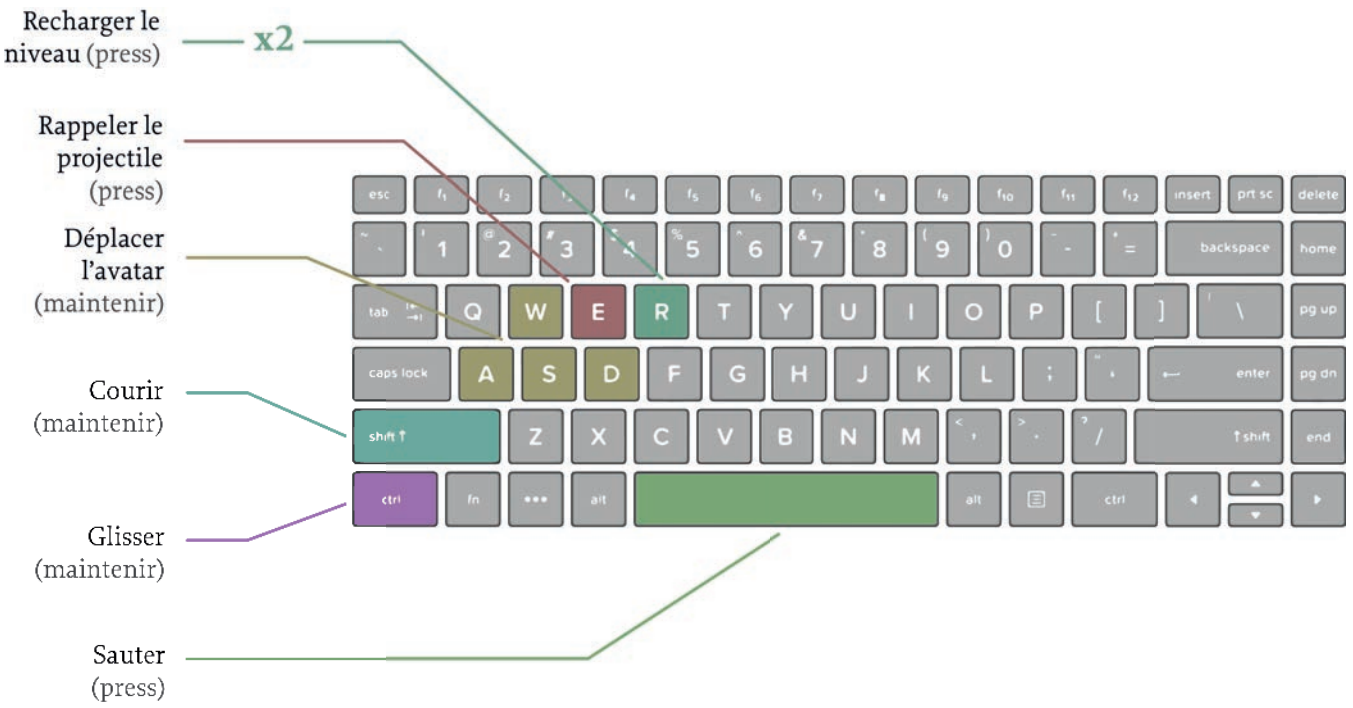
Comme nous ciblons des Hardcores, il ne faut pas essayer de réinventer les habitudes.



Clavier

Comme pour la souris, nous avons adopté les contrôles conventionnels de FPS.

Pour recharger le niveau, nous avons choisis un appuie double sur le bouton R pour s'assurer que le joueur ne relance pas par erreur.

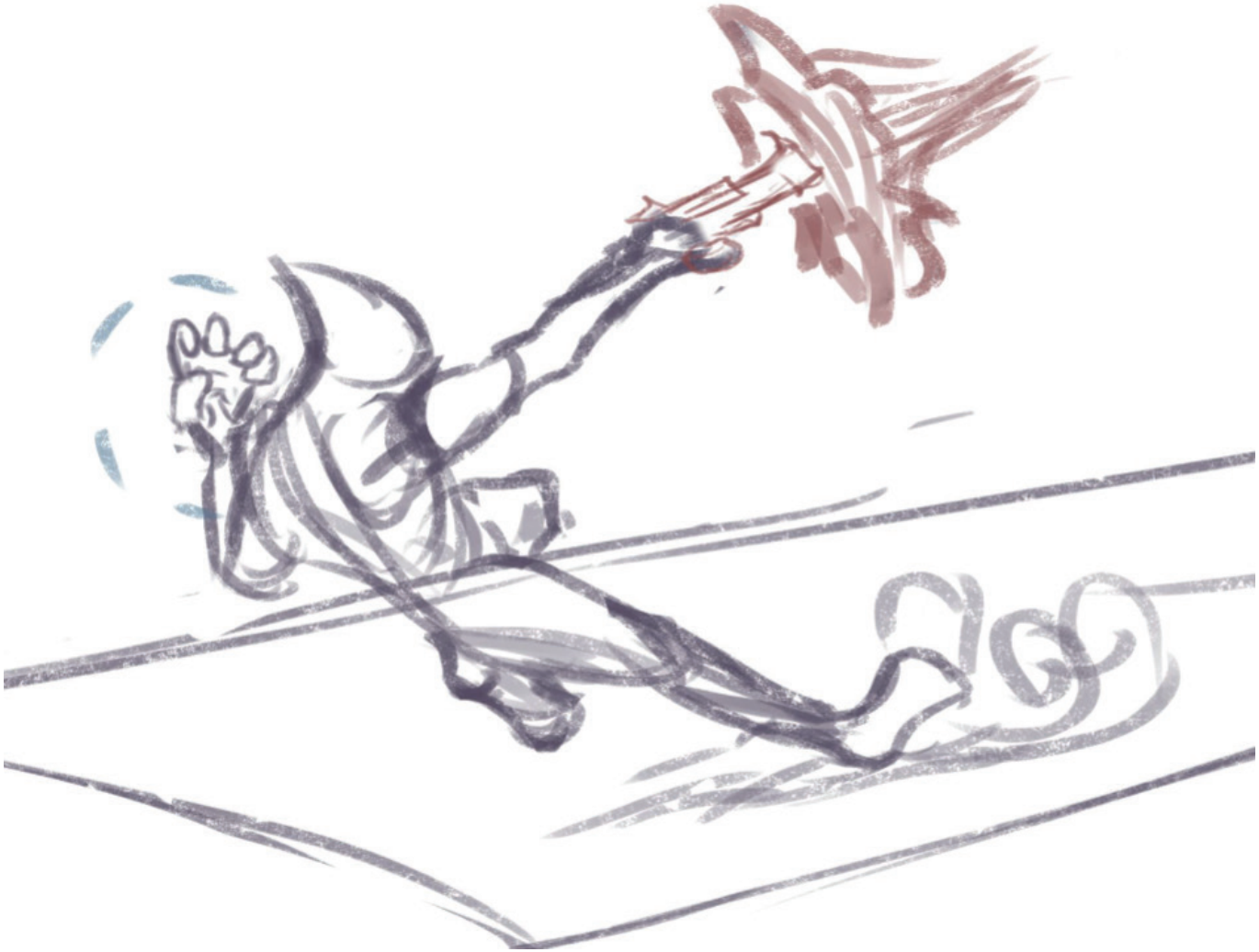


Character

Le joueur contrôle un cyborg évoluant dans de grand espace. Il subit la gravité, les collisions et les frottements.
De part sa nature, l'avatar est très rapide et agile.

Il peut se déplacer en avant, arrière, gauche et droite, il peut marcher ou courir, sauter, glisser. Glisser lui permet de prendre de la vitesse en descente et de la conserver sur un sol plat.

L'avatar possède des projectiles qu'il peut lancer dans la direction de sa caméra. Une fois l'un d'entre eux lancé, il peut soit le ramener dans sa main, soit se téléporter dessus. Lors de la téléportation, l'avatar conserve toute sa vélocité
Il n'a qu'un nombre limité de projectiles et peut en récupérer en tirant sur certaines cibles.



Character

L'avatar subit la gravité et a une réalité physique (il subit les collision). Il fait 3 unités de haut, 1 unité de large.
Il a à sa disposition les actions suivantes :

Marcher :

Il marche à la vitesse de 8 unités par seconde.

Courir :

Il court à la vitesse de 20 unités par seconde. Si il touche le sol à une vitesse supérieur à 20u/s et qu'il court, sa vitesse descend rapidement à 20u/s.

Sauter :

Il saute de 7.5 unités (2.5x sa taille).

Glisser :

En glissant, il perd sa vitesse plus lentement et ne peut se diriger que très légèrement. La vitesse de glissade n'est pas limitée.

Lorsqu'il glisse il fait la 0.5 fois sa taille (1.5 unité).

S'accroupir :

L'avatar s'accroupit si il essaie de glisser mais que sa vitesse est inférieur à 15u/s.

Accroupi, il fait 0.75 fois sa taille (environ 2 unités).

Lancer un téléporteur et se téléporter :

Il peut lancer un projectile qui subit la gravité et se téléporter dessus. L'avatar conserve la vélocité du projectile.

Tirer :

Il peut tirer sur des cibles grâce à une arme. Cette action lui redonne un projectile cité précédemment.

Références Mouvements

Pour le character controller de notre jeu, nous nous sommes inspirés de différent FPS en essayant de combiner les parties intéressantes pour nos intentions.

Pour le controller en lui même, nous nous sommes inspiré du comportement de Ultrakill. Ultrakill est un Fast FPS très rapide et nerveux. La rapidité et réactivité des mouvements et le dynamisme du saut sont les points retenu de ce jeu.

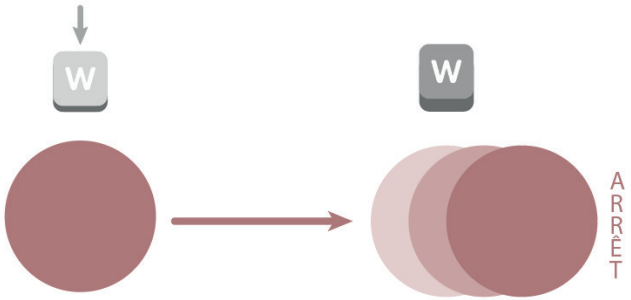
Notre glissade est inspiré de Dusk et Titanfall 2. La conservation de la vitesse, le comportement de la camera et le gamefeel global de la glissade de Titanfall 2. Le controller de Warstride Challenge nous a aussi inspiré pour cette glissade.

Pour le polish et le gamefeel globale du controller, nous nous sommes inspiré de Shady Knight, un platefromer 3D aux inspiratinons de Fast FPS.

Déplacements

La marche est une marche classique de FPS. Elle n'est pas très complexe car peu utilisé. Elle permet au joueur qui en ont besoin d'être précis et de se déplacer lentement.

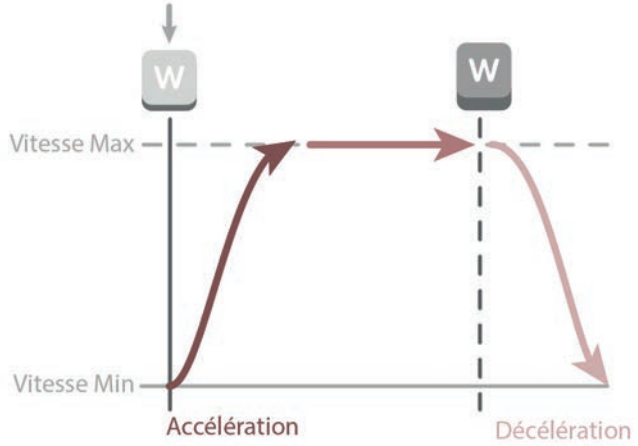
La course est réactive et doit avoir un semblant de réalisme. L'avatar à donc une inertie car il a un poids mais pour ne pas desservir la maniabilité, cette inertie est faible.



De plus la glissade permet de garder sa vélocité dans la direction de l'avatar et rendre l'avatar glissant de base viendrait à rapprocher les deux états de course et de glissade.

La course permet de prendre des angles secs rapidement grâce à sa réactivité et sa faible inertie mais sa vitesse est limité à 20u/s.

La courbe de prise et perte de vitesse étant la suivante, pour ne pas perdre la vitesse accumulé, le joueur va devoir glisser.



Le but est de lui imposer un choix entre conservation de vitesse et direction de l'avatar.

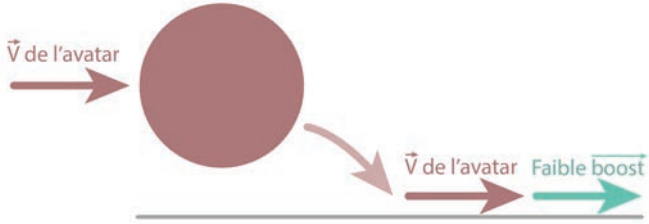
Cet obligation va le forcer à alterner rapidement entre les modes de déplacement et va demander une vraie maitrise du déplacement pour terminer rapidement les niveaux tout en créant du dynamisme dans le gameplay.

Glissade

Pour conserver sa vitesse, l'avatar peut glisser. Cette glissade n'est pas directionnelle, c'est à dire qu'elle ne réoriente pas l'avatar dans la direction de son regard. Elle conserve simplement la vélocité de l'avatar.

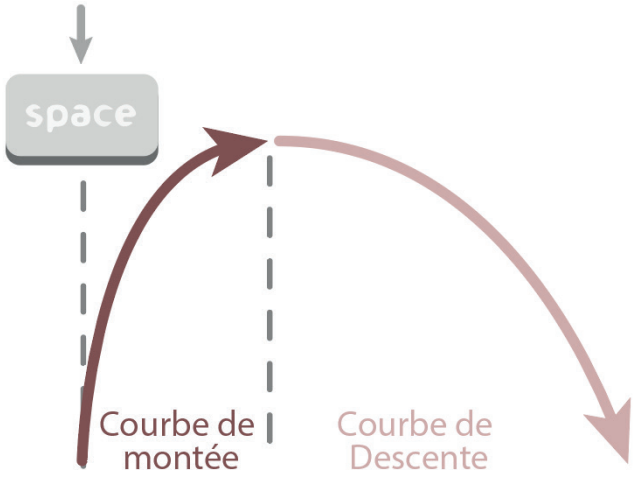
Dans cet état, la caméra ne peut plus tourner à 360° et le joueur ne peut que très légèrement diriger l'avatar à droite et à gauche.

La glissade vient ajouter une faible force dans la direction de l'avatar, cela permet de contrebalancer la perte de vitesse dû au frotement de l'air en plus de donner un impacte physique à la mécanique.



Saut

Le saut est classique. Il n'est pas dynamique, c'est à dire qu'il n'est pas sensible à la durée d'appuie.



Il est possible de sauter peu importe l'état actuelle de l'avatar. Il peut donc sauter pendant une glissade, la course et la marche.

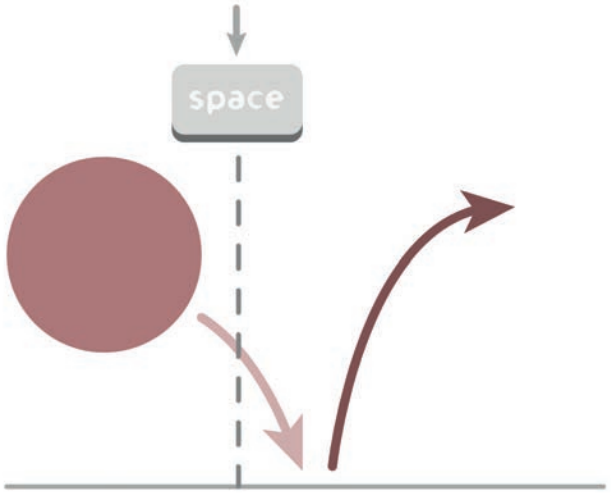
Le saut n'annule pas la vélocité de l'avatar, il peut conserver sa vélocité dans toute les directions.

Ces choix permettent de fluidifier l'enchaînement entre les actions.

Game Feel

L'ajout de deux mécaniques simples permet de simplifier l'utilisation et la tolérance du saut.

Le premier, le buffer, permet de ne pas devoir appuyer au moment exacte où l'avatar touche le sol. Si l'appuie est légèrement avant, l'avatar saute quand même en touchant le sol.

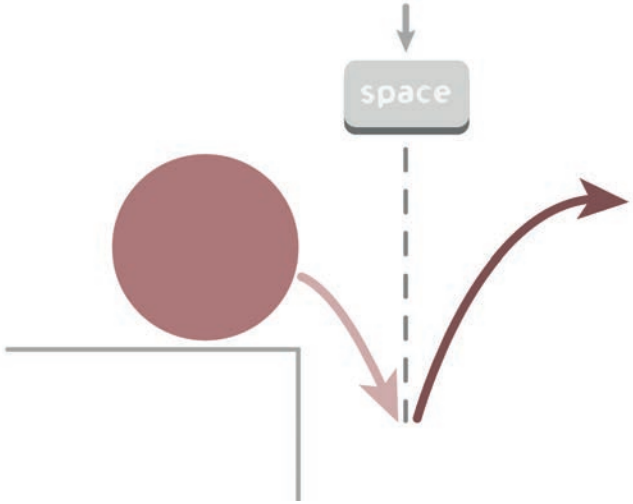


Notre but n'est pas de challenger le timing sur le saut, cet ajout est nécessaire. De plus, il permet de fluidifier l'enchaînement des actions ce qui est un de nos objectifs.

Le deuxième ajout est le coyote time.

Cette mécanique permet d'ajouter une plus grande tolérenance en plus de donner une légère profondeur au saut.

Le fonctionnement est le suivant : Si l'avatar quitte une plateforme sans sauter, le joueur à un très court laps de temps pour faire sauter l'avatar.

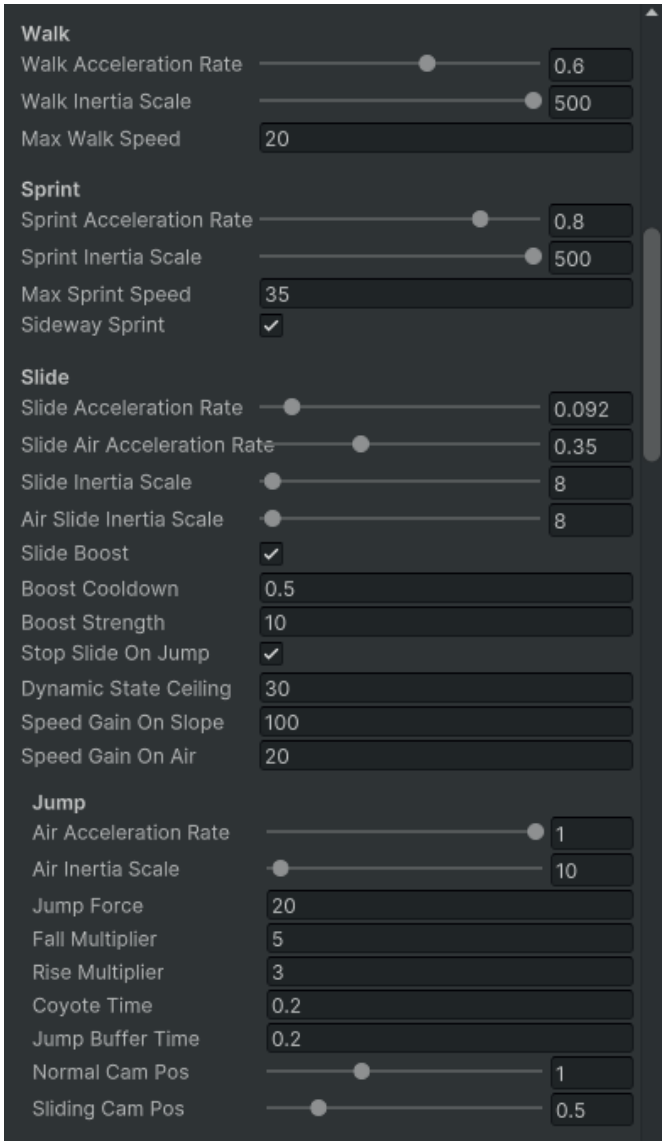


Cet ajout permet au Hardcore d'optimiser les sauts et permet au Midcore d'avoir une plus grande marge d'erreur.

Pour avoir le plus de contrôle possible sur tous les metrics et le game feel du controller, nous avons fait en sorte que la plupart des paramètres soient réglables dans l'inspecteur de Unity.

Ainsi, la nature glissante de l'avatar est réglables ainsi que sa vitesse de marche ou de course. La liste ci contre est exhaustive.

Cela permet de pouvoir équilibrer chacune des actions du controller efficacement et rapidement.



Références téléporteur

L'idée de lancer un objet sur lequel se téléporter est notre base sur laquelle nous avons ajouté des mécaniques. Nous avons réalisé rapidement que cette mécaniques était le centre de gameplay du personnage Sombra de Overwatch. Nous avons donc regardé comment la mécanique principale de lancé et ses feedbacks permettait de ne pas être perdu et de bien se reperer dans l'espace.

Pour revenir sur les feedbacks, ils donnent beaucoup d'indications de direction et de position.

C'est donc ce que nous avons essayé de reproduire pour que notre joueur comprenne vers où il lance son téléporteur et où il va attérir.

Feedbacks Sombra - Overwatch

Un autre personnage a une mécanique de téléportation semblable à la notre. Loba dans Apex Legend. Elle aussi peut lancer un projectile sur lequel se téléporter.

Comme pour Sombra, nos intentions sont différentes mais la mécanique est semblable. Nous avons donc regardé comment les developpeurs indiquait la trajectoire et le point d'arrivé lors de la téléportation.

Comme le téléporteur de loba a un comportement très spéciale, nous n'avons vraiment regardé que les feedbacks de la mécanique.

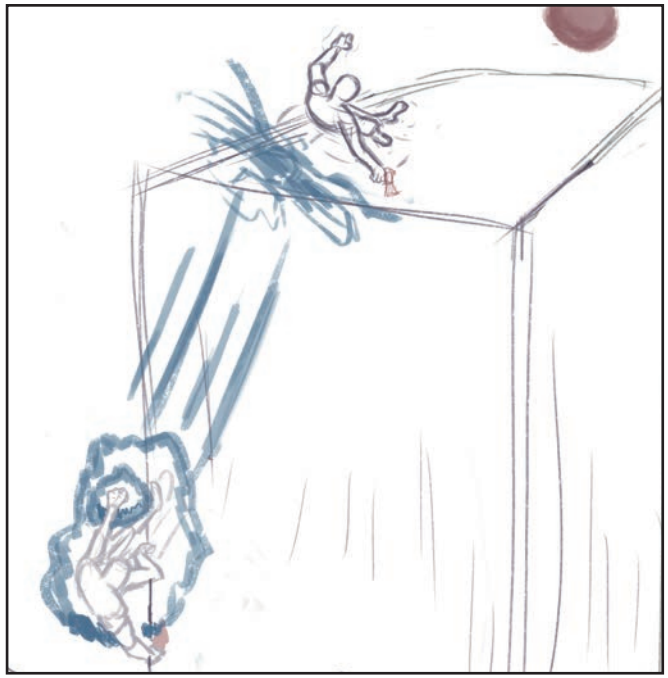
Prévisualisation trajectoire Loba - Apex Legends

Téléporteur

Le téléporteur est la mécanique principale de notre jeu (avec le déplacement). Il se matérialise par un objet physique subissant la gravité et les collisions.

Il rebondit sur les surfaces en béton mais est détruit sur les parasites.

Il est lancé à une certaine vélocité depuis la main de l'avatar vers le centre de l'écran.



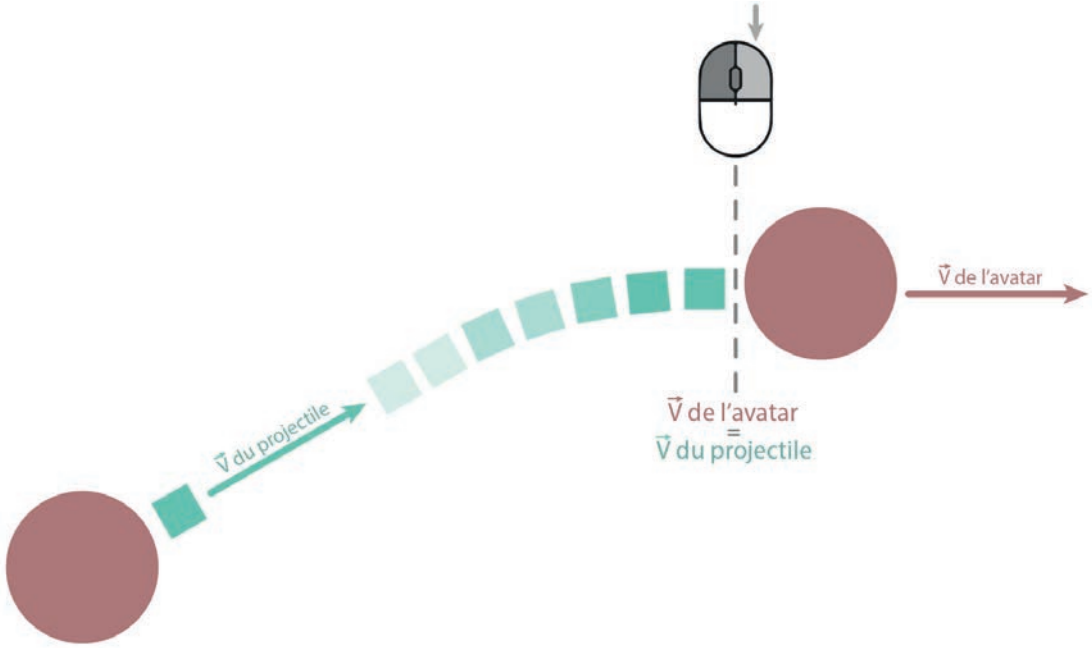
Cette ressource est limité. L'avatar ne peut posséder que deux charges au maximum.

Si l'avatar a au moins une charge et qu'il lance le téléporteur, deux choix s'offrent au joueur :

- Rappeler le téléporteur dans la main de l'avatar. Cette action ne consomme pas de charge.
- Se téléporter à la position du téléporteur. Cette action consomme une charge.

Au moment de la téléportation, l'avatar prend la position du téléporteur et ce dernier disparaît. De plus, l'avatar se voit appliqué la vélocité du téléporteur au moment de la téléportation.

Ce choix permet de pouvoir anticiper la trajectoire de l'avatar après téléportation tout en ne ralentissant pas le rythme globale du gameplay.



Téléporteur

Comme dit précédemment, le téléporteur est limité en nombre. Le joueur ne peut en avoir que deux au maximum.

Ce choix permet une liberté de mouvement et un enchaînement personnalisé. Cela ajoute plus de situation de jeu. Il est possible de récupérer des charges quand un téléporteur est déjà lancé, ou d'en récuperer tout court.

Cela vient créer une gestion des ressources que le joueur a en sa possession.

Notre jeu pourrait se s'exprimer comme une ligne droite dans laquelle il faut exécuter les actions comme le level designer l'a décidé. En limitant à deux et non à un téléporteur, il est possible que le joueur crée son propre chemin.

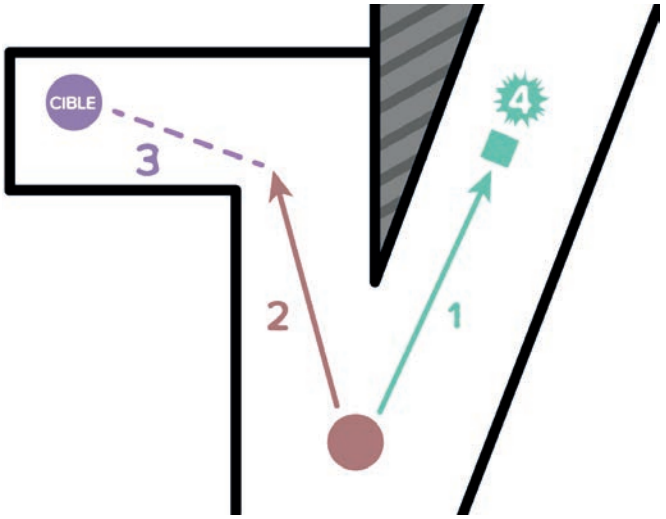
La limitation permet aussi une diversification du gameplay. Ne pas avoir de téléporteur force à utiliser les autres mécaniques du jeu.

L'intérêt vient s'amplifier grâce à notre cible de joueur. Les complétionnistes / compétiteurs ont tendances à rejouer les niveaux et les optimiser.

Comme ils connaissent déjà les niveaux, cette mécaniques permet d'optimiser les trajectoires en fonction des connaissances déjà acquises.

Un exemple d'optimisation serait le suivant. Plutôt que de tout faire étape par étape, le joueur :

1. Lance son téléporteur en avance
2. Se dirige dans le cul de sac
3. Tire sur la cible pour récuperer une charge
4. Se téléporte



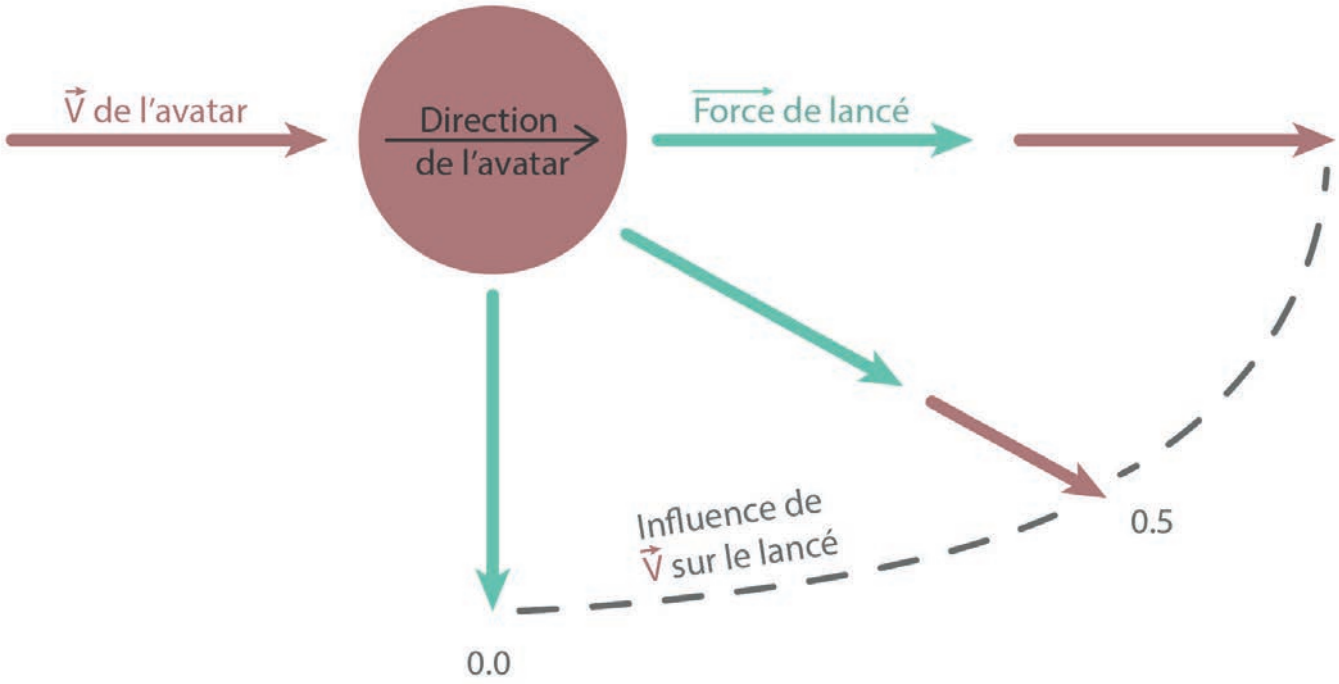
Ce fonctionnement permet une plus grande diversité dans le Level Design et une plus grande profondeur dans le gameplay.

Vitesse du téléporteur

La vitesse de lancé du téléporteur est relative à celle de l'avatar.

L'exemple est le suivant : On prend une balle que l'on lance depuis un voiture en mouvement. Cet objet va aller à une vitesse égal à la vitesse de la voiture additionné à la force de lancé. Cela ne s'applique que si on lance la balle dans la dans la direction où avance la voiture.

Il en est de même pour notre téléporteur. L'avatar joue la voiture et le téléporteur la balle.



L'angle de lancé fait varier l'influence de la vitesse de l'avatar. Si le téléporteur est lancé dans la direction de la vitesse de l'avatar, l'influence est maximale et si l'angle de lancé est de 90°, l'influence est nulle.

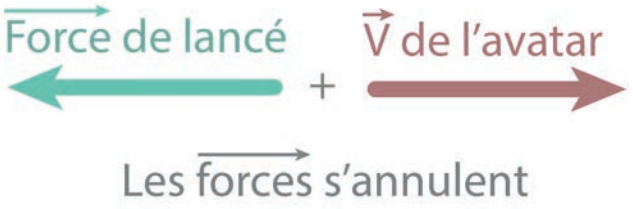
La problématique est qu'il est donc possible de prendre une vitesse infini grâce à ce fonctionnement.

Il faut donc limiter la vitesse de l'avatar. Notre but n'est pas de prendre le plus de vitesse possible mais plutôt de bien exécuter une succession d'action sans faire d'erreur.

Limiter la vitesse de l'avatar permet de limiter automatiquement la force de lancé du téléporteur.

En revanche, dans notre jeu, cette influence n'a d'impacte que vers l'avant.

Dans la réalité, si je lance ma balle dans la direction opposé à la vitesse de ma voiture, si ma force de lancé est égal à la vitesse de la voiture, la balle va rester sur place.



Ce fonctionnement est assez contre intuitif et dessert notre gameplay.

Type de niveaux

Le type de niveau principale est un niveau couloir à terminer le plus rapidement possible. C'est le genre de niveau le plus simple à exploiter pour faire un jeu orienté speedrun.

Le but est d'aller d'un point A à un point B le plus rapidement possible.

Pour appuyer l'état critique, le joueur n'a qu'un temps donner pour aller à la fin du niveau avant que ce dernier soit envahi par le parasite occupant les lieux.

L'autre type de niveau possible consiste à devoir "nettoyer" une zone de tout les parasites présents dans cette dernière.

Nous avons priorisé la production de niveau "couloir".

Pour les couloirs, nos références sont CyberHook, Warstride Challenge, Stop Dead et Ghostrunner.

Ingrédient de Level Design

Pour rythmer les niveaux et créer de la diversité dans le gameplay et les comportement nous avons ajouté trois ingrédients permettant d'avancer dans les niveaux.

- Une zone qui viens tuer l'avatar lorsqu'il collisionne avec. Cela vient créer des zones inaccessibles.
- Une cible sur laquelle tirer pour recharger un téléporteur.
- Une Cible plus grande sur laquelle il faut tirer 4 fois pour finir le niveau. Cela permet de grandement augmenter la frénésie uniquement à la dernière seconde.
- Un un intérupteur permettant d'ouvrir des portes. Cela permet de conditionner l'avancé de l'avatar dans le niveau.

En suivant le modèle de Roger Caillois, Shift semble s'orienter majoritairement vers le Ludus.

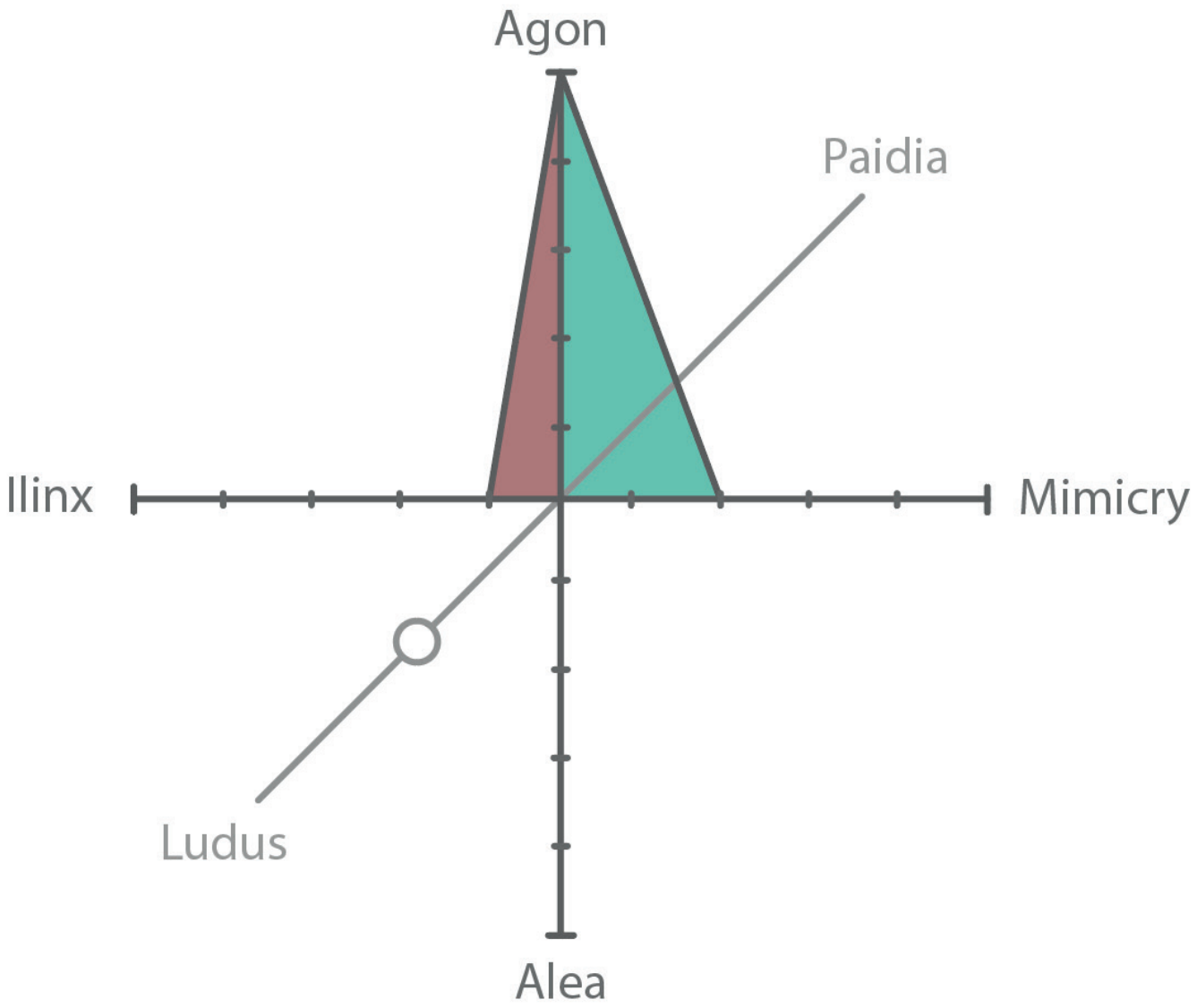
En effet, le jeu est régis par de nombreuses règles et le Level design est assez linéaire. Il y a donc peu de place à la liberté. En revanche, le joueur peut emprunter plusieurs chemin et a un controller et une mécanique de téléportation le rendant assez libre.

Pour ce qui est des autres axes, Shift est clairement axé sur le Agon c'est à dire la compétition.

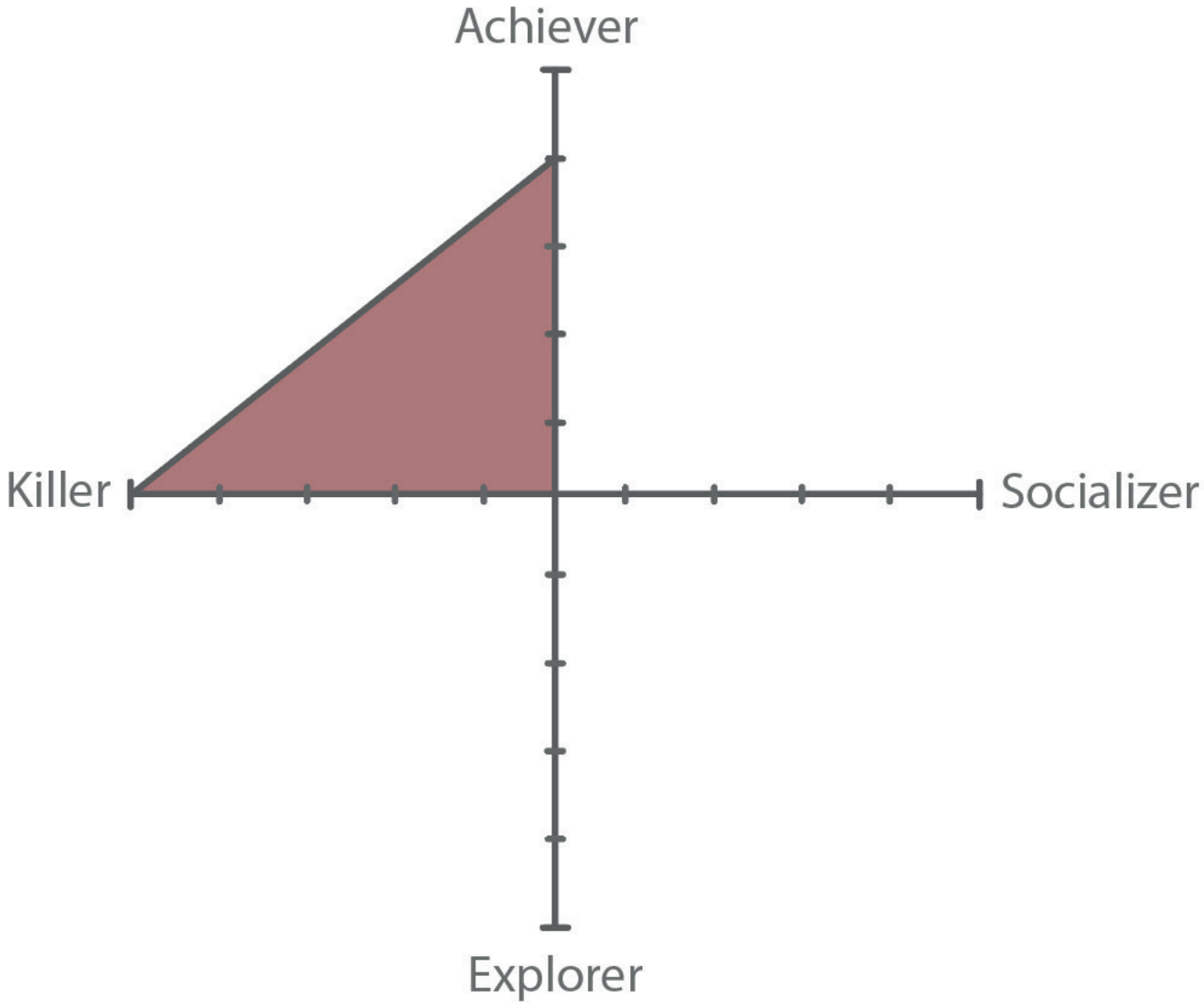
Il y a peu de simulation. Le joueur incarne un avatar mi humain mi robotique le tout a la première personne, cela laisse assez libre court a l'imagination.

Il y a peu d'ilinx et la seule mécanique qui vient déstabiliser le joueur est la téléportation. En revanche, nous avons essayé au plus de réduire cet effet.

Il n'y a pas d'aléatoire, la physique est déterministe et le level design est fixe.



Modèle de Bartle



De par la nature de Shift, la typologie des joueurs touché par notre jeu est assez claire.

Shift est un jeu de speedrun, cela implique directement que les joueurs “Killers” sont visés par le jeu.

Le jeu comporte plusieurs niveaux avec plusieurs degré de réussite en fonction du temps ce qui implique que les “Achievers” aussi sont grandement concerné par Shift.

En revanche, il n’y a pas d’interaction sociale entre des joueurs puisque le jeu est solo et ne permet pas de communiquer.

Enfin, le coté exploration du monde est faible voir absent dû à la limitation en temps et en ressources du jeu.

Challenges principaux

Timing

Le premier challenge est le timing. Notre jeu est un plateformer dans lequel on doit faire des actions à des moments précis : sauter, tirer, lacer et se téléporter.
Tout est assez prédictible et anticipable mais les fenêtres d'opportunité peuvent être assez faibles.

Précision

Le deuxième challenge principale est celui de visé. La majorité des déplacement son basé sur le téléporteur qu'on doit lancer. Même si il subit la gravité, il faut viser dans la direction souhaité. La cible est en réalité une zone à atteindre.

Pour ce qui est du tir, il est entièrement basé sur la précision. La taille des cibles est variable.

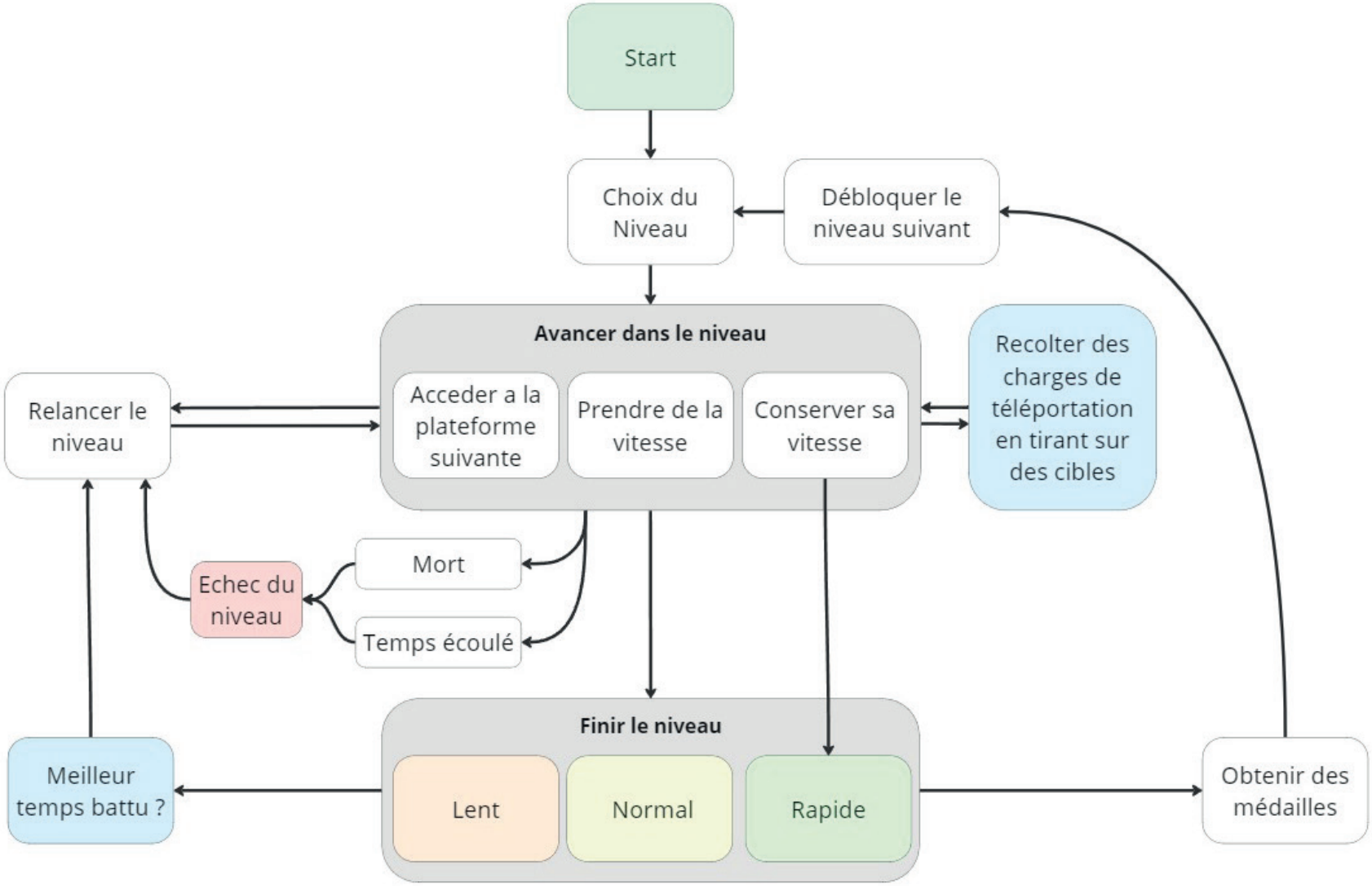
	Objectif	Challenge	Reward
Court	Récuperer une charge de téléporteur	Viser une cible et tirer au bon moment	Récupère une charge de téléporteur
	Prendre de la vitesse	Courrir ou lancer son téléporteur dans la bonne direction et se téléporter	Augmentation de la vitesse
	Atteindre la plateforme suivante	Lancer son téléporteur et se téléporté au bon moment	Progresser dans le niveau
	Se téléporter	Recuperer une charge en visant et tirant sur une cible Lancer le téléporteur dans la bonne direction	Avancer dans le niveau et prendre de la vitesse

	Objectif	Challenge	Reward
Moyen	Avancer dans le niveau	Se repérer dans l'espace et enchaîner les challenges	Progresser dans le niveau
	Conserver sa vitesse	Glisser dans la bonne direction et au bon moment	Conserver sa vitesse

	Objectif	Challenge	Reward
Long	Atteindre la fin du niveau	Bien enchainer les challenges	Finir le niveau
	Améliorer son temps	Mémorisation du niveau et meilleur enchainement des challenges	Améliorer son temps sur le niveau

Feedbacks

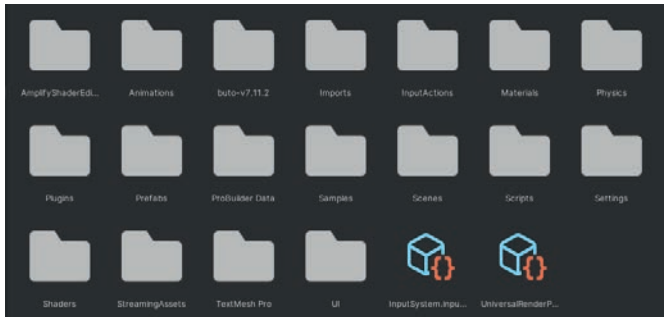
Action	Feedback	Caméra	Animation	Couleur	Son
Se déplacer		Se penche en fonction de la direction	Bras en leger mouvements de gauche à droite		Bruit de pas
Sauter		Mouvement haut bas de la caméra	Bras qui montent et descendent en fonction du temps en l'air		Frottements et son d'atterissage
Aller à une haute vitesse	Speedlines directionnelles Abération chromatique Distortion bords de l'écran	Légère augmentation FOV		Blanc Vert	Bruit de vent en fonction de la vitesse
Lancer le Téléporteur	Texte UI : «TP lancé» TP propulsé vers l'avant	Légère augmentation FOV	Animation de propulsion du projectile avec Bras gauche		Bruit de vent en fonction de la vitesse
Téléporteur en l'air	Particules derrières le TP Texte UI : «TP lancé»	Fish eye effect Distortion Abération chromatique Fov passe de 140 à normal	Animation du bras gauche	Blanc Vert	Son émis par le TP
Se téléporter	Perte d'une charge Changement de position	Fish eye effect Distortion Abération chromatique Fov passe de 140 à normal	Animation du bras gauche	Blanc Vert	Bruit distortion
Tirer	Musle Flash Bord écran assombris Trait vers la cible Anim Bras Droit		Animation de recul du bras droit	Blanc Projectile	Bruit de coup de feu Impact surface
Cible touché	Particules qui se dirige vers l'avatar Recharge un TP Destruction de la cible		Dissolution du parasite	Blanc Cible Projectile	Absorbtion energie Recharge TP
Interupteur touché	Porte s'ouvre OU Parasite détruit Interupteur s'éteind OU Destruction de la cible		Ouverture de porte Dissolution du parasite	Blanc Cible	Porte qui s'ouvre Parasite qui meurt


Start
Choix du Niveau
Débloquer le niveau suivant
Avancer dans le niveau
Acceder a la plateforme suivante
Prendre de la vitesse
Conserver sa vitesse
Recolter des charges de téléportation en tirant sur des cibles
Relancer le niveau
Mort
Echec du niveau
Temps écoulé
Finir le niveau
Lent
Normal
Rapide
Meilleur temps battu ?
Obtenir des médailles

DOCUMENTATION TECHNIQUE

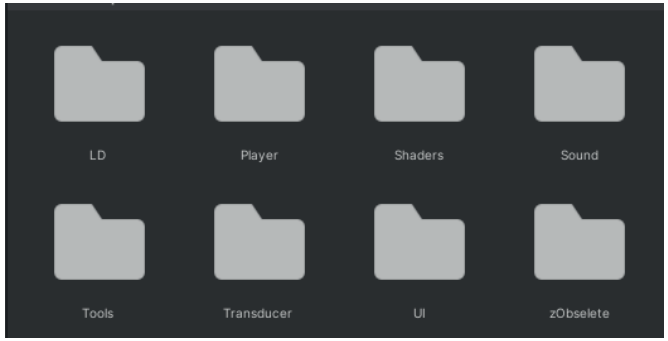
Organisation

Chaque catégorie d'asset est rangé dans son dossier a part pour retrouver facilement chaque asset en cas de besoin.



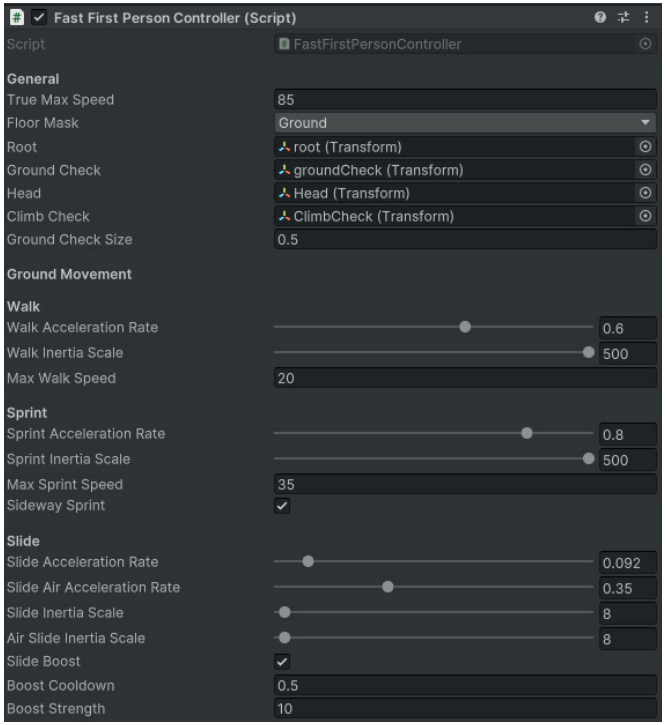
Les scripts sont organisés en différentes catégories regroupant chaque scripts qui leurs sont associés.

Tools regroupe des éléments génériques utilisés partout ou à des fins spécifique comme le GameManager ou la gestion des sauvegardes.



Inspecteur

Ce projet nécessitant énormément de peaufinage dans le fonctionnement de la physique, des mouvements de l'avatar et du téléporteur, la quasi totalité des scripts sont intégralement paramétrables.



Prefab

De plus, afin de faciliter la création de nouvelle scène, une prefab du joueur a été créé comportant tous les scripts et objets nécessaire pour ne pas avoir à tout rajouter et reparamétrer, risque d'oublier un élément important et perdre du temps.

Code

Une nomenclature a été établi afin de ne pas se perdre dans l'utilisation des variables publiques et privées :

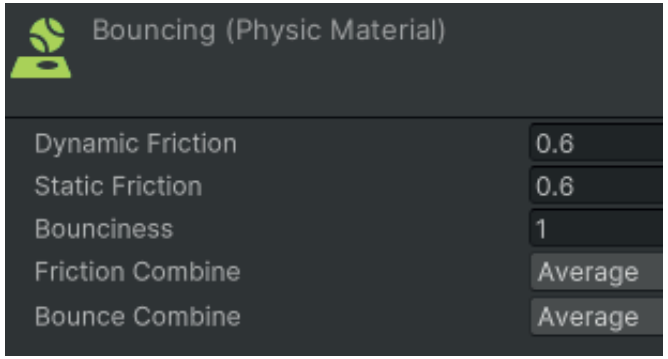
variablePublique
_variablePrivée

Téléporteur

3 actions sont possibles avec le téléporteur : le lancer, se téléporter et le rappeler.

Pour les 2 premières actions afin de faciliter la prise en main du joueur elles sont placées sur le même input dont l'action change si le téléporteur est lancé ou non grâce à une variable booléenne _inHand.

Lors du lancé, le téléporteur reçoit une force impulse fournie en paramètre dans la direction où regarde la caméra. Les rebonds de celui-ci sont géré par le boxcollider et un Material Physic qu'on lui a fourni.
Un problème majeur s'est alors posé à nous.

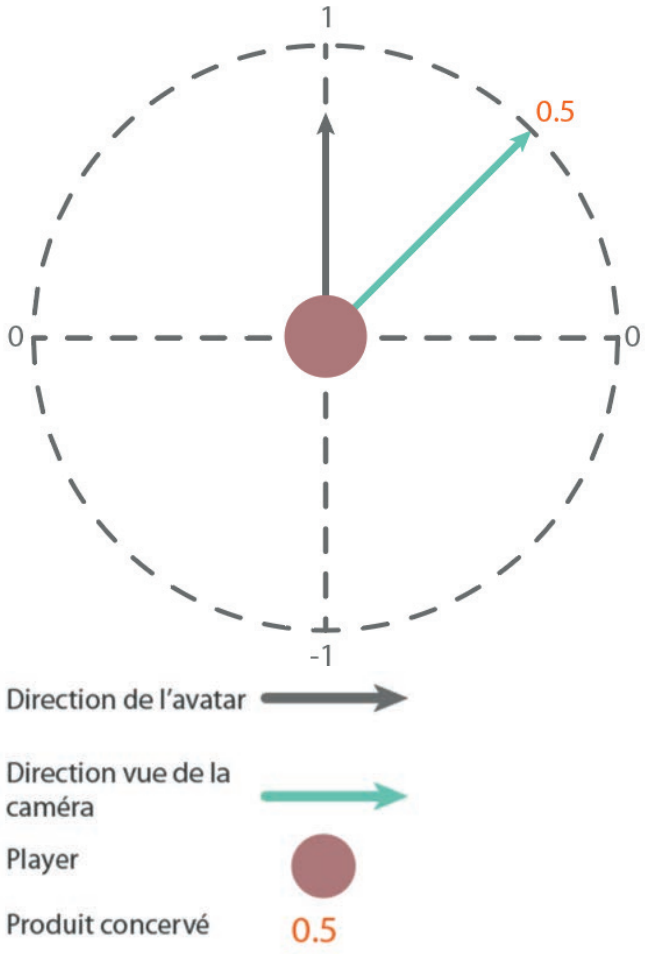


Au moment du lancé le téléporteur passait derrière le joueur car sa vitesse était inférieur à celui-ci.

Une solution que nous avons trouvé pour résoudre ce problème est d'ajouter une partie de la vitesse du joueur à celle du transducteur. Pour cela nous avons utilisé un produit scalaire (dot product, toujours compris entre 1 et -1) afin de calculer le "pourcentage" de la vitesse du joueur que l'on conserve entre la direction du joueur et la direction dans laquelle on regarde.

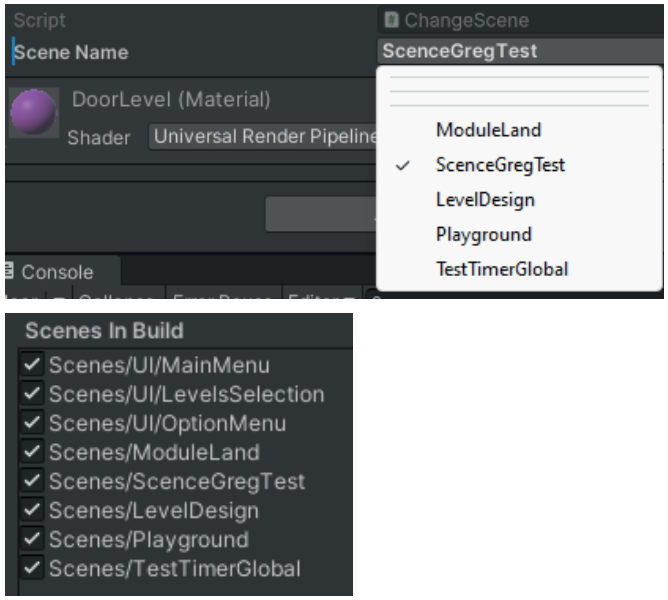
On limite ce produit à 0 pour simuler un angle de 180°.

Si dot product = 1, on conserve 100% de la vitesse
Si dot product = 0, on conserve 0% de la vitesse



Outils

Afin de faciliter le travail pour la création de niveau et l'ajout des zones de fin. Un menu déroulant affichant tous les niveaux est ajouté dans le build, qui ne sont pas dans un dossier UI, pour éviter de devoir toujours écrire le nom et éviter d'éventuelles erreurs d'inattention.



Sauvegarde

Afin de conserver une trace de la progression des joueurs une sauvegarde local est faite sur leur machine, grâce au PlayerPrefs, pour sauvegarder les temps de complétion des niveaux notamment.

Au premier chargement de l'instance du game manager, une copie des sauvegardes locales des niveaux est faite en reprenant tous les niveaux présents dans la build, si un des niveaux n'a pas de sauvegarde, une par défaut est créée.

```
public void CreateListScenes()
{
    if (scenesDico.Count == 0 ) // ne fait pas si déjà remplis
    {
        Debug.Log("------ CREATION ------");
        for (int i = 0; i < SceneManager.sceneCountInBuildSettings; i++)
        {
            string scenePath = SceneUtility.GetScenePathByBuildIndex(i);

            // Ne stocke pas la scène si elle est dans le dossier "UI"
            if (!scenePath.Contains("/UI/"))
            {
                string sceneName = System.IO.Path.GetFileNameWithoutExtension(scenePath);

                TupleScoreUnlock _props = new TupleScoreUnlock();

                // Si n'existe pas, le créer
                if (!PlayerPrefs.HasKey(sceneName))
                {
                    _props = TupleInfoLevel(999f, sceneName);//gros temps par defaut pour set le best timer plus tard
                    //local save
                    PlayerPrefs.SetString(sceneName, $"{999f}_{false}");
                }
                else
                {
                    _props = TupleLocal(sceneName);
                }
                scenesDico.Add(sceneName, _props);
            }

        }
```

Sauvegarde

Stocké comme un dictionnaire sous forme de key=›value, nous nous servons du nom de la scène, du LV, comme key unique et lui donnons une chaîne de caractère contenant le meilleur temps réalisé et si le niveau est débloqué ou non (pour prévoir des niveaux à débloquer mais qui n'est pas implémenté)

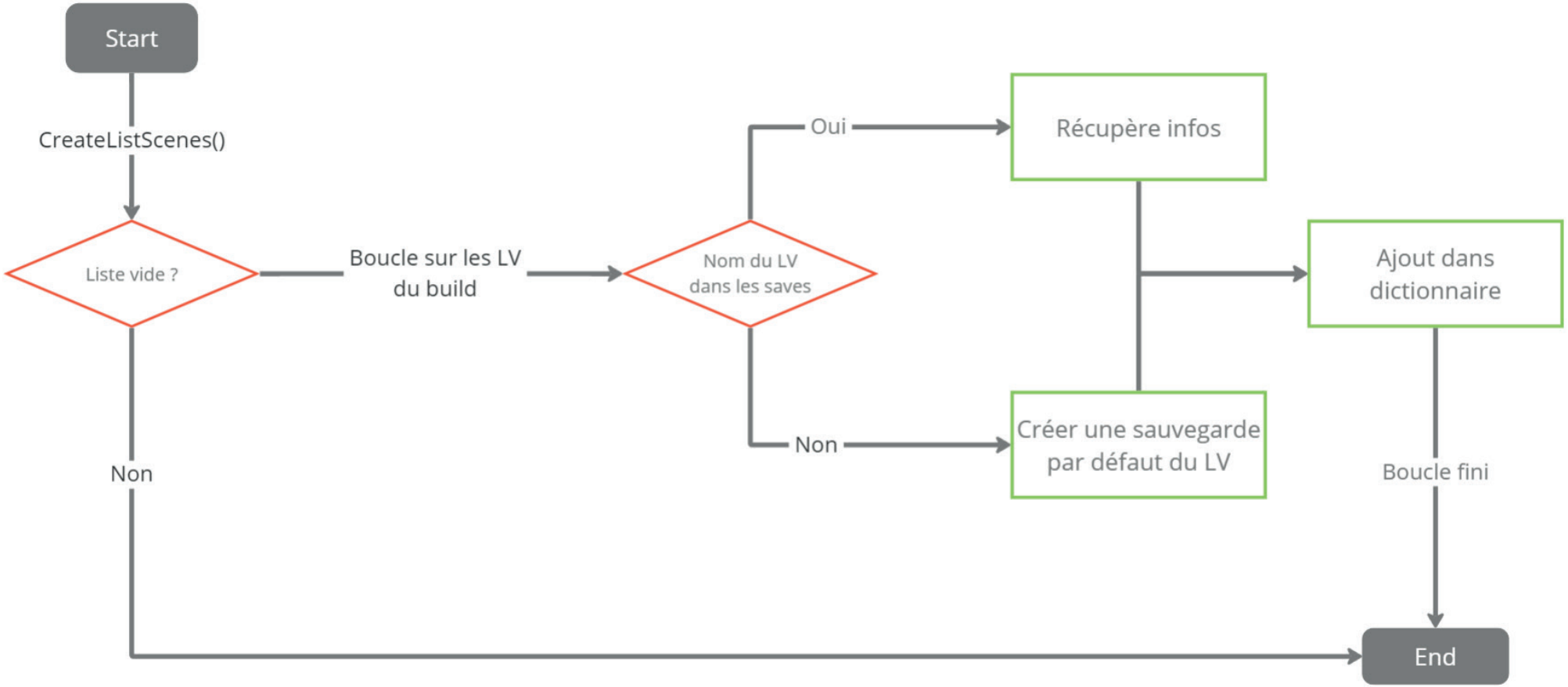
string tupleTimerUnlock = $»{_bestTimer}_{_isUnlock}»;

Suivant le même schéma, la copie des sauvegardes dans unity se présente sous la forme d'un dictionnaire avec le nom du LV en clé et un tuple de valeur, une structure de donnée contenant un float bestTimer et un boolean isUnlock que l'on récupère en séparant la chaîne de caractère à partir du séparateur "_".

Pour finir à chaque fin de niveau, on va comparer le meilleur temps de ce niveau contenue dans notre copie de sauvegarde et le temps de complétion du niveau, mettre à jour le meilleur temps si besoin puis mettre à jour la sauvegarde local

```
public void LocalSave(string _lvName, float _timer, bool _isUnlock)
{
    TupleScoreUnlock _infoLv = TupleLocal(_lvName); // sceneDictionary[_lvName]
    float _bestTimer = _infoLv.bestScore;

    if (_timer < _bestTimer)
    {
        _bestTimer = _timer;
    }

    //convertion en string, plus simple à gérer comme ça
    string tupleTimerUnlock = $"{_bestTimer}_{_isUnlock}";
    Debug.Log(tupleTimerUnlock);

    //save local avec nom de lv en key
    PlayerPrefs.SetString(_lvName, tupleTimerUnlock);
}
```

```
public struct TupleScoreUnlock
{
    public float bestScore;
    public bool isUnlock;
}
```



Flowchart de la Création de la copie des sauvegarde au chargement du jeu

Rebind des touches

Etant un jeu ciblant les hardcores, chaque joueur a sa propre configuration de touche. Leur donner la possibilité de changer leurs touches nous semblait donc important.

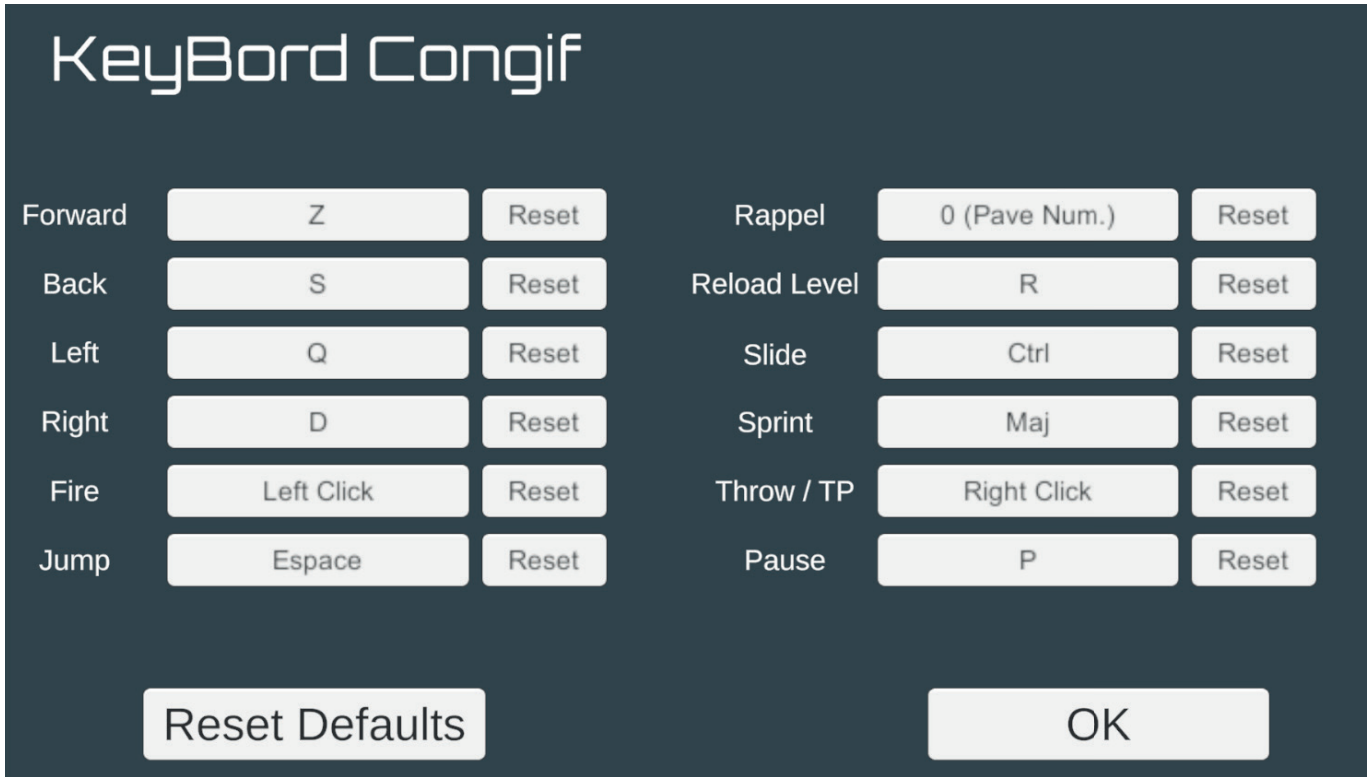
Pour cela nous avons repris l'exemple fourni par l'asset d'input system pour le "rebind UI", modifié par la suite pour correspondre à nos attentes.

A l'ouverture du menu, un script, RebindSaveLoad, charge les rebinds sauvegardés en local quand le menu s'ouvre pour les afficher dans le menu et les applique au controler des inputs.

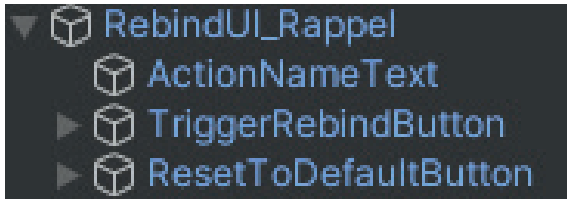
Quand on le ferme, il applique les rebind présents dans le menu au controler des inputs et les sauvegardes en local.

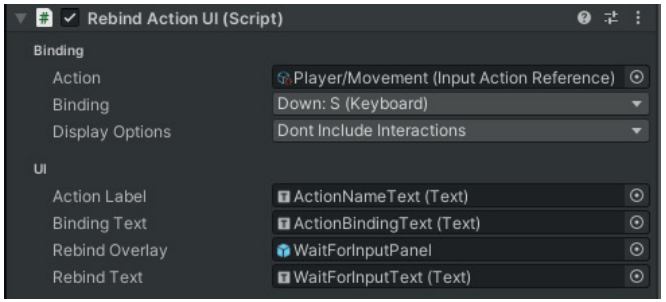
```
Script Unity (3 références de ressources) | 0 références
public class RebindSaveLoad : MonoBehaviour
{
    public InputActionAsset actions;
    public bool justLoad;

    Message Unity | 0 références
    public void OnEnable()
    {
        var rebinds = PlayerPrefs.GetString("rebinds");//recupère rebind save en local

        if (!string.IsNullOrEmpty(rebinds))
        {
            actions.LoadBindingOverridesFromJson(rebinds);//charge "copie" UI

            GameManager.controls.Disable();
            GameManager.controls.asset.LoadBindingOverridesFromJson(rebinds);
            GameManager.controls.Enable();
        }
    }

    Message Unity | 0 références
    public void OnDisable()
    {
        if (!justLoad) {
            var rebinds = actions.SaveBindingOverridesAsJson();//save "copie"
            PlayerPrefs.SetString("rebinds", rebinds);// sauvegarde des rebinds
        }

    }
```

Le menu de rebind est composé de prefabs de bouton composé du nom de l'input, un bouton cliquable qui affiche l'input associé à l'action et un bouton pour remettre l'input par défaut.



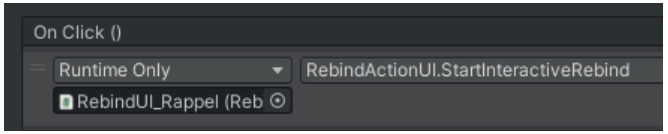
Le parent "RebindUI_action" contient le script RebindActionUI qui prend en paramètre l'action que l'on veut rebind et les différents champs textes qui changent dynamiquement en fonction de l'action et l'écran à afficher lors du rebind.





Lors du clique sur le bouton "TriggerRebind" la fonction StartInteractiveRebind est appelée et va effectuer plusieurs actions et vérifications dans le code pour associer une nouvelle touche à l'action associée.

Comme vérifier si l'action est "composite" (plusieurs input pour une action comme le mouvement), vérifier que le nouvel input n'est pas déjà utilisé et bloqué le rebind si c'est le cas.

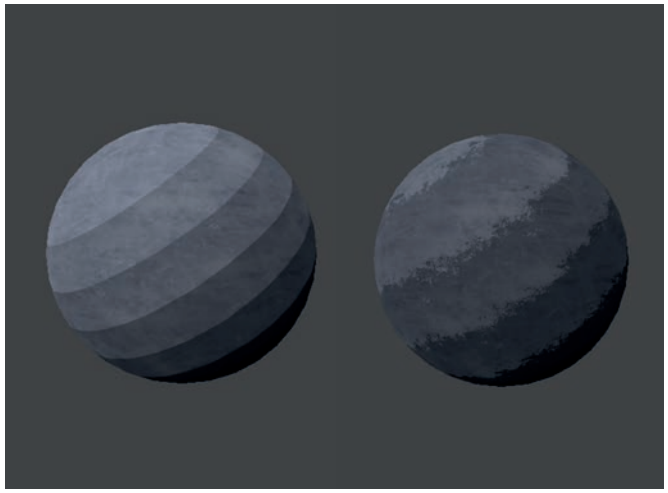


Shading

Nous utilisons une technique de cel shading classique à laquelle nous avons ajouté un support de normal map et roughness map, afin d'apporter un aspect "noisy" au traitement de la lumière.

La normal map est injectée avant le traitement comme dans un shader ordinaire.

La roughness map, elle, est multipliée à la lumière juste avant que celle-ci ne soit altérée par la Shadow Ramp.



En jeu, le résultat est donc le suivant : Des ombres plus "noisy", des cellules plus distinguables et plus de contraste au global.

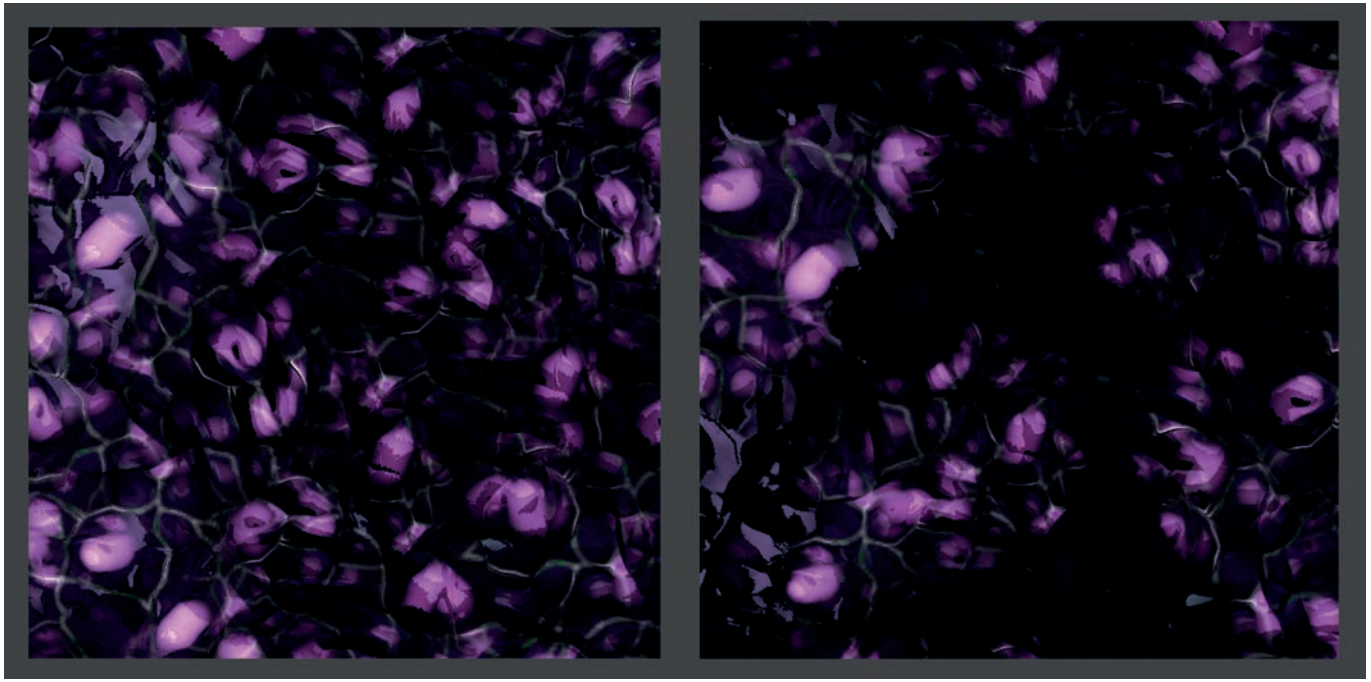
Parasite

Le parasite reprend les principes évoqués plus haut en ajoutant un support de map pseudo-émissives (elle n'émet pas de lumières du fait que nous utilisons URP qui ne supporte pas les lumières émissives en temps réel.).

Nous avons également modifié la grille pour mapper le temps sur le déplacement des uv afin d'appliquer un mouvement continu sur le parasite, la rotation de chaque tuile étant déjà aléatoire, elle se déplace toute dans un sens différent, donnant un aspect grouillant au parasite.

A cause de ce mouvement, en observant le parasite, on pouvait voir les délimitations de la grille de tiling. Nous avons ajouté à cela un masque noir et réfléchissant sur la texture, mouvant de manière asynchrone à celle-ci.

Celui-ci a eu plusieurs bénéfices : casser les délimitations de la grille, apporter un niveau de détail supplémentaire et renforcer l'aspect "grouillant" du matériau en général.



Tiling

L'environnement de (jeu) est basé sur deux matériaux différents. Un matériau de "Béton" et un matériau de "Parasite".

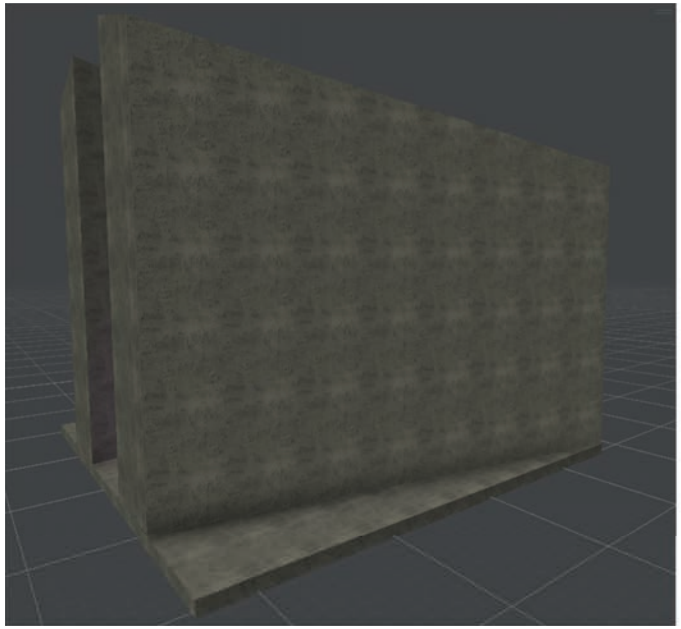
Les deux utilisent une technique de Tiling personnalisée pour rendre la répétition des tuiles plus subtiles.

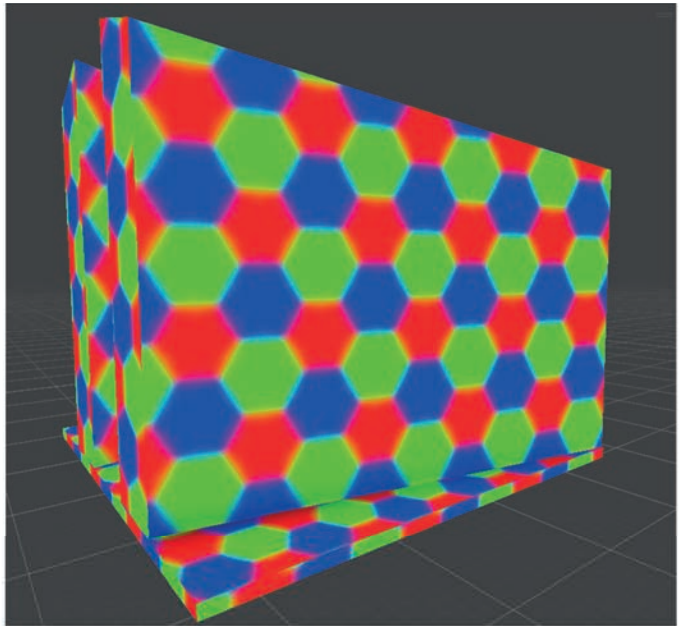
Celle-ci se base sur une grille sur laquelle on vient appliquer la texture en l'altérant de plusieurs façons.

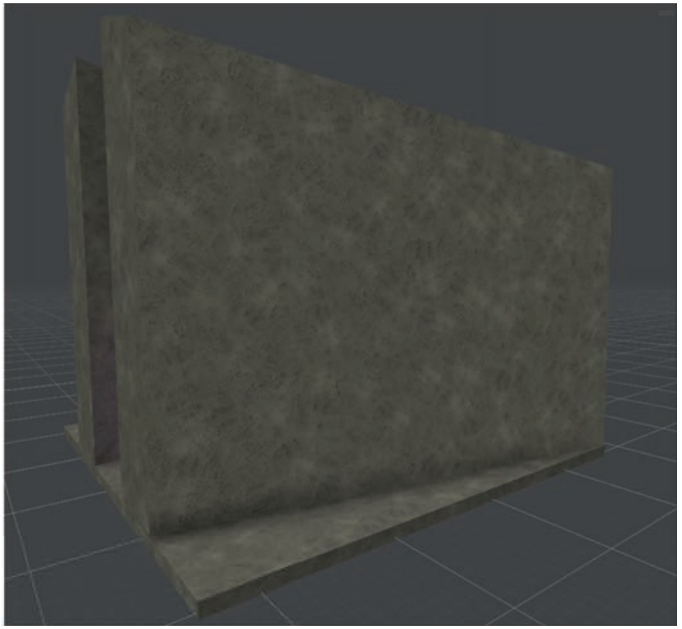
1. Elle est découpée de manière hexagonale pour permettre une meilleure liaisons entre les tuiles comparées à une grille de carrés classique.
2. Sur chaque tuile est appliquée une rotation, un offset et une modification de la taille aléatoire.
3. Pour lier chaque tuile ensemble, les bords de chacune sont floutés pour permettre une meilleure transition entre chaque tuile.

Ces trois altérations permettent au tiling d'être infini sans rencontrer de répétition perceptible.

Dans notre environement grandement composé de béton, c'était très important de pouvoir varier les textures sans avoir à tout faire "à la main".



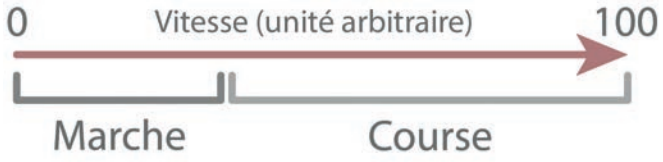




Le Controller est basé sur une state machine comportant 4 états : Marche, Course, Slide, Crouch. Nous avions besoin du fait du GD, d'un controller dynamique et réactif, il était donc important que celui-ci puisse hiérarchiser les états correctement.

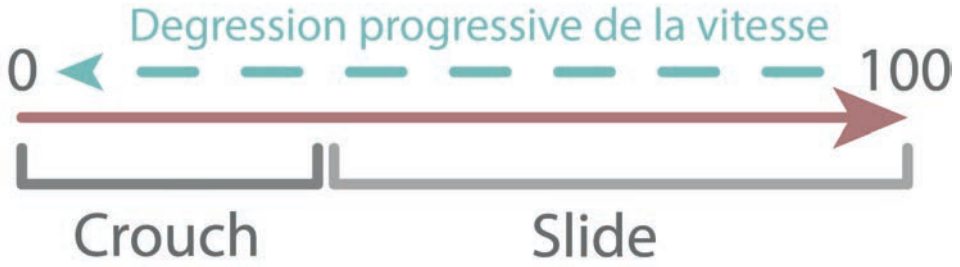
Les états de marche et de course ont une limite de vitesse, entrer dans un de ces états entraîne une accélération vers cette limite, si la vitesse du joueur est supérieur au cap de l'état dans lequel il entre.

Si cet état est la marche, sa vitesse est instantanément réduite au cap de vitesse de marche. Dans l'état de course, la réduction est progressive.



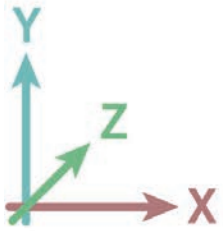
Changement d'état lorque la touche de slide est relaché

L'état slide n'a pas de limite de vitesse, lorsque le joueur entre dans cet état, il gagne un boost sur sa vitesse actuelle, puis, perds de la vitesse en continu jusqu'à ce qu'il atteigne une certaine vitesse qui le fera repasser en mode crouch.



Le controller utilise un rigidbody, auquel on applique une combinaison de modification de la vélocité et d'ajout de force (pour le boost et le saut).

L'intérêt de cette technique hybride est d'avoir un contrôle très précis sur la vitesse de l'avatar via la modification de la vélocité et d'ajouter des forces "enfants" (dépendantes) de cette vélocité via les AddForces.



$$XZ = \begin{matrix} \text{Rigidbody.Velocity} = (\text{inputs} * \text{maxSpeed}) \searrow_0 \\ \text{Rigidbody.Addforce(slideBoost)} \end{matrix}$$

$$Y = \text{Rigidbody.Velocity} = \text{JumpForce}$$

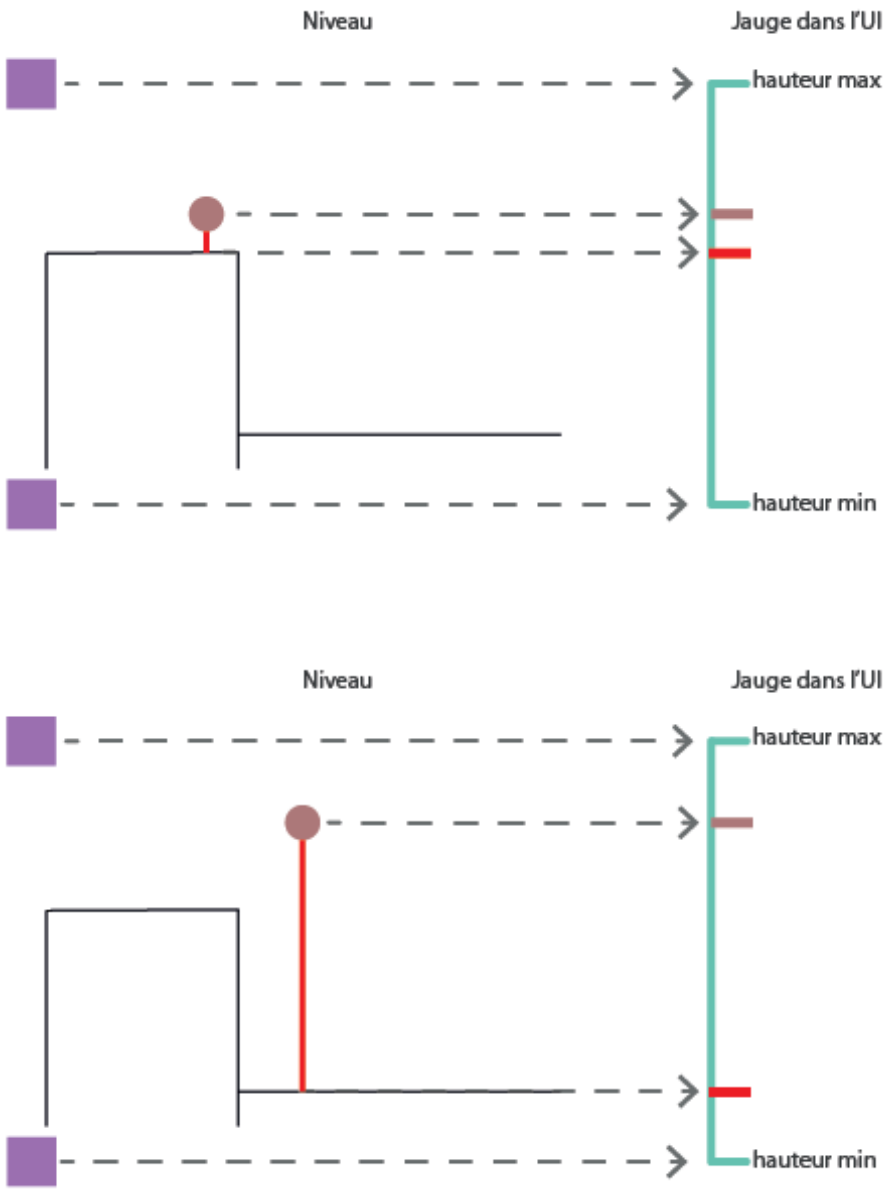
Altimètre

Étant en constant mouvement et majoritairement dans les airs, la caméra FPS nous permettait difficilement de savoir à quel moment on allait toucher le sol. Cela pouvait poser problème notamment pour bien timer des téléportations lors de chutes avec le parasite en dessous de l'avatar.

Pour remédier à cela nous avons pensé à un système d'altimètre en nous inspirant de celui de Fortnite qui indique la position du joueur et le moment du déploiement de parachute.

La différence majeure avec notre problématique est que la hauteur du sol sous notre joueur n'est jamais la même. Pour remédier à cela nous avons pensé à placer 2 objets à la hauteur maximale et minimale de nos niveaux qui nous serviront de référence pour la position du joueur et du sol. Pour connaître la position du sol, un Raycast est tiré du joueur vers le bas et on récupère la position de ce qu'il touche.

Ces positions sont constamment mises à jour et affichées dans une jauge sur l'écran du joueur lui permettant de savoir constamment quand il touchera le sol.



DIRECTION ARTISTIQUE

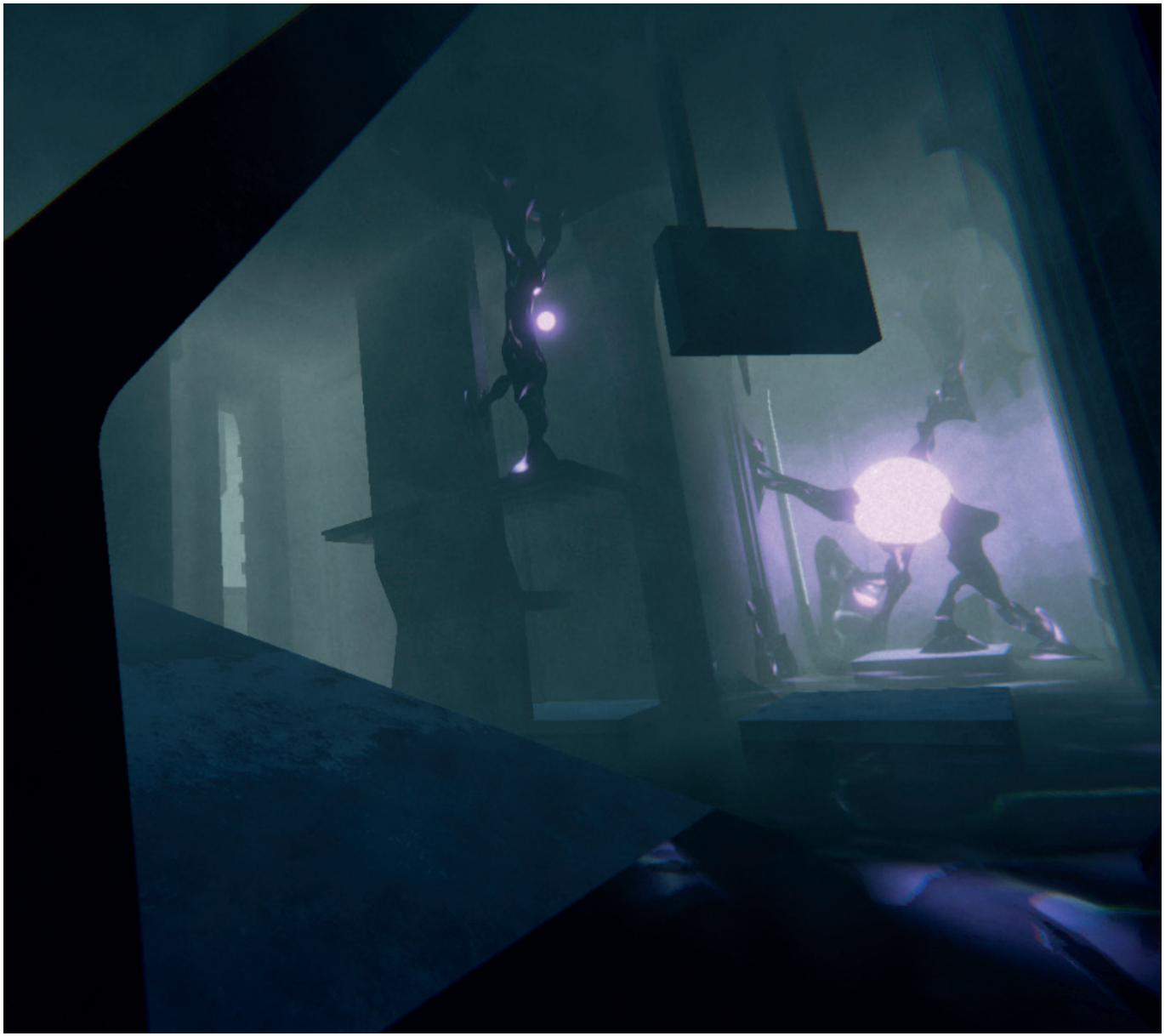
Nous voulions que le joueur ait à sa disposition un environnement où il pourrait s'exprimer en termes de mouvement.

Nous voulions également que l'environnement soit cohérent en termes d'esthétique sans abandonner la lisibilité.

Nous nous sommes donc limités à deux éléments de style majeurs pour l'environnement et avons adapté nos choix de la manière la plus cohérente possible.

Le but est de créer du contraste entre les deux éléments pour que la lisibilité ne soit pas influencé par la haute vitesse de l'avatar.

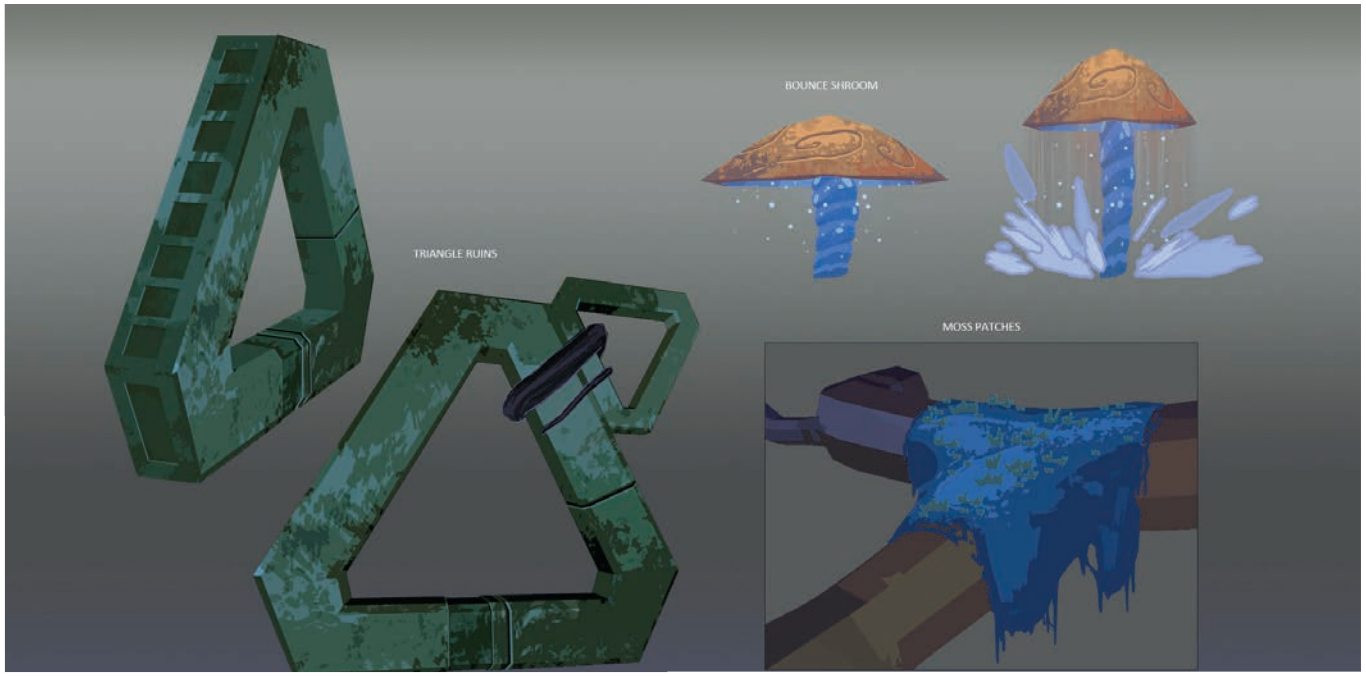
Nous avons choisi de styliser le langage graphique avec du cel shading pour donner une homogénéité dans le style du jeu.

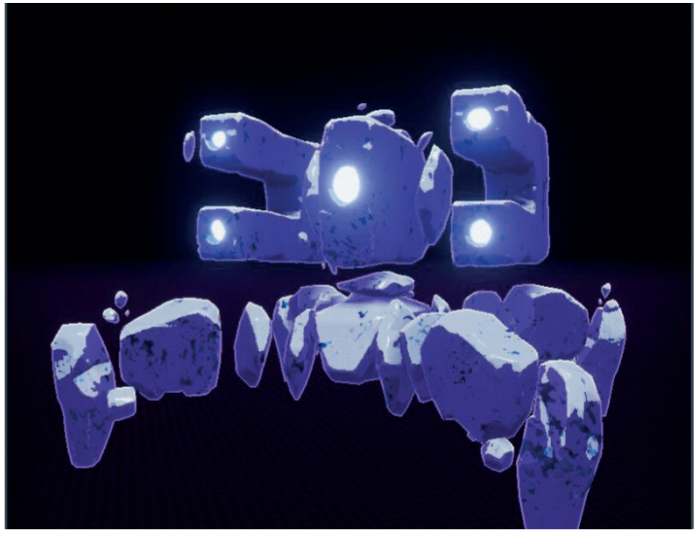


Langage Graphique

Notre inspiration principale pour le langage graphique est le jeu Risk Of Rain 2 et son interprétation particulière du cel shading.

Là où le cel shading est en principe caractérisé par son aspect lisse, les transitions entre les différentes cases de lumières sont ici “brouillées”. Nous avons répliqué cet aspect pour nos propres shaders.





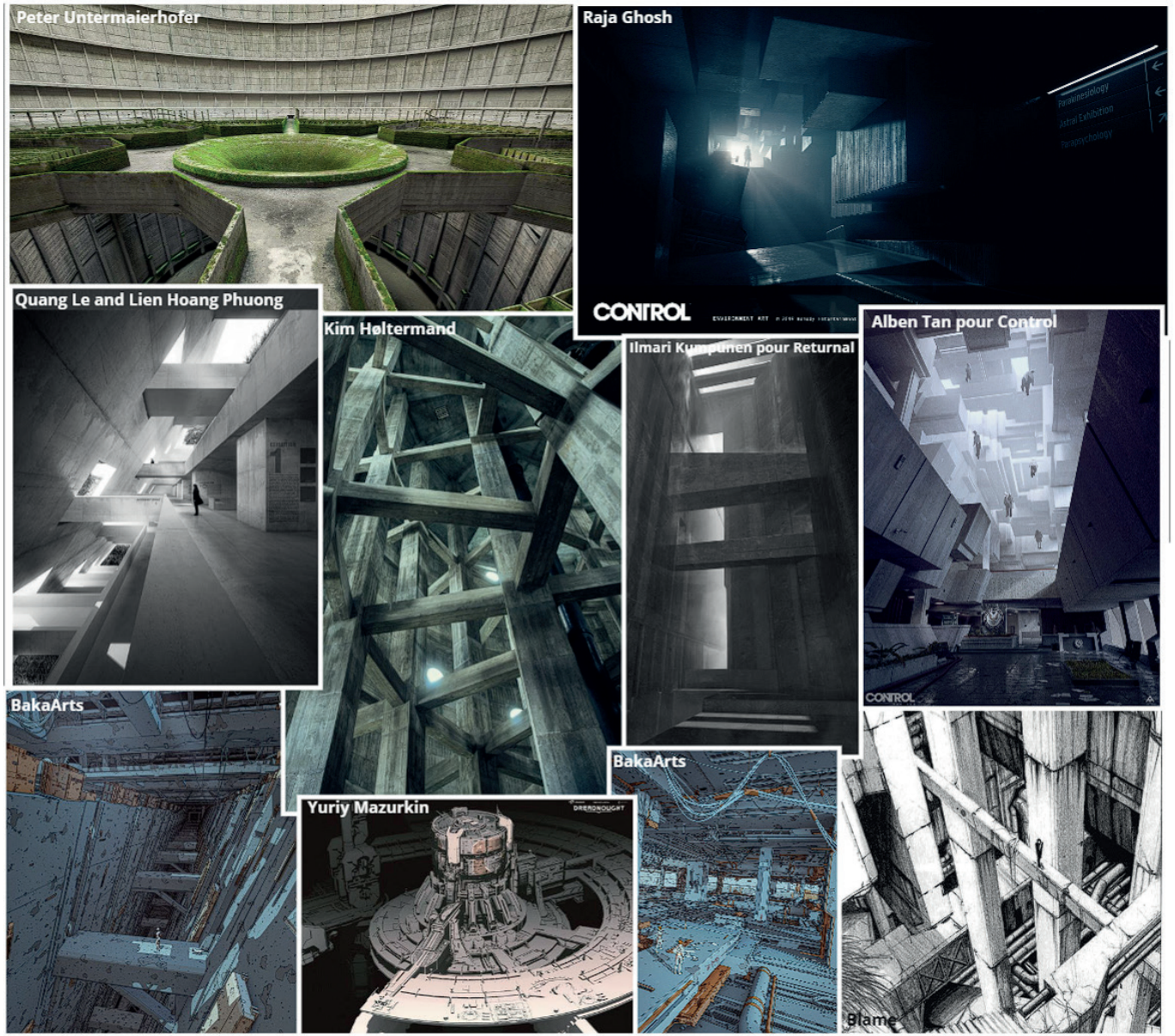
Notre but pour l'environnement de Shift est de présenter un univers à la fois lumineux et épuré mais froid, mystérieux et intimidant.

Pour ce faire, nous nous basons sur une architecture brutaliste corrompue par un étrange organisme.

Le brutalisme est un courant architecturale datant des années 50 ayant pour caractéristique d'être très brut. Souvent composé de béton uniquement, les bâtiments brutaliste sont très froid et imposant.

Ils sont aussi caractérisés par leurs formes assez carrés et plates très reconnaissables.

D'autres inspirations viennent ajouter à cette architecture. Certains visuel du manga Blame notement ou des visuels du jeu Control se passent dans des bâtiments similaires au brutalisme.



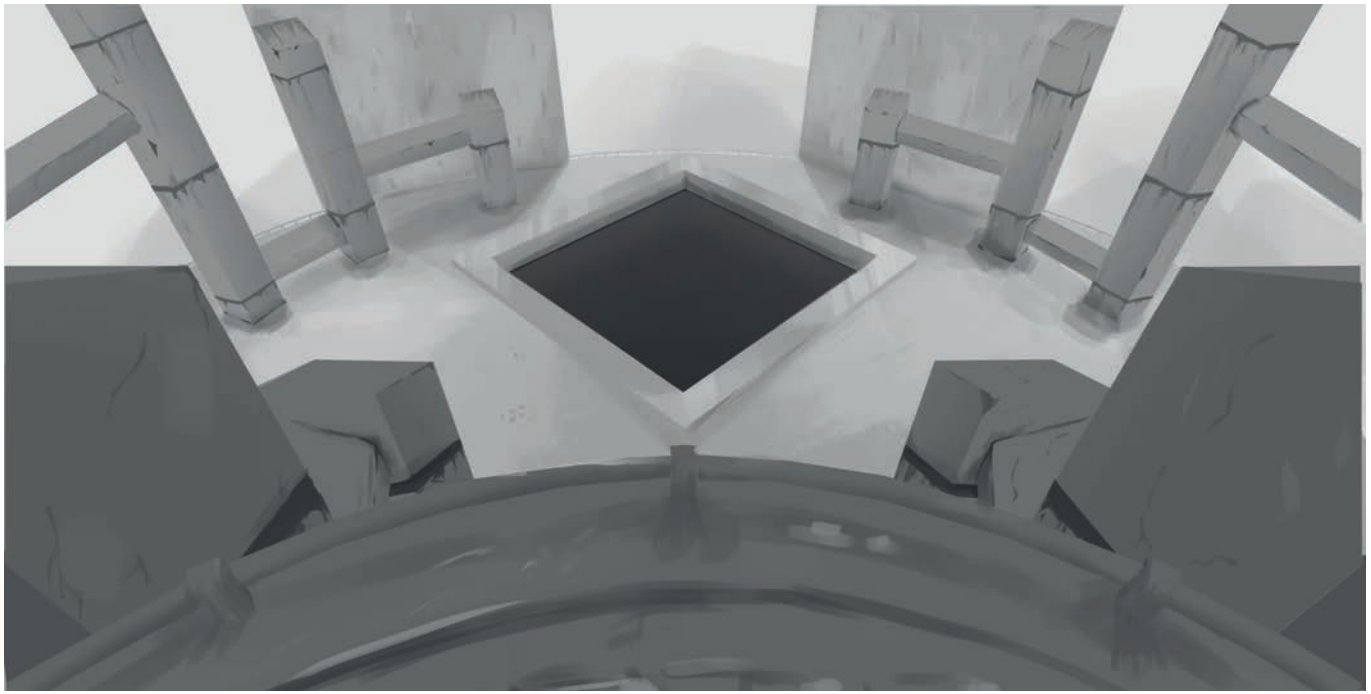
Intentions D'environnement

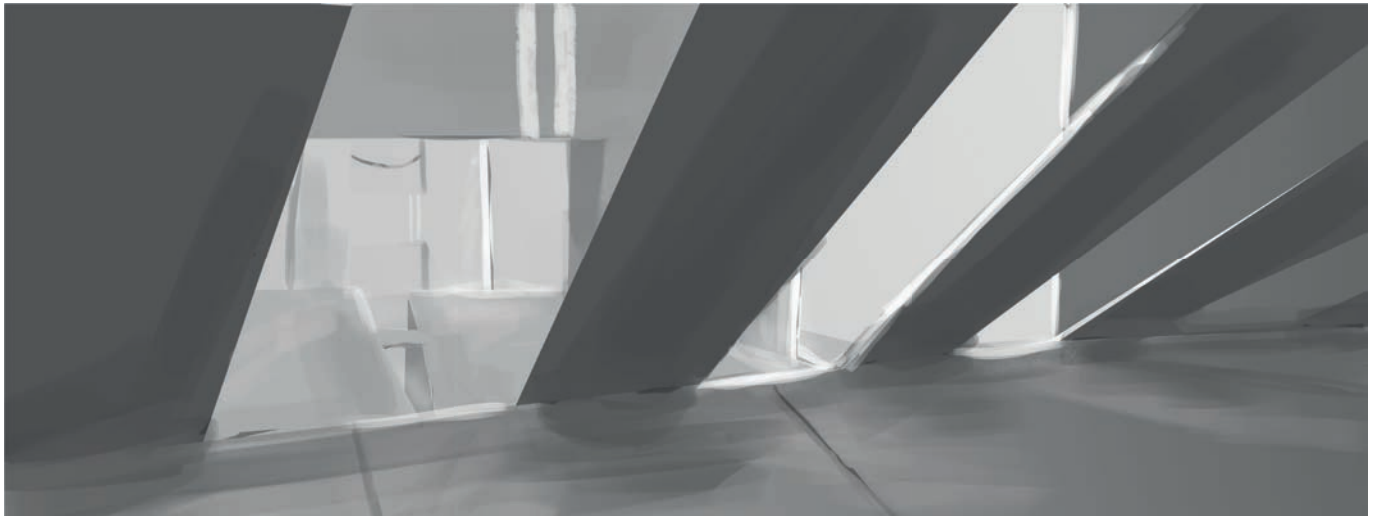
Nous avions besoin d'un environnement massif, avec des grands espaces mais néanmoins fermé. Nous avons donc pensé à l'architecture brutaliste : un bon compromis pour le cahier des charges et peu dépendante en termes de production

Dans cet environnement nous avions besoin d'ajouter des landmarks, des cibles et des obstacles.

Pour remplir ces trois rôles nous avons eu l'idée d'ajouter un parasite.

Il se répand dans l'environnement, créant des zones de mort, contient des "têtes", créant des cibles et présente un fort contraste avec le béton le rendant repérables de loin et l'habille.

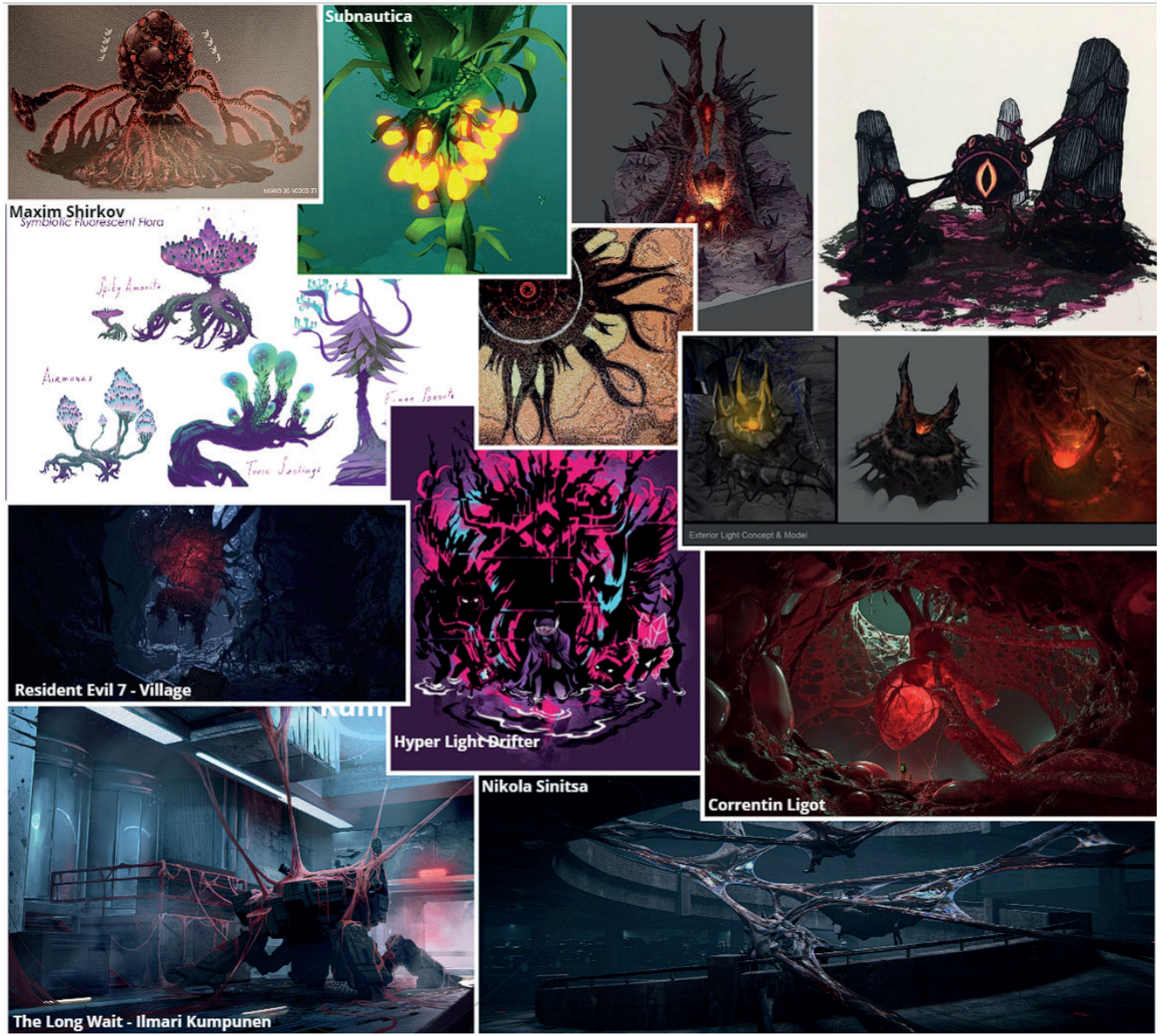




Grâce à l'environnement présenté précédemment, le joueur progresse et évolue dans un décor lisible et offrant de la liberté de mouvement.

Pour créer des objectifs et des cibles, il faut contraster avec cet environnement plat et épuré pour que ces ingrédients ressortent du décor. Nous avons donc décidé de peupler cette architecture de créatures organiques difforme.

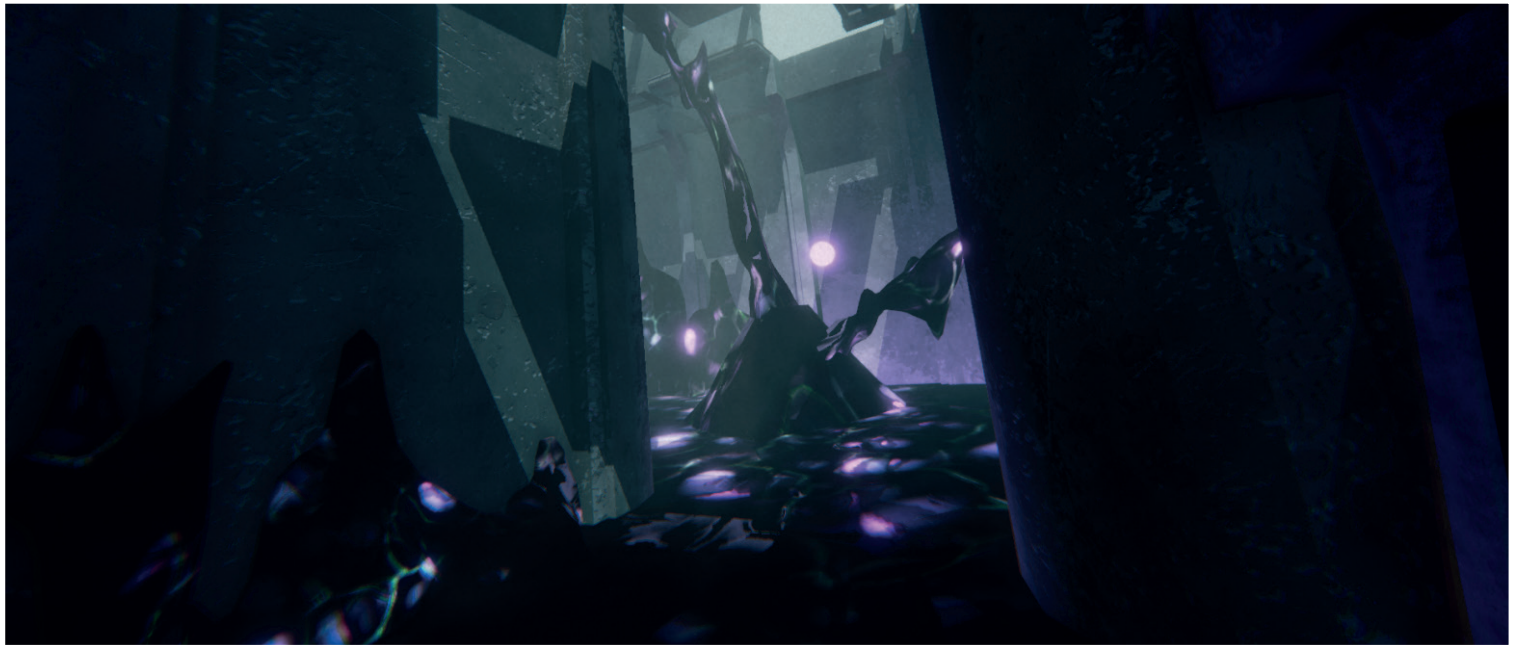
Comme dit précédemment, les créatures qui peuple l'environnement sont utilisées comme des objectifs, des landmarks et des cibles. Elles sont des guides visuels pour le chemin à suivre du joueur.



Intentions D'environnement

Nous voulions que le parasite ait un aspect “goo” et ait une apparence qui n’évoque pas forcément une créature à première vue, afin de pouvoir en couvrir des grandes zones.

On comprendrait que c’est une créature en voyant la “tête”.



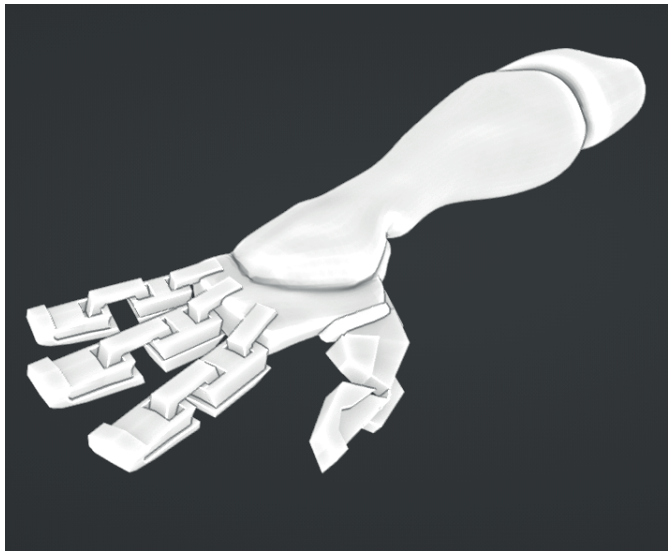
Avatar

Nous avions comme intention pour le personnage d'avoir un cyborg.

En effet, il est humanoïde, capable de déplacer à très grande vitesse, de se téléporter, a une interface / visière mais il doit être sensible à l'attaque du parasite.

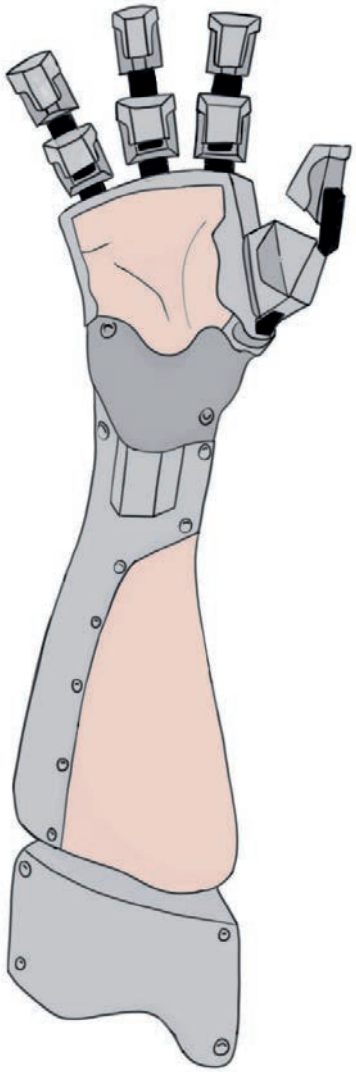
Faire un personnage mi-humain mi-robot semblait être la bonne solution

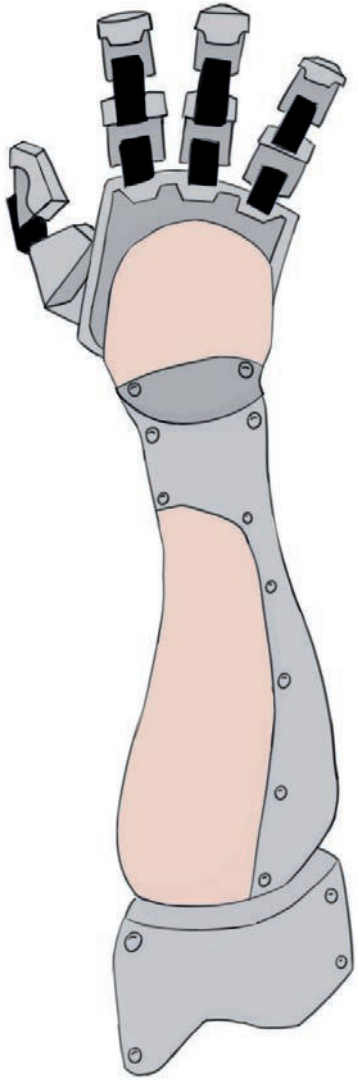
Les parties les plus importante pour le FPS sont les deux bras et le point de vue (ce que l'avatar voit : informations à l'écran)



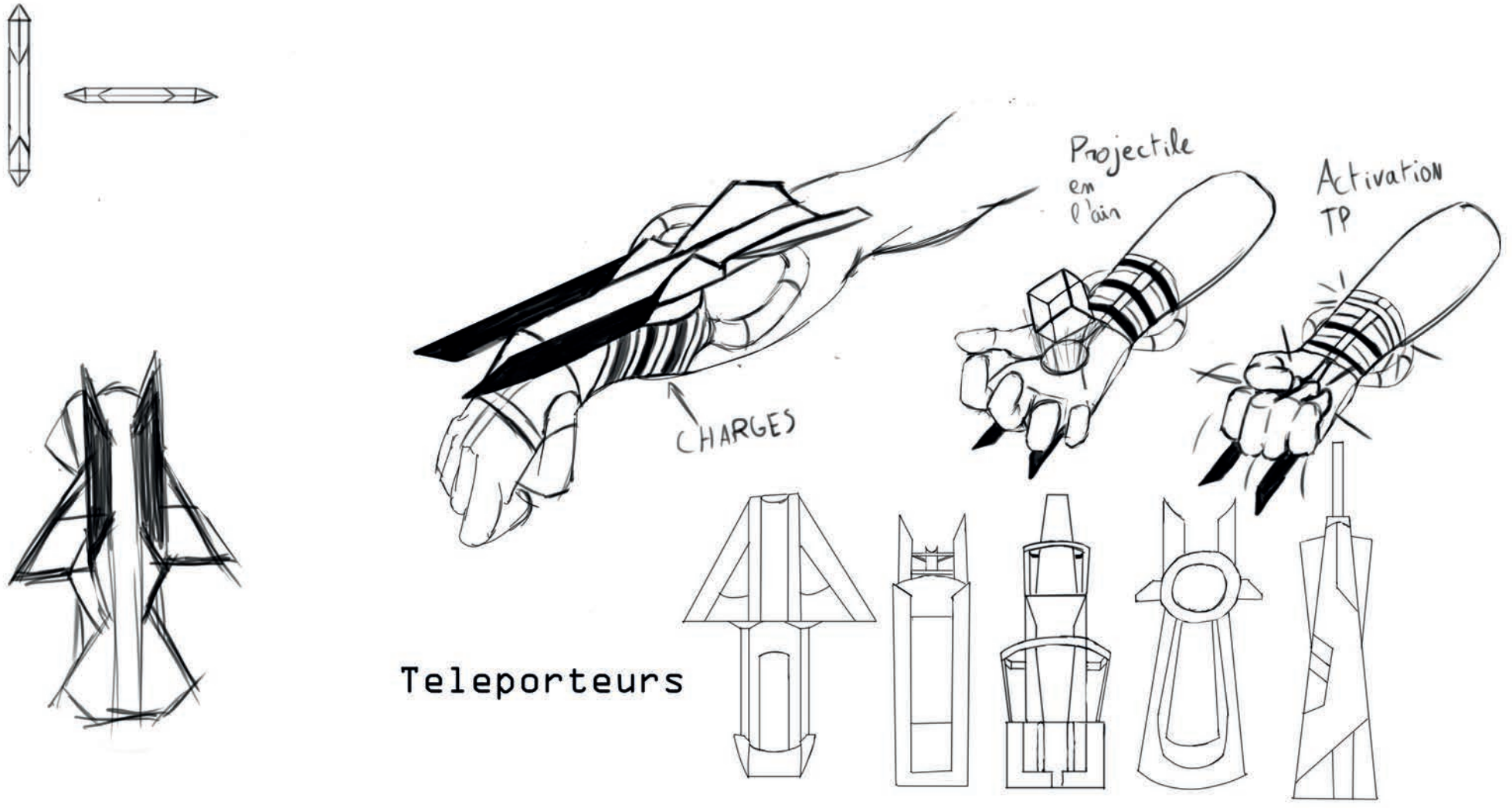
Pour les bras, nous avons choisi de montrer les deux coté de l'avatar : Humain et Robot.

On voit donc des parties mécaniques et des parties de peau pour faire comprendre ce mélange.





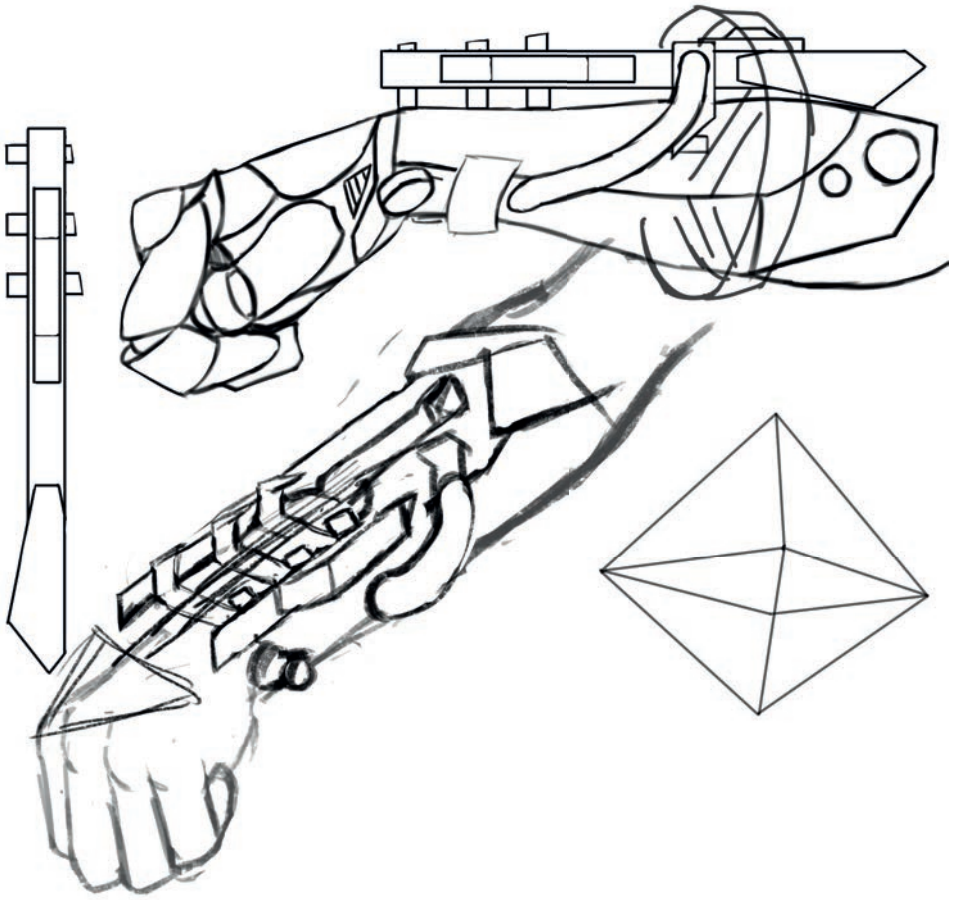
Le téléporteur est un dispositif qui s'accroche sur le bras, projette un objet géométrique sur lequel le joueur peut se rappeler.
Ci dessous les itérations du lanceurs de téléporteur accroché au bras gauche.

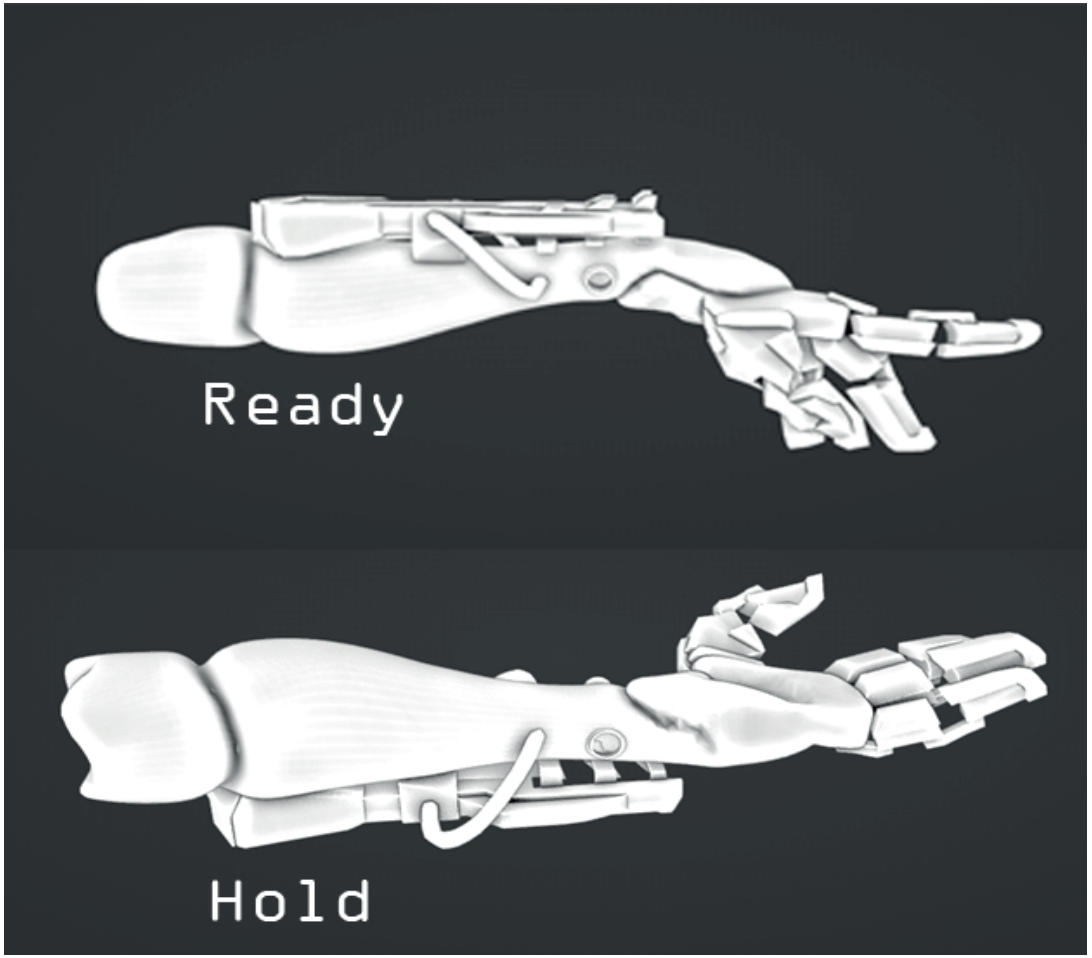


Lanceur

Le lanceur final est une sorte de propulseur énergétique qui lance a haute vitesse le téléporteur sans utiliser la force du bras.

Lorsque le projectile est en l'air, une image de celui-ci est affichée sous forme d'hologramme dans la paume du bras qui porte le dispositif. Cela nous permet d'avoir une séparation nette des états

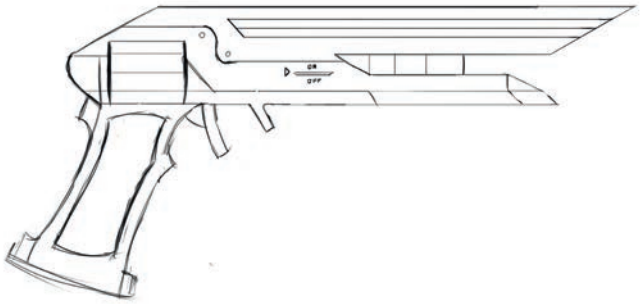


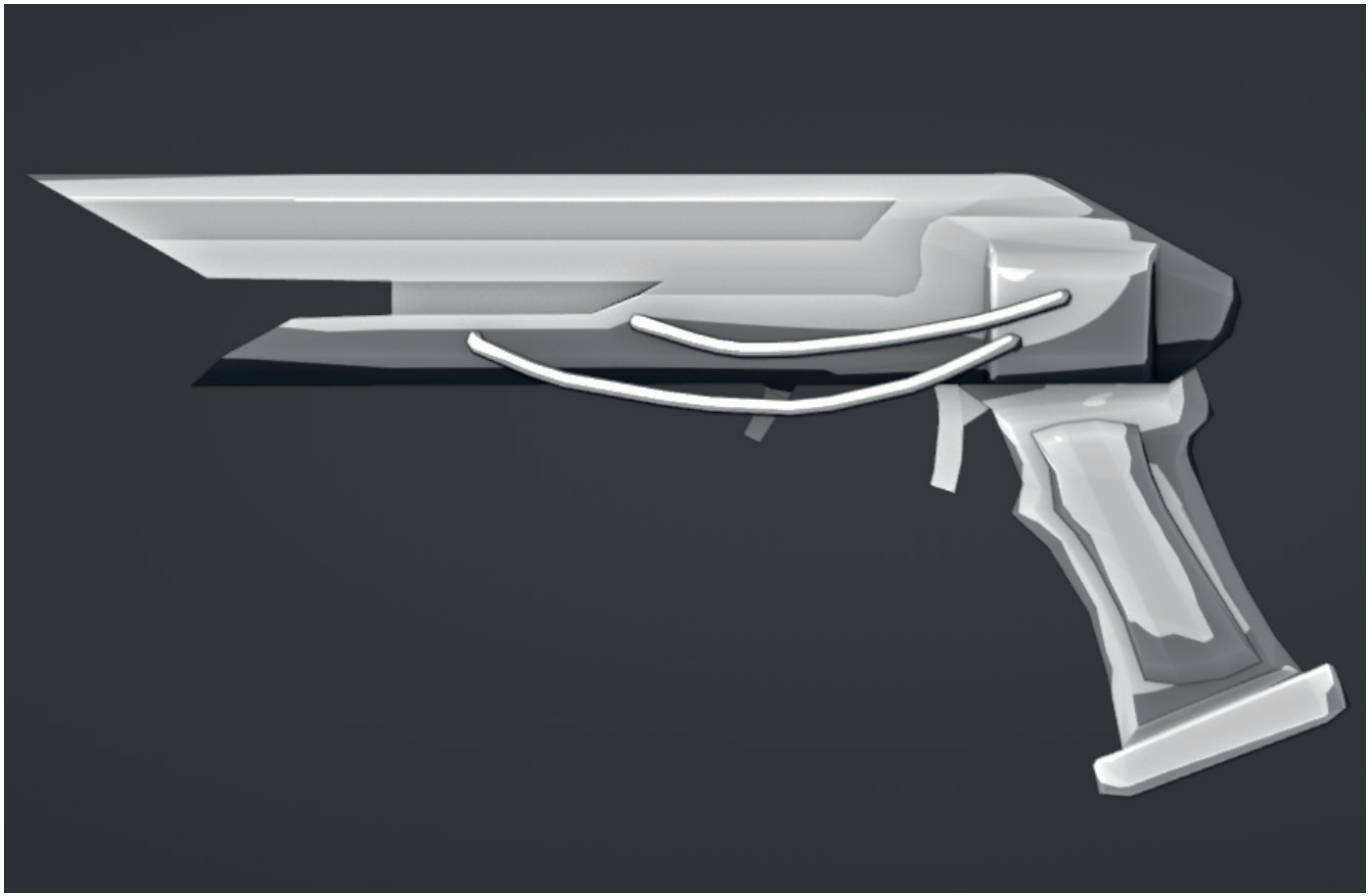


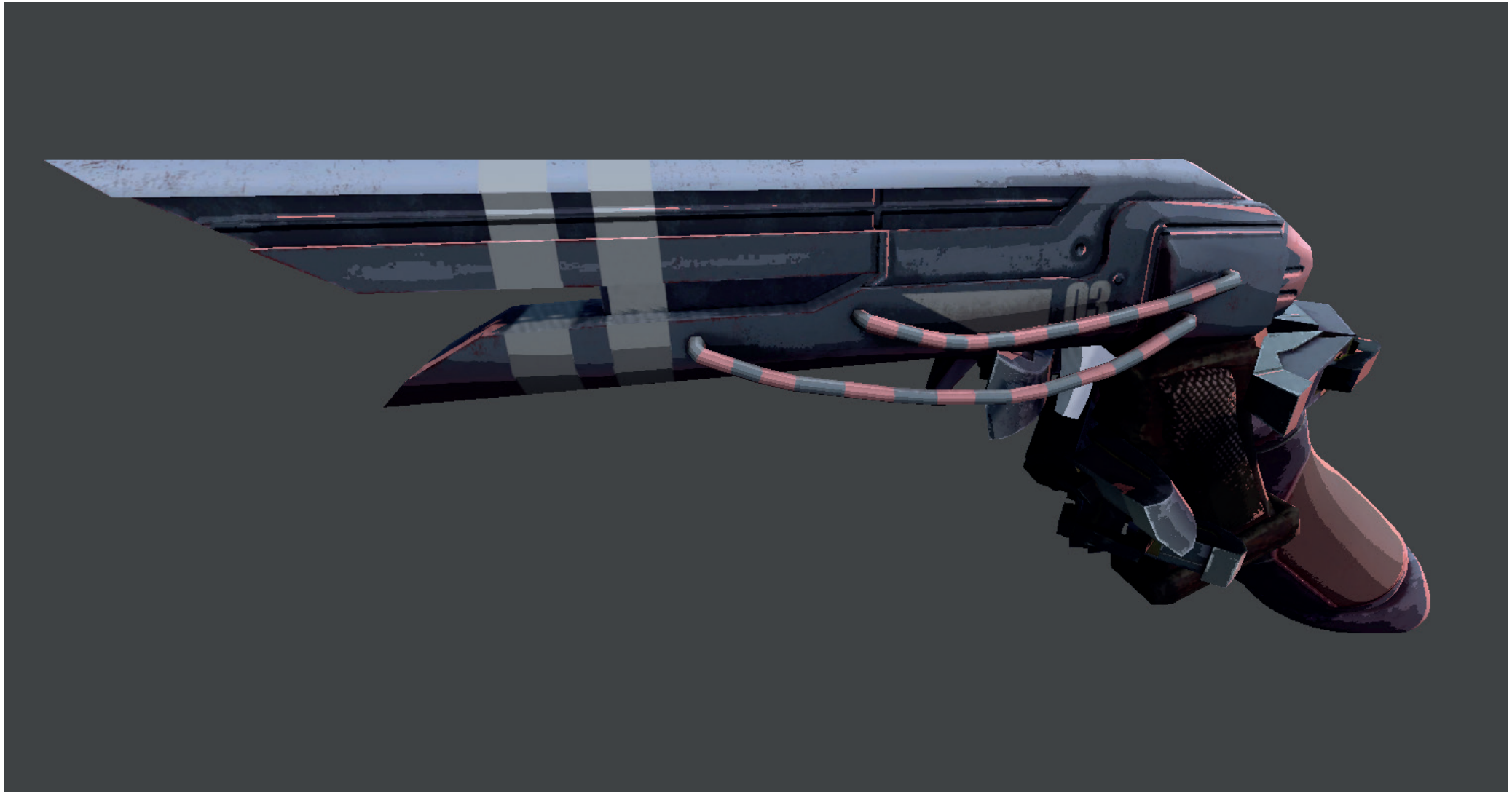
Le pistolet doit également être portable pour ne pas entraver les mouvements de l'avatar et doit pouvoir être manié à une main pour ne pas interférer avec la main gauche.

Il doit également tirer des projectiles en hitscan. Nous avons donc créés une arme de poing laser.

De plus, il ne nécessite pas d'être rechargé et tirer des balles énergétiques est donc une solution simple pour régler le problème de chargeur.






03

Pistolet

Comme dit dans la partie Game Design, le joueur a besoin d'information à l'écran indiquant des informations essentiels.

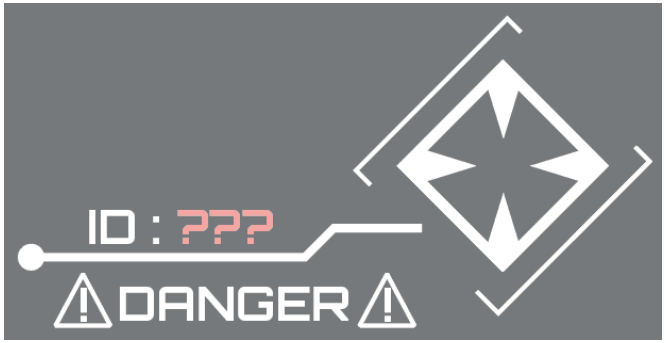
La vitesse est donc indiqué par une jauge qui monte et descend indiquant la vitesse exacte grâce à un texte.

Le temps du niveau est indiqué par un simple timer en petit au milieu de l'écran.

L'information de téléporteur lancé est signifié par le bras gauche mais aussi un indicateur autour du projectile lorsqu'il est dans l'angle de la camera.

Pour montrer la position les cibles, un indicateur semblable au précédent vient les entourer.

Bien que ressemblant à l'indicateur de téléporteur dans le visuel, dans le jeu les indicateur sont différents (page suivante)





L'indicateur du centre de l'écran, le nombre de charge et le temps avant que le téléporteur retourne en main sont tous indiqué au même endroit.

Résultat UI

LEVEL DESIGN

Type de niveaux

Dans la partie game design, nous avons déjà developpé ce point mais voici un redit.

Le type de niveau principale est un niveau couloir à terminer le plus rapidement possible. C'est le genre de niveau le plus simple à exploiter pour faire un jeu orienté speedrun.

Le but est d'aller d'un point A à un point B le plus rapidement possible.

Pour appuyer l'état critique, le joueur n'a qu'un temps donner pour aller à la fin du niveau avant que ce dernier soit envahi par le parasite occupant les lieux.

L'autre type de niveau possible consiste à devoir "nettoyer" une zone de tout les parasites présents dans cette dernière.

Nous avons priorisé la production de niveau "couloir".

Pour les couloirs, nos références sont CyberHook, Warstride Challenge, Stop Dead et Ghostrunner.

Ingrédient de Level Design

Pour rythmer les niveaux et créer de la diversité dans le gameplay et les comportement nous avons ajouté trois ingrédients permettant d'avancer dans les niveaux.

Le premier est une zone qui viens tuer l'avatar lorsqu'il collisionne avec. Cela vient créer des zones inaccessibles.

Le deuxième, déjà cité précédemment, est une cible sur laquelle tirer pour recharger un téléporteur.

Le dernier est une autre cible permettant de libérer une zone des zones de mort la bloquant. Cela permet de conditionner l'avancé de l'avatar dans le niveau.

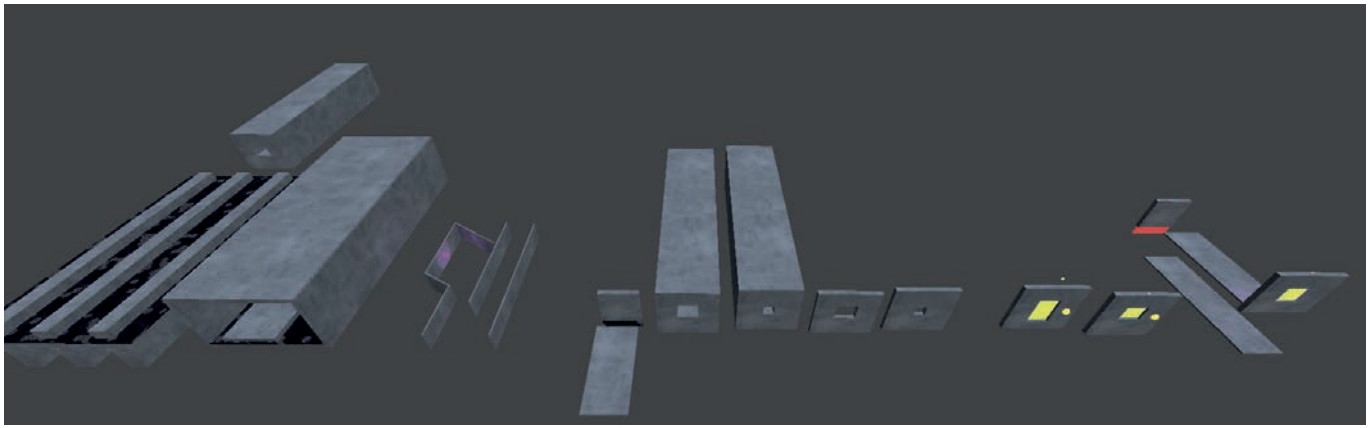
Processus de création

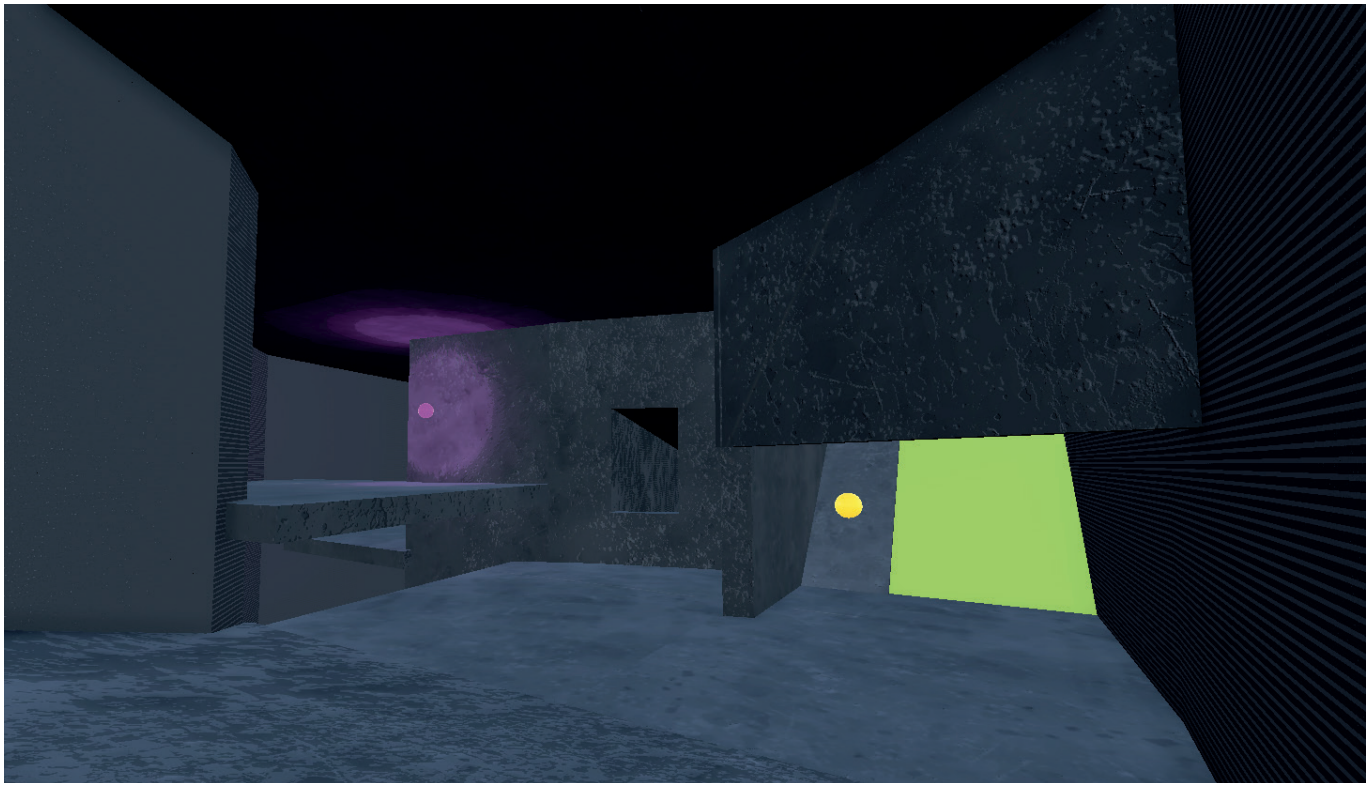
Comme dans un premier plan notre choix s'est porté sur des niveaux en forme de couloirs, il a fallu conceptualiser des modules montrant les utilisations de la mécanique de lancé. Le processus était le suivant :

- Placer des blocs classique de unity pour la circulatoire et l'utilisation des modules
- S'assurer de leur bon fonctionnement et enchaînement
- Créer un niveau en mélangeant les modules

Le but dans la constructions des niveaux était de créer plusieurs chemins explorant chacun une utilisation différente des mécaniques du jeu.

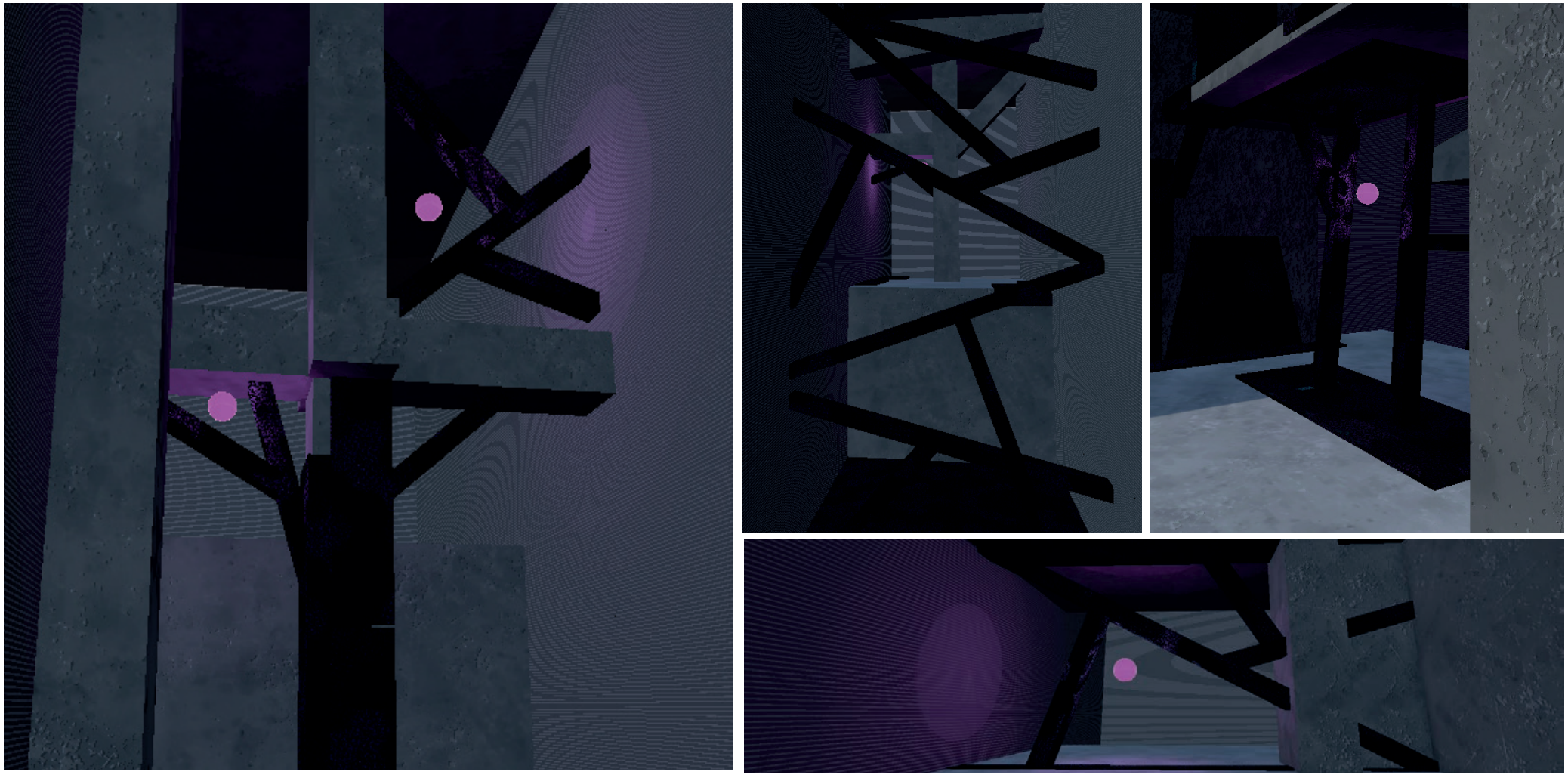
Ci contre, on voit qu'un chemin challenge l'enchainement des actions de téléportation et de tir (gauche), qu'un autre challenge la précision de lancé du téléporteur et la glissade (milieu) et un dernier challenge le tire et les reflèxes (derrière la porte) (droite).





Une fois les niveaux réalisé grâce aux modules, il a fallut les habiller. Cela à permis de simplifier des zones en ouvrant plus l'espace ou de complexifier les challenges en fermant et saturant les espaces de zone de morts par exemple.

Cela a permis de voir une première ébauche de niveau et des circulatoire de ces derniers.



Modélisation

Modélisation
Toutes les étapes précédentes ne sont que placeholder. Pour simplifier la modélisation, les niveaux ont été exporté de Unity en FBX puis placé dans Blender. Cela permet lors de la modélisation de respecter les tailles des niveaux et modules.

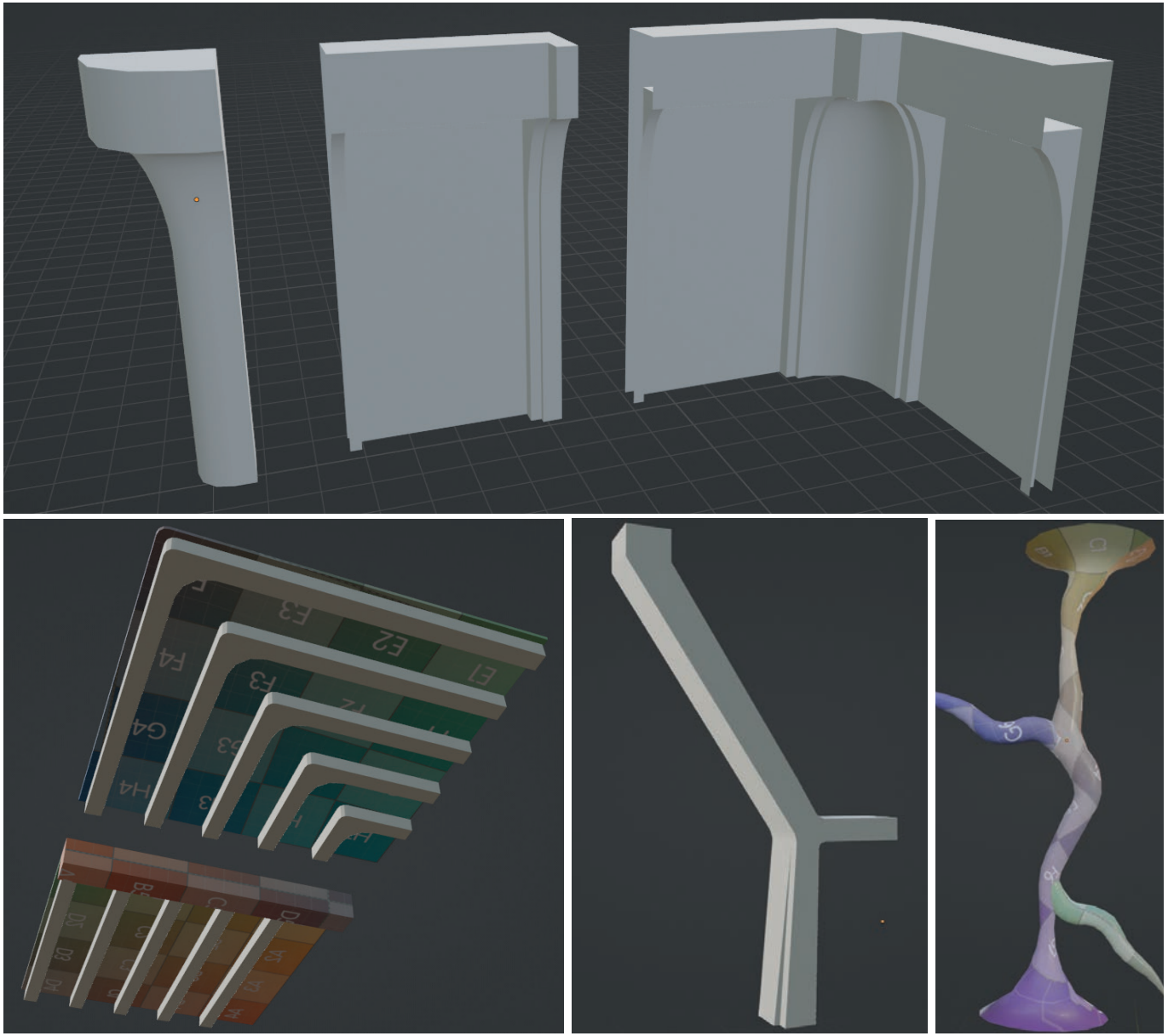
Faire de la modélisations modulaire vient créer beaucoup de problématiques et de réflexions complexes à mettre en place en peu de temps.

Le workflow à été le suivant:

- Trouver des références visuels brutalistes pour styliser la modélisation
- Modéliser des modules
- Les liers entre eux

Au global, les modules les plus importants sont au nombre de trois. Les murs, les plateformes et les piliers.

Grace au style brutaliste, le reste peut être composé de cubes coupé et déformé facilement et rapidement.

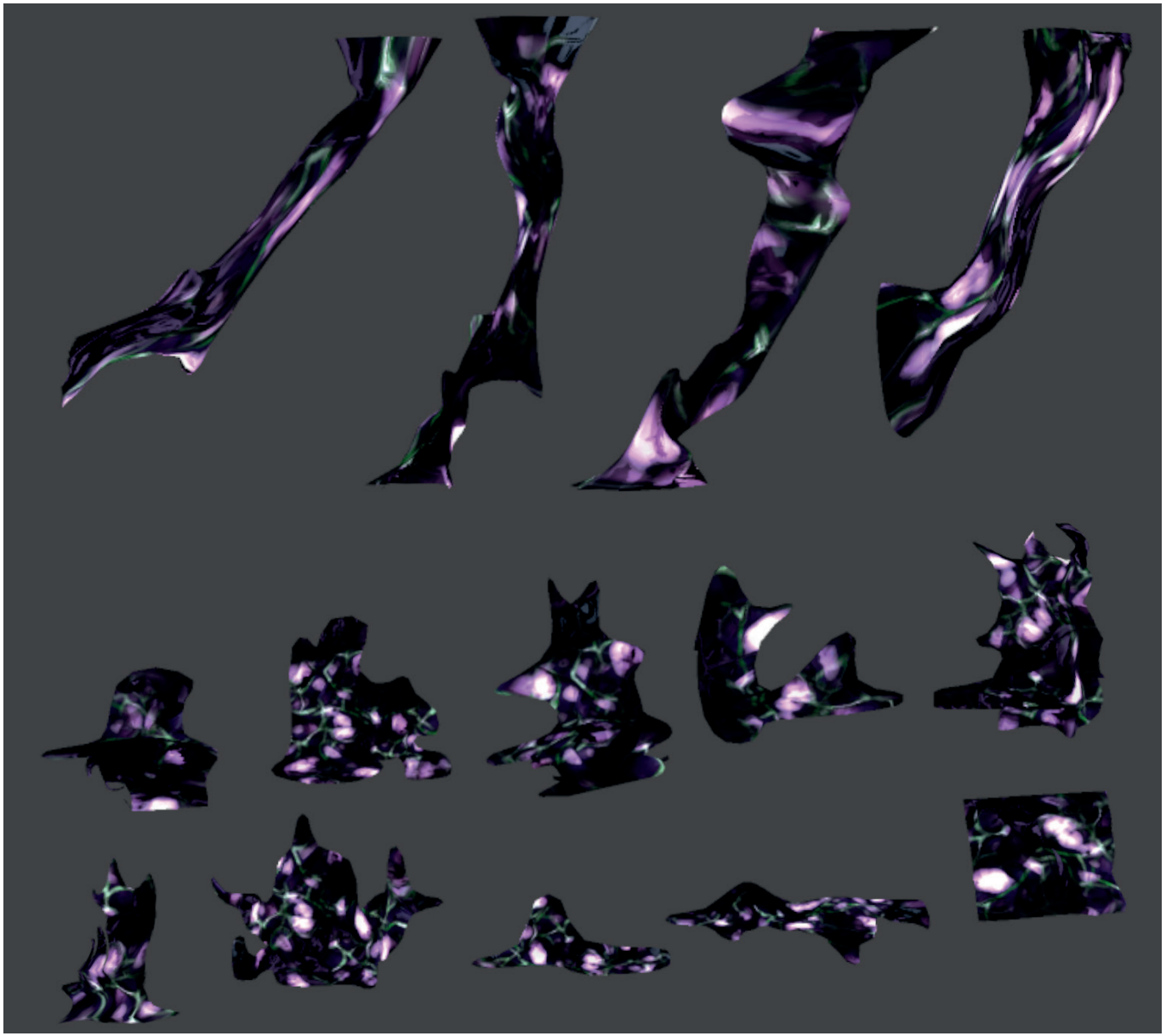


Ci contre, les modules de parasites. Ils ont des formes différentes permettant d'être mélangé entre eux pour formé des zones de formes et tailles différentes.

Les bras sont soit reliés à :

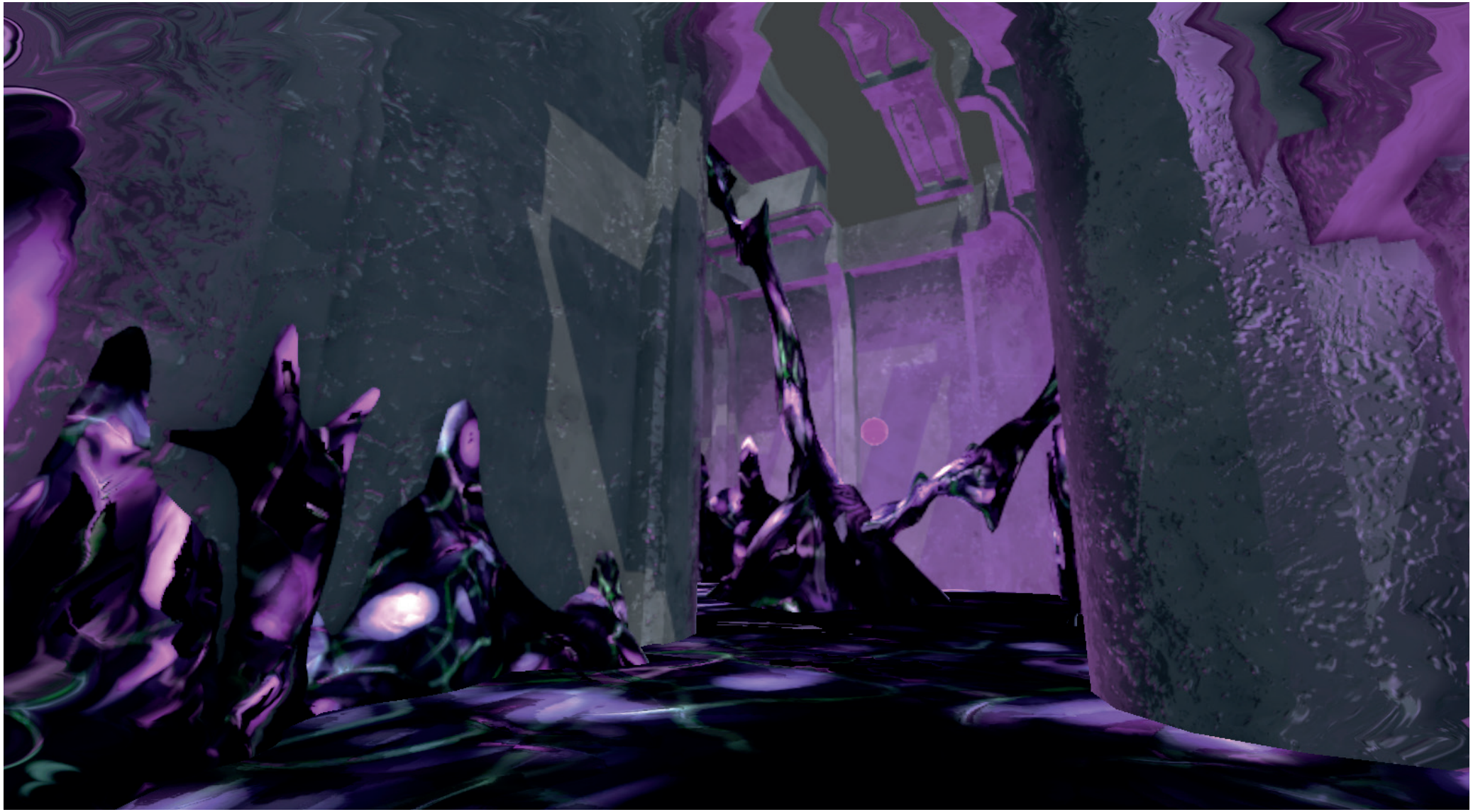
- Un sol et un plafond
- Un sol et un mur
- Un sol et un plafond

Ces deux options permettent de varier les directions des bras facilement.

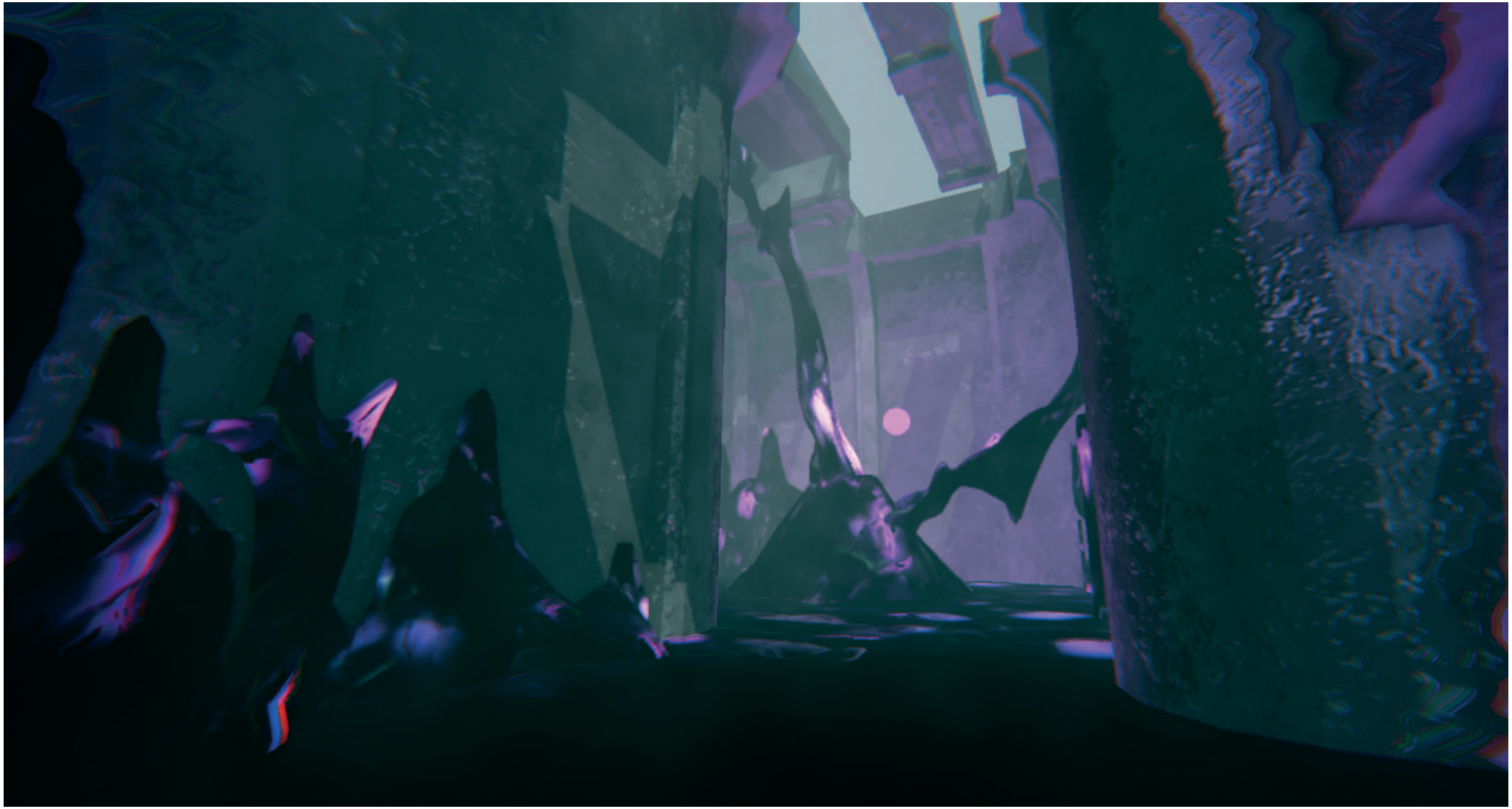


Résultat

En déformant certains modules et en implémentant le tout dans Unity avec les textures et shaders d'environnements, le résultat est le suivant.



Pour améliorer la lisibilité, nous avons ajouté du post effect permettant de mieux discerner les distance et la profondeur. De plus, cela permet d'augmenter l'immersion.



SOUND DESIGN

Shift est un Fast FPS avec une direction artistique brutaliste et noisy. Le gameplay est donc dynamique et beaucoup d'action peuvent être réalisé en même temps.

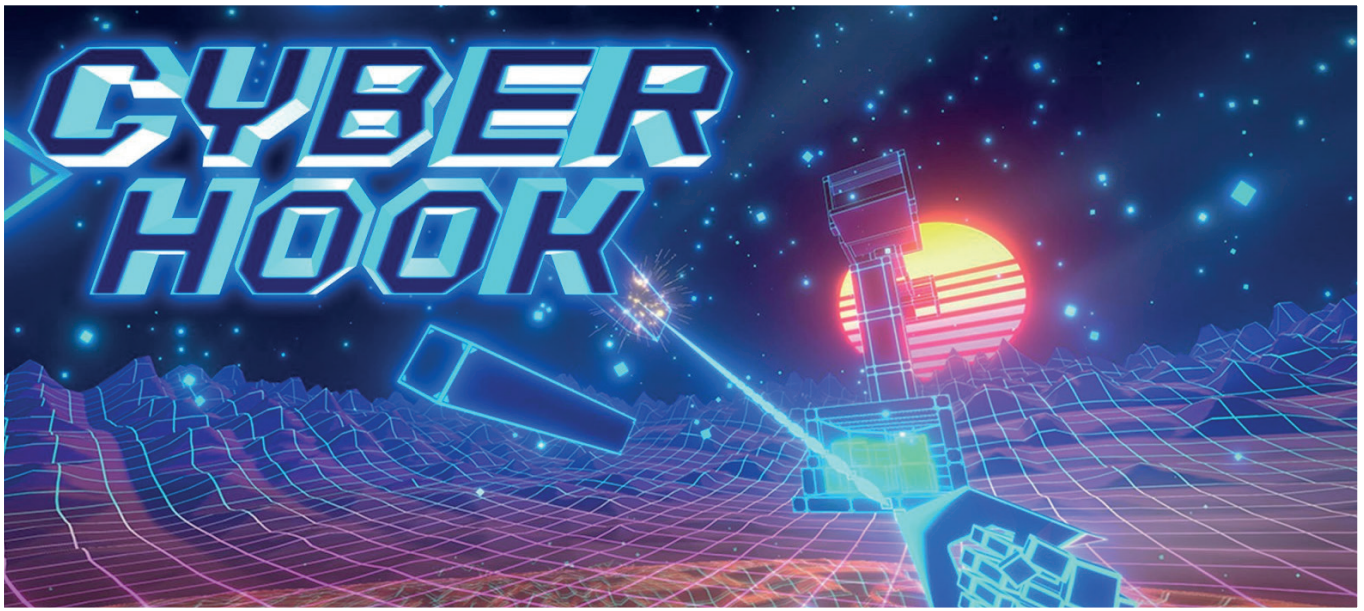
Notre avatar est un cyborg (mi humain, mi machine) pouvant se déplacer, se téléporter et tirer avec une arme.

De par notre direction artistique «sale», les effets sonores doivent être en lien avec les visuels. Le but est donc de rendre les sons compréhensible mais noisy.

Nos inspirations sont Ultrakill, qui colle parfaitement avec ce coté sale, noisy de la direction artistique et CyberHook qui est plus une inspiration pour les sons de vitesse en lien avec les visuels de nos speedlines.

L'enjeu majeur est de rendre la navigation compréhensible grâce au son. La vitesse à laquelle l'avatar se déplace et ses interactions avec l'environnement sont clés.

De même pour le téléporteur. Il faut pouvoir comprendre et différencier chaque interactions pour ne pas devoir se concentrer sur celle ci autrement que par le son.



Intentions et Inspirations

L'event List est organisé de sorte a différencier les différentes catégories de son et ce qu'elles contiennent.
Prenons la catégorie Character ici. Elle est composé de sous catégories

- Mouvements
- Tir
- Téléporteur

Ces sous catégories sont aussi composé de sous catégories.

Par exemple pour Mouvements, one retrouve ces sous catégories

- Général
- Walk / Sprint / Crouch
- Slide

Cela permet de bien segmenter les parties de l'event list. On retrouve ensuite pour chaque son, son état de complétion, une rapide description, et sa condition de déclenchement.

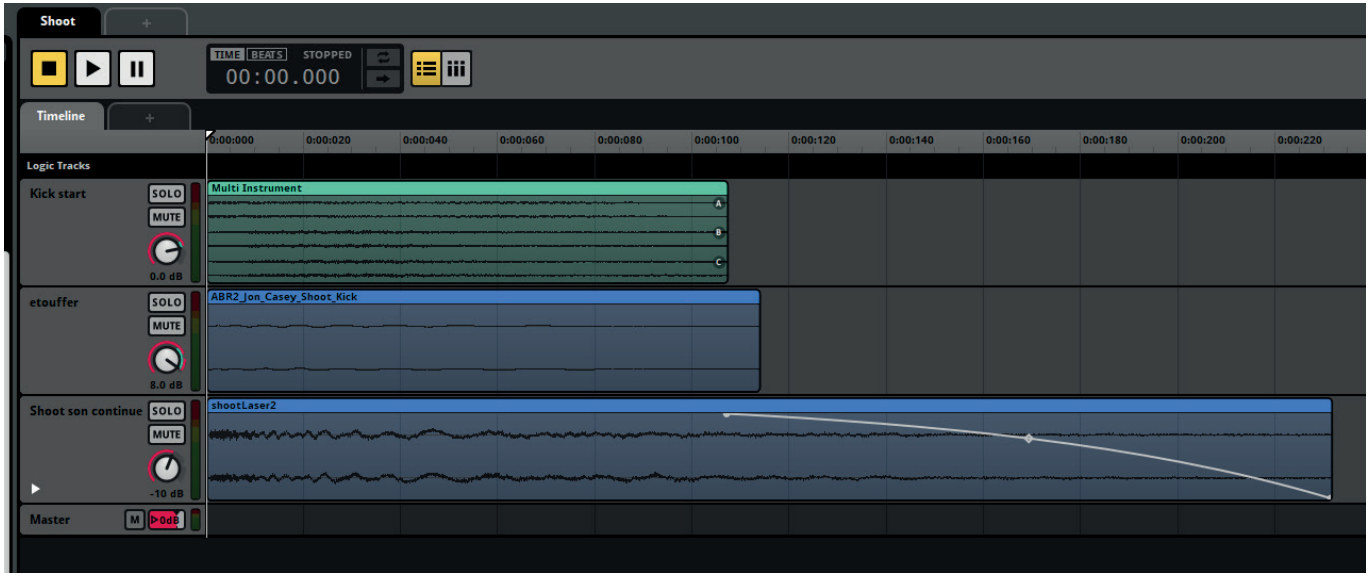
Class	Group	Priority	Status	Fmod Event name	Sound description	Caption/Trigger condition (human readable)			Event type	Loop
Character										
	Movement									
		General								
		P0	Done	Speed	Son de vent, volume et evolution relatif à la speed	Sound linked to player speed			2D	Yes
		P2	Not Started		//	The player is falling			2D	Yes
		P1	Done	Jump	//	The player press the jump button while grounded			2D	No
		P2	Not Started		//	The player stand up from crouch/slide			2D	No
		P2	Not Started		//	The player press the crouch/slide button while grounded			2D	No
		P2	Placeholder	TounchGround	//	The player touch gound after is falling			2D	No
		Walk/Sprint/Crouch								
		P0	Not Started		//	The player is walking/crouching while grounded			2D	No
		P1	Not Started		Faster Walk Sound	The player is sprinting while grounded			2D	No
		P0	Not Started		//	The player reaches the ground after being in the air in walk/sprint/crouch mode			2D	No
		Slide								
		P0	Placeholder	Slide	volume relatif à la speed	The player is sliding while grounded			2D	Yes
		P2	Not Started		//	The player reaches the ground after being in the air in slide mode			2D	No

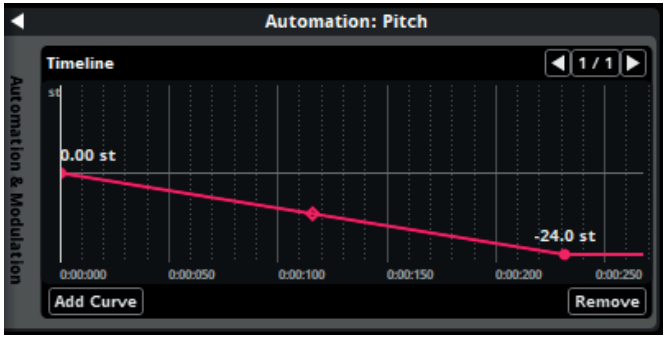
Amené à être utilisé afin de recharger notre mécanique de déplacement (tir sur une cible pour recharger notre téléporteur) nous avons voulu décomposer le son de notre «laser» en 2 parties :

- Le kick, la détonation principale au moment du tir
- Un son de glissement sur la continuité du tir.

Le glissement à un pitch qui réduit rapidement pour donner une sensation de mouvement et de vitesse aux tires.
Une reverb accompagne le tout pour donner un aspect plus «futuriste» au tir.
Le son étouffé est principalement là pour rendre la transition entre le lick et le glissement plus agréable et smooth.

Un random sur les pitchs et les volumes pour ajouter plus de variation à la détonation qui est un son qui peut être entendu plusieurs fois d'affilé.



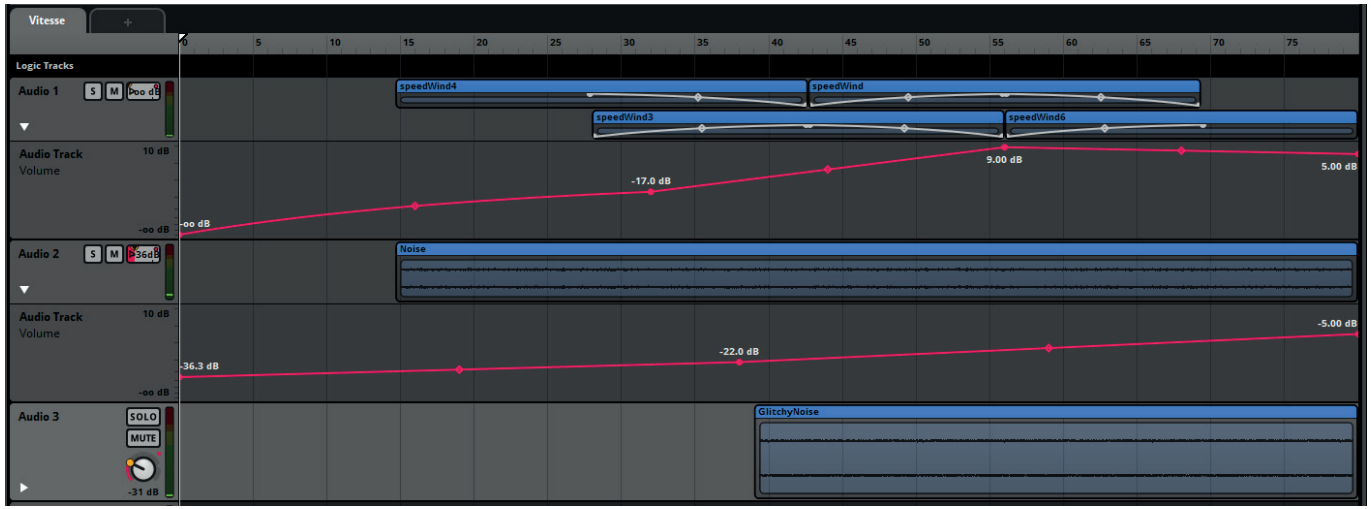


Un des indicateurs majeurs d'une prise de vitesse est un son de vent évoluant en fonction de celle-ci.

En partant de ce constat, notre idée était ici d'avoir une plus ou moins grande quantité de vent en fonction de la vitesse que l'on récupère du joueur grâce à la variable "Vitesse".

Il y a 5 sons de vent plus ou moins intense qui fade les un dans les autres pour que les transitions soient fluides.
La track est régie par un automation du volume pour que plus l'avatar se déplace rapidement, plus le son de vent est fort.

On commence à entendre le vent à partir de d'une vitesse de 35, ce qui correspond à la vitesse de course de l'avatar en jeu. En dessous, l'avatar marche et le bruit du vent n'est pas présent.



Nous avons ajouté un bruit blanc léger pour correspondre davantage à notre esthétique visuel du jeu. Cela ajoute un côté plus sale, moins de vent naturel. Sur la fin, à une vitesse élevée, nous avons ajouté un bruit qui vient distordre très légèrement le son du vent.
L'idée est de coller avec le visuel de nos speedlines qui sont distordues. Sur le vent et le bruit blanc, un chorus a été ajouté pour rendre le son encore plus sale et tremblant.

Enfin, sur le vent, un effet "fangler" a été placé avec une faible fréquence et une faible profondeur pour dénaturer le son d'origine. Comme la vitesse peut varier entre deux valeurs éloignées, un seek speed assez rapide a été ajouté pour que les transitions entre les vitesses ne soient pas trop brutales et sèches.

Un des sons les plus présent est celui de la mécanique principale, celle de téléportation. L'action de téléportation dans notre projet peut-être décomposée en deux parties :

- La destruction d'un hologramme, représentant le téléporteur, dans la main de l'avatar.
- La téléportation en elle-même.

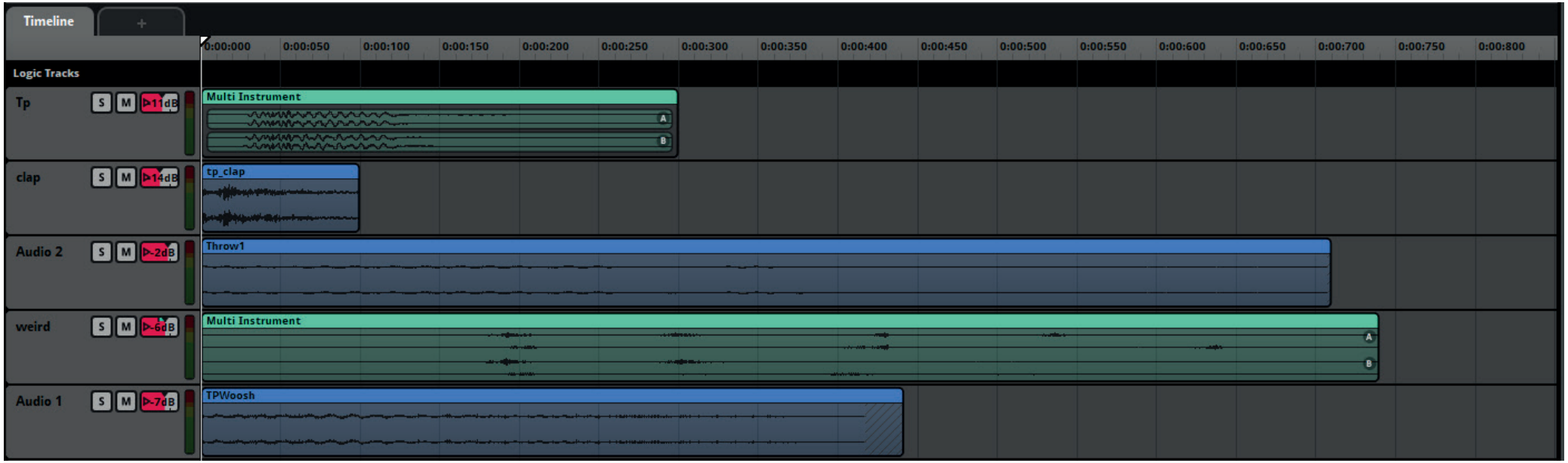
Pour la première partie, un son de clap de main métallique, en cohérence avec le visuel, maintenu à un faible volume, du fait du potentiel irritant d'un clap et de la fréquence d'activation de l'event dans le jeu.

Pour la seconde partie, l'objectif est de créer une sensation de «bulle», du fait du déplacement instantanée. Pour ce faire, une basse , très forte et très saturée, pour le côté étouffant et assourdissant auquel nous avons ajoutée une reverb late, pour préserver la phase des fréquences basses au maximum, tout en donnant du volume au son.

Nous avons ajouté un bruit métallique de plus haute fréquence avec une reverb courte, mais large, qui sert à donner un aspect technologique au son, mais aussi à contrebalancer la basse.

L'ajout d'un son étrange à répétition rapide pour apporter de la variété et un côté ésotérique à la technologie, inspiré du fait qu'elle puise son énergie dans un organisme alien.

Enfin, un léger son de distorsion pour justifier le déplacement de l'avatar dans l'espace.



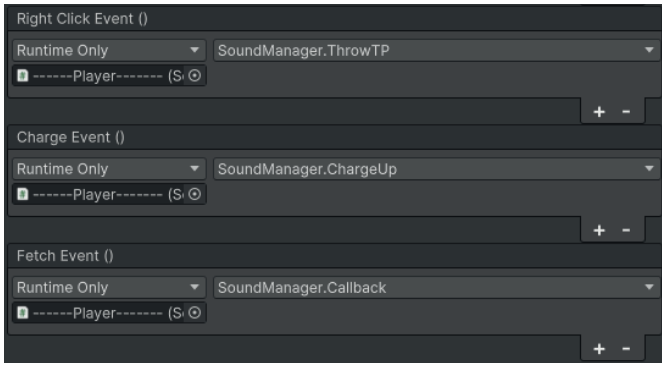
Intégration

Pour l'intégration du son dans le code, nous avons créé un script "SoundManager" regroupant la majorité de nos sons sous forme de fonction. Ces fonctions seront ensuite appelées par des événements dans d'autres scripts, selon certaines conditions. (cf image, le script du lancer du TP par exemple).

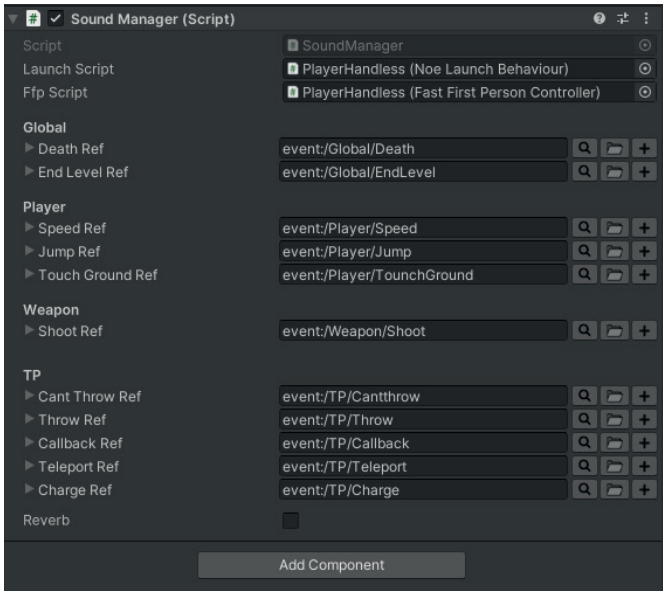
Cela permet de centraliser la gestion audio, d'avoir une meilleure organisation et de simplifier l'appel des sons, n'étant pas dispersé à travers tous nos scripts, afin de réduire le risque d'erreurs.

De plus, en renseignant les chemins des événements FMOD directement dans l'inspecteur à l'aide d'objets de type EventReference, nous pouvons sélectionner directement des événements existants après une build FMOD.

Cela évite les erreurs de saisie dans le code, facilite l'accès aux événements souhaités, et permet de voir immédiatement s'il y a un problème de connexion avec l'événement, car un message d'erreur apparaît sous la ligne concernée en cas de problème.



```
private void FetchInput(InputAction.CallbackContext context)
{
    if (context.performed)
    {
        if (!_inHand)
        {
            fetchEvent.Invoke();
            // callback pas dans Fetch car Fetch utilisé aussi pour la destruction
            Fetch();
        }
    }
}
```



```
public void Callback()
{
    FMODUnity.RuntimeManager.PlayOneShot(callbackRef);
}

0 références
public void Teleport()
{
    FMODUnity.RuntimeManager.PlayOneShot(teleportRef);
}

0 références
public void ChargeUp()
{
    FMODUnity.RuntimeManager.PlayOneShot(chargeRef);
}

#endregion

#region Player
1 référence
public void Speed()
{
    _speedEvent.setParameterByName("Vitesse", ffpScript.getFullSpeed
}
```

Notre intention pour la musique était d'appuyer sur l'ambiance sombre et oppressante incitée par les visuels. Nous souhaitions également qu'elle soit rythmée afin de maintenir la concentration du joueur sur son gameplay. Pour son évolution, elle gagne en intensité en fonction de la progression du joueur dans le niveau, agissant en tant que guide mais aussi en tant que motivation et pression en accord avec nos intentions de frénésie. Cette dernière est donc grandement renforcé en fin de niveau.

Nous avons donc composé une musique techno évolutive, à base de synthétiseurs graveleux, noisy. Il y a 4 boucles différentes et la transition entre celle-ci est progressive au travers du niveau :

1. Dans la première, il y a simplement la basse fondamentale du morceau.
2. On vient ajouter deux kick par mesure, des percussions ainsi qu'un synthé lead pour introduire un rythme et une mélodie au morceau.
3. Le kick passe sur chaque temps pour accélérer le rythme, on ajoute de nouvelles percussions et la basse fondamentale change de motif.
4. Identique à la troisième à l'exception de l'ajout d'un son aigu tous les deux temps et surtout du mix qui est beaucoup plus distordu et met plus en avant le kick, pour donner un aspect d'urgence absolue.