Gameplay programmer

Portfolio

Paul Hugy

2025

Paul_Hugy

Salut !

Mon nom est Paul Hugy, et je suis un Gameplay Programmer.

Je suis spécialisé dans la création d’architectures de jeu solides et flexibles, permettant de développer des systèmes de gameplay puissants tout en restant faciles à faire évoluer.
Mon approche est avant tout centrée sur les besoins du jeu, tout en fournissant aux game designers et artistes les outils et les workflows nécessaires pour exprimer pleinement leur vision dans le moteur.

Passionné et investi, j’ai à cœur de concevoir des expériences de jeu réactives, engageantes et soignées.

Je suis impatient de mettre mes compétences au service de vos projets ambitieux.

J’espère que vous prendrez plaisir à explorer mon portfolio !

Retrouvez moi sur :

06 99 29 24 43

paul@hugy.fr

linkedin.com/in/paul-hugy/

paul-hugy.itch.io/

artstation.com/paul_hugy

Section
Projets Complets

Jeux développés sur plusieurs semaines, seul ou en équipe. Ces projets mettent en avant des systèmes de gameplay aboutis, une structuration propre du code et une attention particulière à la finition et à l'expérience utilisateur.

Kraken Project

Jeu d’arcade en 3D développé sous Unity
2024

Language :
C#

Rôles :
Chargé de toute la programmation ainsi que du Game Design

Durée :
Projet sur 3 semaines

Equipe :
Equipe de 2 personnes, réalisé en collaboration avec Bastien Canto

---

Concept :
Kraken Project est un jeu de flipper 3D original au thème pirate, dans lequel vous contrôlez une balle en forme de crâne à travers trois environnements différents : le pont du bateau, la cale de rangement et la prison.

Votre objectif est de marquer un maximum de points en activant des rampes, déclenchant des plaques de pression et améliorant des tonneaux interactifs en naviguant entre les salles, tout en évitant que la balle ne quitte le plateau.

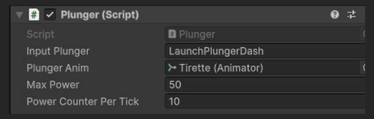
Particularité du gameplay : Toutes les quelques secondes, vous disposez d’une capacité spéciale, le Dash Temporel, qui ralentit brièvement le temps et vous propulse précisément dans la direction souhaitée, ajoutant ainsi une dimension stratégique et dynamique unique au jeu.

Projet Complet

Mécaniques inspirées du flipper classique :

Plungeur (Lanceur) :

Système de lancement de la balle en début de partie avec gestion de la force (input pressé/relâché).

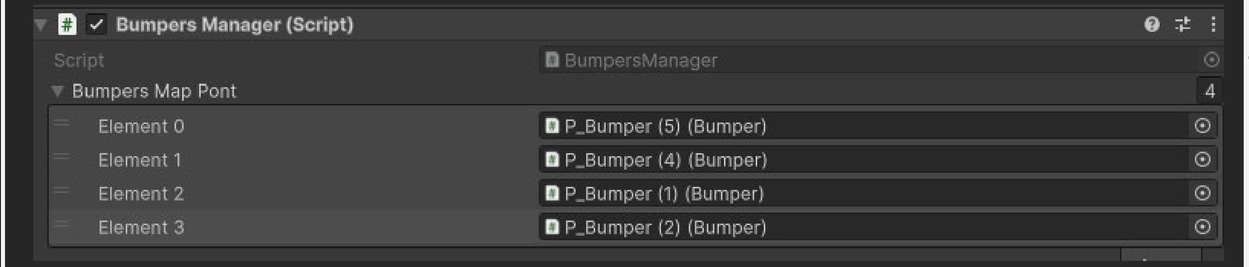


Bumpers :

Objets réactifs répartis dans les trois scènes, renvoient la balle dynamiquement, donnent du score a l'impact.

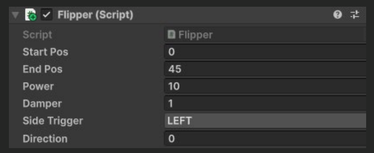
À chaque impact de la balle, leur état (0 ou 1) est mis à jour. Lorsque tous les bumpers d'une salle sont activés simultanément, un état d'amélioration est déclenché, augmentant leur valeur en score et modifiant leur feedback visuel/sonore.

Le système repose sur un gestionnaire de salle qui vérifie l'état global des bumpers via un tableau booléen, déclenchant l'upgrade dès que toutes les valeurs passent à 1.



Flippers :

Palettes réalisées grâce à un hinge joint et un rigidbody pour avoir un temps de réaction fluide et responsive sans ghosting.



Slingshots :

Contrairement aux bumpers, ils n'ont qu'un seul sens de propulsion et servent à guider ou piéger la balle selon le level design.

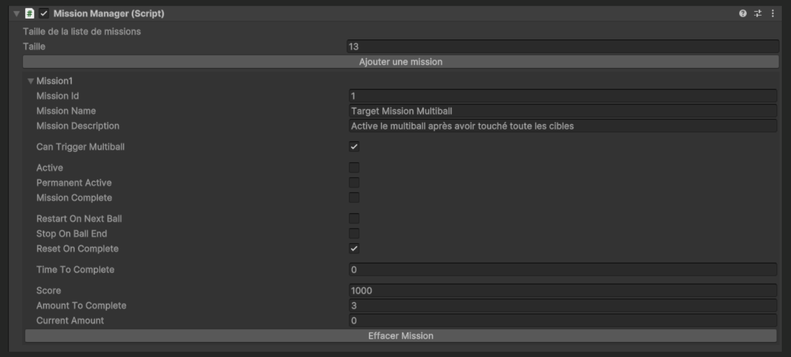
Kraken Project

Différentes Missions à réaliser pour gagner du score :

Mission Manager :

Le cœur du système repose sur un Mission Manager global, capable de :

- Gérer une liste de missions actives
- Suivre leur progression
- Déclencher des événements spéciaux (ex: Multiball)
- Gérer les resets en cas d'échec ou de perte de vie

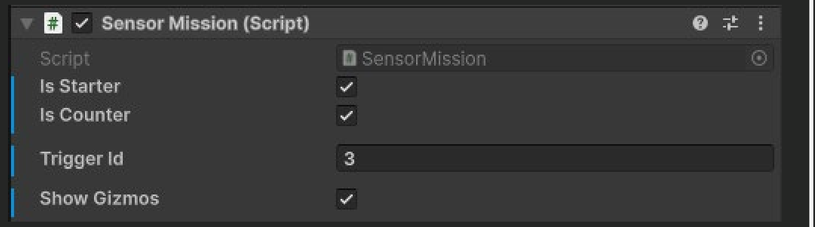


Chaque mission est entièrement configurable dans l'inspecteur (score, conditions, timer, multi-event, etc.), ce qui permet une grande flexibilité pour designer les objectifs.

Interaction via Sensors :

Les missions sont déclenchées ou progressées via des capteurs placés dans le monde.

Chaque sensor peut :
- Lancer une mision
- Compter une progression



Les sensors sont associé à une mission précise via un triggerID répértorié dans le mission manager.

Il utilisent OnTriggerEnter ou OnCollisionEnter selon les besoins, et sont visualisables dans la scène via Gizmos, facilitant le placement et le debug.

Résultat : une mission peut être démarrée en touchant une cible, et progresser automatiquement à chaque action liée (plaque, rampe, bumper...).

Cibles / Plaques de pression :

Plaques de pression activées au passage de la balle.
Elles donnent un score immédiat, mais peuvent aussi être reliées à des objectifs plus complexes via le système de missions.

Multiball :

Chaque salle contient une charge de multiball, représentée physiquement par un tas de bombes.

Une fois en collision, le capteur incrémente une mission dédiée (ID unique) via le MissionManager. Une fois les 3 charges récupérées (une par salle), la mission est complétée automatiquement et le multiball est déclenché.

Effets du Multiball :
- 2 nouvelles balles sont générées pour atteindre un total de 3.
- Les passages entre salles sont bloqués temporairement.
- Des particules de fumée et objets visuels indiquent la fermeture.
- Le joueur peut marquer un maximum de points.

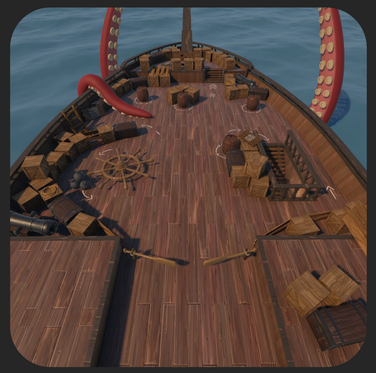
Lorsque 1 seule balle reste, le multiball prend fin et les passages sont à nouveau accessibles.

Kraken Project

Un flipper multi-plateaux :

Passage de Salles :

Trois environnements connectés (Pont, Cale, Prison).
Vous pouvez passez de chaque salle au autres en passant par différents point dans la salle, changeant ainsi dynamiquement le point de vue caméra sur le plateau.



Pont

Cale

Prison

Lorsque la balle atteint un trigger de sortie elle est detruite par le destructeur et est téléportée au crafteur lié à celui-ci afin de rejoindre la salle connectée.

Tp Ball Destructeur (Script)

Script	TpBallDestructeur
Tp Ball Crafteur	P_Crafteur (Tp Ball Crafteur)
Room Tp	CALLE

Tp Ball Crafteur (Script)

Script	TpBallCrafteur
Ball Prefab	P_Ball
Tp Crafteur	P_Crafteur (Transform)
Ball Direction	Direction (Transform)
Power	500

Chaque salle possède une position de caméra dédiée, définie dans le CameraManager.
Lorsque la balle change de salle, la caméra effectue une transition vers la nouvelle position.
Le joueur ne perd jamais de vue la balle, la caméra reste fluide, centrée sur l'action, ce qui améliore considérablement la lisibilité du gameplay.

Camera Manager (Script)

Script	CameraManager				
Pont Vector	X	0	Y	3.64	Z 4.409
Cale Vector	X	0	Y	-5.772	Z 3.71
Prison Vector	X	0	Y	-13.565	Z 4.25
Capitaine Vector	X	-10	Y	5	Z 5
Max Distance	7				
Speed Go To Ball Cam	2				
Speed Back Cam	4				
Ball To Camera Zoom	6				
Active Follow Camera	✓				
Ball	None (Game Object)				
Room Active	PONT				

Mécanique Signature :

Dash Temporel :

Toutes les X secondes, le joueur peut ralentir le temps et choisir une direction pour propulser la balle.

Ce dash ajoute une couche stratégique et intervient comme un "joker" pour sauver une situation critique ou viser une cible précise.

Dash Manager (Script)

Script	DashManager
Dash Variable	
Can Dash	✓
Dash Rdy	✓
Dash Force	20
Dash Cd	15
Max Prep Time	4

Kraken Project

Gameplay Loop :

1 - Spawn de balle & début de partie

Le joueur débute avec 3 balles. (Vies)

Une balle est instanciée depuis un point de spawn, avec reset des missions et du cooldown du dash. Elle est toujours instanciée au niveau du pont sur le point du Plunger.

Le système vérifie que le joueur n'a pas déjà une balle active avant d'en créer une nouvelle.

2 - Jouer / compléter des missions / survivre

Tant qu'il y a au moins 1 balle sur le plateau, la partie continue.

Les missions peuvent être complétées (cibles, rampes...) et les joueurs accumulent des points.

Si certaines conditions sont remplies, le mode multiball se déclenche.

3 - Perte de balle

Si la balle sort du terrain (LoseArea), elle est détruite.

Si c'est une balle normale :

- Le nombre de balles restantes est décrémenté.
- Une nouvelle balle est automatiquement créée si le joueur en a encore.
- Sinon, c'est la fin de la partie (transition de scène vers écran de fin).

Si c'est une balle multiball, elle est simplement détruite (sans impact sur les vies).

4 - Fin de Partie & Leaderboard Online

Affichage des meilleurs scores (Top N) avec pseudo et score.

Saisie d'un nom de joueur via TMP_InputField.

Upload automatique du score après validation.

Mise à jour automatique du classement après ajout d'un score.

Communication via la clé publique sécurisée de l'API.
Le leaderboard utilise Leaderboard Creator, une solution open-source hébergée sur GitHub.

Lien de l'outil leaderboard creator par Danial Jumagaliyev :
https://danqzq.itch.io/leaderboard-creator

Essayez le jeu :

https://bastini.itch.io/kraken-project

Mot de la fin :

Kraken Project a été mon tout premier projet complet.
Étant seul responsable du game design et du développement, j'ai pu concevoir ma propre architecture gameplay et apprendre à organiser proprement mes systèmes (physique, scoring, multiball, dash...), tout en les faisant interagir sans bugs ni conflits.

C'était la première fois que je concevais un jeu de A à Z: gameplay, UI, menu, boucle de jeu, feedbacks, son et même build final.

Ce projet a aussi été un excellent exercice de game feel: transformer un concept aussi simple que le flipper en une expérience fluide, lisible et agréable à jouer sur PC, en 3D, demande bien plus de finesse qu'il n'y paraît.

J'espère que ce projet vous aura plu !

Paul_Hugy

Kraken Project

Bolt & Nut :
Ferailleurs de l'espace

Shoot'em up 3D développé sous Unity
2024

Language :
C#

Rôles :
Chef de projet
Chargé de toute la programmation ainsi que de l'integration
Participation au game design et level design

Durée :
Projet sur 6 semaines

Equipe :
Equipe de 4 personnes

Concept :
Shoot 'Em Up en vue à la troisième personne avec défilement vertical.
Dans ce jeu, les joueurs doivent naviguer à travers trois environnements différents, en survivant à des vagues de vaisseaux ennemis.
Ils devront utiliser les ressources récupérées sur les ennemis pour gagner en puissance, atteindre et vaincre le boss final, et gravir les échelons pour devenir le meilleur récupérateur.

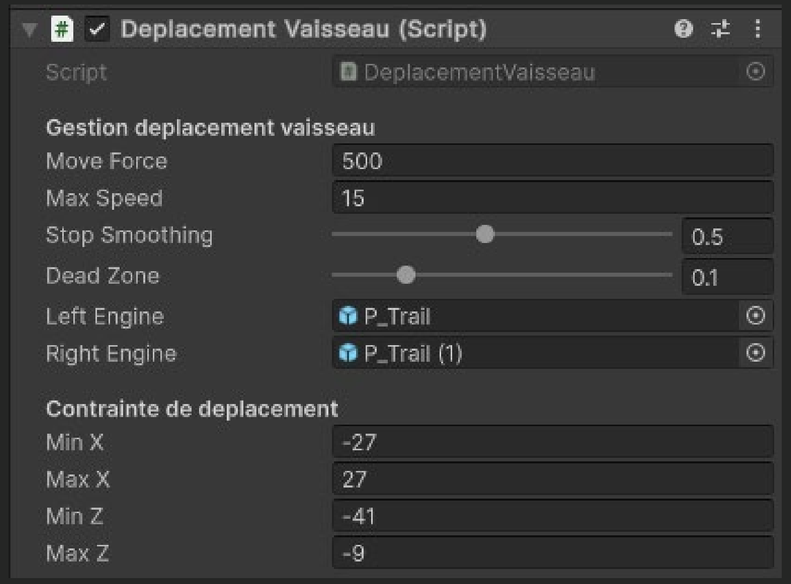
Projet Complet

Personnage Jouable, le vaisseau :

Déplacement :

J'ai développé un système complet de déplacement du vaisseau en 3D avec une approche orientée game feel : fluide, limité, lisible et agréable à contrôler.

- Physique contrôlée (Rigidbody) pour une sensation de mouvement progressive et naturelle.
- Vitesse maximale.
- Zone de déplacement limitée dans un espace défini (X / Z).
- Lissage de l'arrêt (SmoothStop).
- Animations dynamiques : Le vaisseau s'incline selon l'input et moteurs latéraux activés/ désactivés selon l'état de mouvement.



Système de tir :

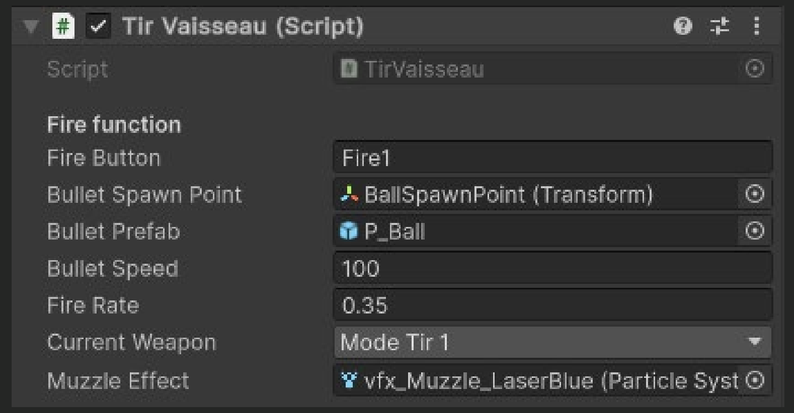
Le tir est au cœur du gameplay de Bolt & Nut.
J'ai conçu un système de tir évolutif en C# intégrant plusieurs modes d'attaque, un système de progression ainsi que des effets visuels.

Ces modes de tir sont utilisé dans le Game Manager en fonction de l'experience accumulé par le joueur.

Mode de tir 2

Mode de tir 5

Le vaisseau dispose de 6 modes de tir progressifs, débloqués automatiquement avec l'XP.
Chaque mode ajoute de la puissance ou de la dispersion, avec un pattern codé à la main.



Ultimate :

L'ultimate est une attaque de zone surpuissante, disponilble à partir du niveau 2.
Cette attaque est unique par niveau et agit comme un "panic button" lorsque le joueur se sent dépassé par les vagues d'ennemis

J'ai créé un VFX marquant qui apparaît au centre du joueur accompagné d'une secousse caméra, le tout rendant son utilisation satisfaisante.

Dégats continus en zone sur :

- Tout les ennemis (tick damage).
- Tous les projectiles ennemis (clear bullet hell).
- Les boss : dégats en % en fonction de leur vie max.

Bouclier :

J'ai mis en place un bouclier unique qui protège le joueur contre un coup et se recharge automatiquement après un délai.

Apparition et destruction du bouclier sont accompagnées de Effets visuels (VFX), Sons dédiés, Fragments 3D en cas de break.

Bolt & Nut

Les ennemis communs :

Classe de base - Héritage :

J'ai développé un système modulaire et extensible d'ennemis basé sur une classe mère Enemy.

Ce système permet une intégration rapide de nouveaux types, très facilement et rapidement tout en garantissant cohérence, feedback et récompense.

Méthodes virtual / abstract permettent à chaque type d'ennemi de surcharger ses déplacements et comportements spécifiques (Move(), Fire(), Die()).

Fonctions générales intégrées :

- Points de vie / barre de vie dynamique.
- Système de tir configurable (canShoot, fireRate, bulletPrefab, etc.).
- Comportement de mort et explosion (son, FX, fragment).
- Distribution d'XP via orbes dropées.
- Système de flash visuel à l'impact (_HitPow shader).
- Rotation UI barre de vie face caméra.
- Récompenses (score / XP) configurables individuellement.

```
public abstract class Enemy : MonoBehaviour
{
    [Header("Characteristics")]
    [SerializeField] protected int maxHealth = 100;  Changed in 0+ assets
    [SerializeField] protected float speed = 5f;  Changed in 0+ assets
    [SerializeField] protected int damage = 10;  Changed in 0+ assets
    [SerializeField] protected int damageCol = 10;  Changed in 0+ assets
    [SerializeField] int currentHealth;  Changed in 0+ assets
    protected bool isDead = false;
    public bool canShoot = false;  Changed in 0+ assets

    [Header("Tir")]
    [SerializeField] protected GameObject bulletPrefab;  Changed in 0+ assets
    [SerializeField] protected Transform bulletSpawnPoint;  Changed in 0+ assets
    [SerializeField] protected float bulletSpeed = 10f;  Changed in 0+ assets
    [SerializeField] protected float fireRate = 1f;  Changed in 0+ assets
    private float nextFireTime = 0f;

    [Header("UI Vie")]
    [SerializeField] protected GameObject healthBarGO;  Changed in 0+ assets
    [SerializeField] protected Image healthBarFill;  Changed in 0+ assets
    [SerializeField] protected Image easeHealthBarFill;  Changed in 0+ assets
    [SerializeField] protected float lerpSpeed = 0.05f;  Changed in 0+ assets
    protected new Camera camera;
```

Chasseur :

Comportement :

- Se déplace en ligne droite de d'un point a un autre.
- Tire à distance vers le joueur en visant dynamiquement dans se direction via une tourelle rotative.
- Despawn automatique au point d'arrivé.

Nuées :

Comportement :

- Se dirigent en continu vers le joueur, avec rotation fluide.
- Utilisent une logique de séparation locale pour ne pas se superposer aux autres Nuées.
- Ne tire pas, mais inflige des dégâts en collision.
- Calcul en temps réel d'un "flocking behavior" simplifié (Comportement vol d'oiseaux).

Garde-Bouclier :

Comportement :

- Tank défensif qui avance en ligne et se synchronise horizontalement avec le joueur. Il ne tire pas et agit comme une barrière physique mouvante pour protéger ses allié.
- Tant que le composant BouclierShield est actif, il bloque les dégâts reçus. Une fois détruit, le tank devient vulnérable.

Tourelle Mobile :

Comportement :

- Déplacement géré par Spline externe (non contrôlé par code via Move()).
- Se déplace lentement de gauche à droite (ou inverse) en suivant une spline d'animation sinusoïdale en haut de l'écran.
- Tire en continu vers l'avant pendant sa traversée.

Brise coque :

Comportement :

- Se positionne puis charge en ligne droite sur la position du joueur après une phease de préparation.
- Chaque charge est précédée d'un alignement vers le vecteur joueur et d'une animation sur chaque piques (via shaders).
- Ne tire pas, mais inflige de lourds dégâts en collision.
- Répète son pattern en boucle tant qu'il est en vie.

Croiseur-Tempête :

Comportement :

- Déclenche cycliquement un rayon face a lui qui represente une zone de dégats.
- Les dégâts sont simulés par des effets visuels + détection de zone (OverlapSphere) et agissent par tick réguliers.
- Reste en l'air et enchaîne les patterns toutes les X secondes.

Les Bosses :

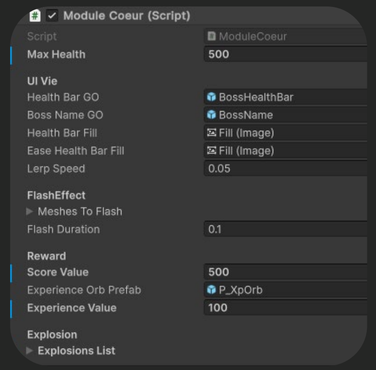
Structure Modulaire :

Les bosses agissentt comme climax du niveau, conçu pour tester les compétences du joueur acquises face aux ennemis rencontrés précédemment (chasseurs, tourelles mobiles, nuées, etc.).

Son design repose sur une architecture modulaire : un Module Cœur central, entouré de plusieurs Modules d'Armes qui conditionnent son comportement et sa résistance.

Module Coeur :

- Composant principal du boss, il possède ses propres points de vie.
- Il calcule une réduction des dégâts reçus en fonction du nombre de modules d'armes encore actifs (de 80% à 0%) : cela force le joueur à gérer les menaces périphériques avant d'atteindre le cœur.
- Il orchestre la destruction du boss entier en cascade quand il est vaincu (désactivation des modules, nettoyage des balles ennemies, VFX, score, XP, etc.).



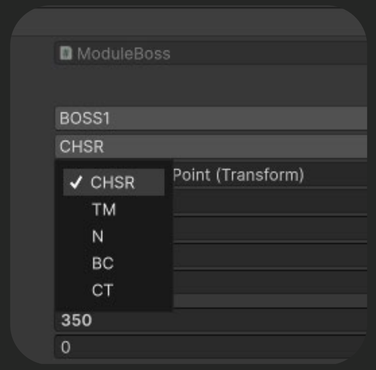
Module Armes :

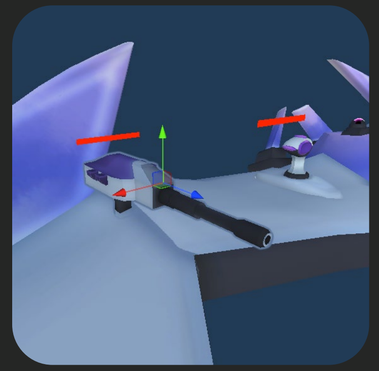
Chaque module hérite d'un système commun :

- Barre de vie indépendante, rotation/visée selon le type, flash d'impact
- Attaque et pattern propres (tir, spawn, zone, etc.).
- Destruction personnalisée avec effet visuel, et possibilité de déclencher des vagues d'ennemis à sa mort.

Les types de modules utilisés ici simulent les ennemis du niveau :

- CHSR (Chasseur) : tourelle rotative avec triple tir décalé.
- TM (Tourelle Mobile) : tir en éventail évolutif.
- N (Nuée) : spawn direct d'ennemis.
- BC (Brise-coque) : spawn d'unités kamikazes.
- CT (Croiseur Tempête) : zone de dégâts alternante, rotation contrôlée.





Premier Prototype :

Les 3 Bosses :

Vaisseau amiral Saul'ty

Raymond Th'a

Le Briseur de monde

- Chaque boss déclenche une phase à son arrivée : une animation d'intro avec musique dédiée est jouée, et le joueur est automatiquement soigné pour aborder le combat dans de bonnes conditions.
- À sa destruction, le boss libère une pluie d'XP, déclenche plusieurs explosions en cascade, marquant clairement la fin de la confrontation.

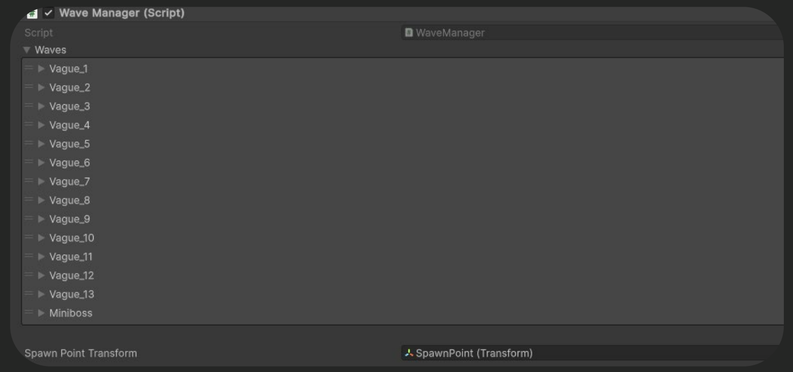
Bolt & Nut

Design orienté gameplay :

Wave Manager :

J'ai conçu un gestionnaire de vagues entièrement modulaire, pensé pour orchestrer la progression rythmée des combats tout au long du niveau, il gère l'entiereté des enchainements d'un niveau.

- Outil designé pour facilité l'integration de vague par scene.
- Creation d'un nombre illimité de vague suivant la structure "Wave"
- Chaque vague est instancié sur le Spawn Point Transform et s'auto-gère.
- La vague suivante ne démarre que lorsque tous les ennemis (ou boss) de la précédente sont éliminés.

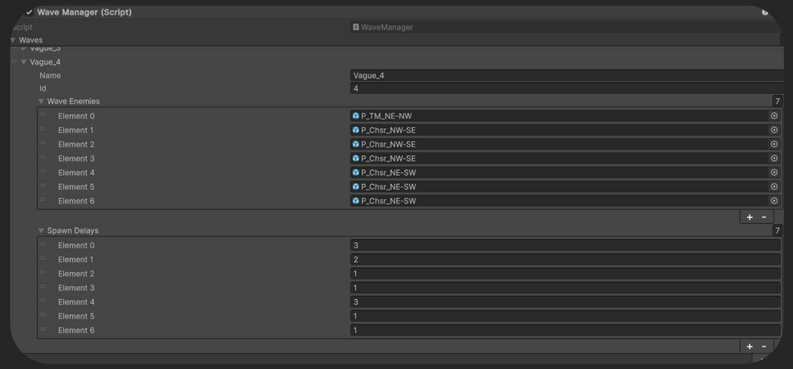


Wave Structure :

Structure modulaire : chaque vague est définie par un Wave contenant :

- Une liste d'ennemis
- Un tableau de délais de spawn associés

Les ennemis sont des préfabs qui peuvent avoir eux memes lors propre comportement (Exemple : 5 nuées dans le même préfab qui s'auto-gère).



Système de progression :

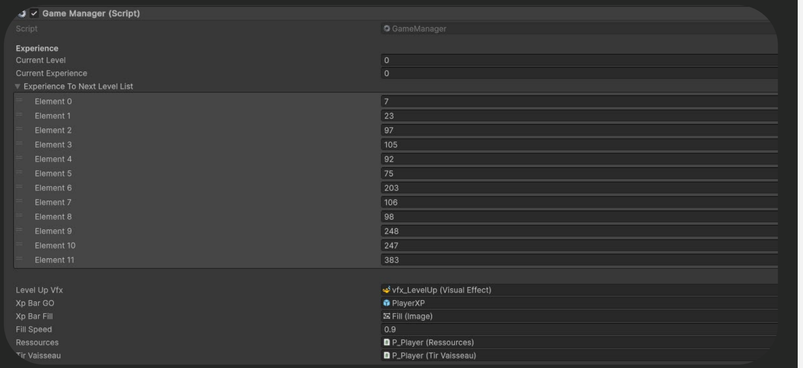
J'ai conçu un système de progression simple et visuel pensé pour accompagner la montée en puissance du joueur tout en maintenant un rythme de jeu clair, progressif et engageant.

Accumulation d'expérience :

- L'XP est automatiquement collectée via les étoiles lâchés par les ennemis à leur mort.
- Chaque ennemi donne une quantité définie d'XP (configurable individuellement).
- Les Boss en fin de niveau déclenchent une pluie d'XP à leur destruction.

Calcul de montée de niveau :

- XP nécessaire par niveau définie dans une liste incrémentale .
- Si la barre atteint le seuil, le joueur passe de niveau, conserve l'XP restante, et débloque immédiatement des bonus (Amélioration de tirs, Cadence ...).



Feedback de level-up :

- FX Visuel (VFX de montée en niveau, éclats à l'ecran).
- FX sonores dédiés.
- Soin automatique du joueur.
- Apparition d'une barre d'XP contextuelle avec effet de disparition automatique.

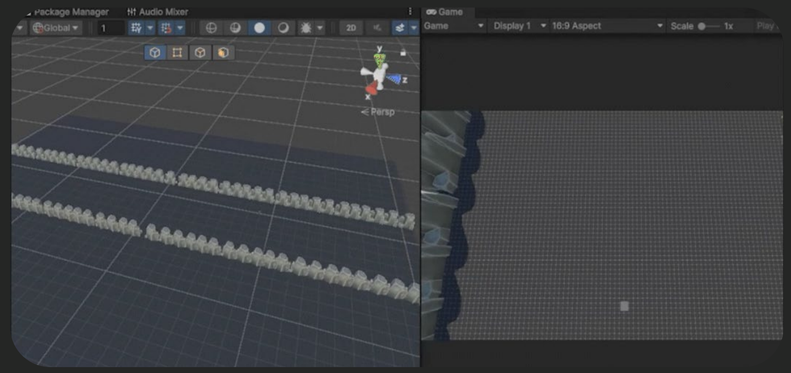
Bolt & Nut

Une structure optimisée :

Map Infinie (Scrolling Loop) :

J'ai conçu un système de map dynamique qui donne l'illusion d'un scrolling infini vers le joueur, tout en optimisant les performances.
Cela permet également d'avoir un jeu sans limite de temps.

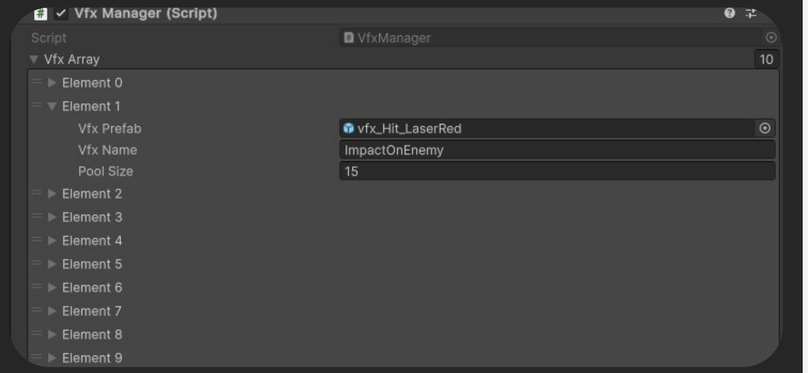
- La map est constituée de tiles (grands blocs de décor) qui se déplacent en continu vers le bas.
- Un trigger placé à l'arrière détruit les tiles sortants et en réinstancie un nouveau au début pour simuler une carte sans fin.



Pooling FX :

J'ai mis en place un système de pooling générique pour tous les VFX du jeu via le VfxManager.
Cela permet d'éviter l'instanciation/destruction répétée des effets visuels pour garantir des performances constantes, même en cas de pics d'action (boss, spams de tirs, explosions).

- Chaque VFX est défini dans une structure Vfx contenant un prefab, un nom d'appel (clé string), un nombre d'instances préallouées (pool size).
- Tous les effets sont instanciés une fois au lancement, désactivés, et réutilisés dynamiquement à l'appel (PlayVFX()).



Admin Panel (Debug UI) :

Un point important du developpement a été de maintenir un paneau administrateur fonctionnel afin de faciliter les itérations et tests en temps réel, sans avoir à relancer la scène ou modifier les scripts manuellement.

Essayez le jeu :

https://paul-hugy.itch.io/bolt-nut-ferrailleurs-de-lespace

Mot de la fin :

Ce projet a été un véritable tremplin de progression, autant en tant que développeur gameplay qu'en tant que chef de projet.

J'y ai conçu l'ensemble des systèmes clés : gestionnaire de vagues, architecture modulaire des boss, système d'XP et de level-up, carte infinie, object pooling, debug panel... avec une priorité constante sur la clarté, la modularité et les feedbacks joueurs (FX, UI, game feel).
Tout a été pensé pour construire une base de gameplay solide, fluide et cohérente.

Mais ce projet m'a surtout permis de travailler en équipe : diriger une petite team, répartir les tâches, gérer les plannings, tout en valorisant les compétences de chacun.
C'est là qu'on comprend à quel point le jeu vidéo est un travail d'équipe passionnant — et qu'un bon projet, c'est aussi savoir tirer le meilleur de chaque talent autour de soi.

Paul_Hugy

Bolt & Nut

e-artsup Vertigo

Expérience en réalité virtuelle développée sur Unreal 5.5
2025

Language :
Blueprint

Rôles :
Lead programmation
Participation Game design / User Experience

Durée :
Projet sur 5 semaines

Equipe :
Equipe de 9 personnes

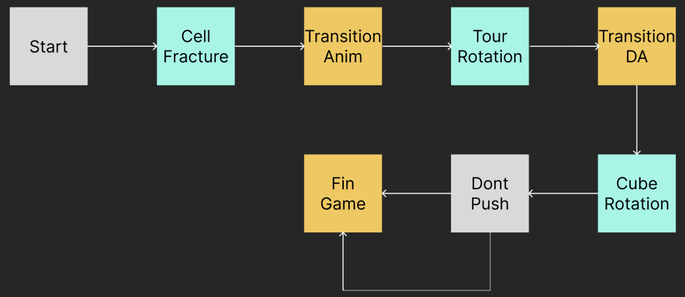
Brief :
Dans le cadre des portes ouvertes de l'école e-artsup Toulouse, l'objectif était de proposer une expérience interactive en réalité virtuelle permettant de mettre en avant la filière Game Design auprès des visiteurs.
Le projet devait être accessible, marquant et immersif, tout en s'intégrant physiquement dans l'espace réel de l'école.

Concept :
Vertigo est une expérience VR mêlant contemplation, énigmes légères et sensations de vertige.
Le joueur commence dans une reconstitution de la salle 307, puis explore plusieurs pièces, chacune proposant une épreuve liée au vertige.
Entre chaque énigme, il traverse des espaces mettant en valeur les filières de l'école (Game, Animation, DA), avec des galeries immersives présentant des travaux étudiants.
Pensée pour être intuitive et accessible, l'expérience dure environ 10 minutes et mise sur un fort impact visuel et sensoriel.

Projet Complet

Gestion de scène :

Nos besoins :



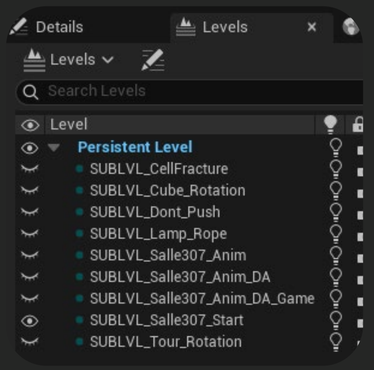
Dans Vertigo, l'expérience repose sur une succession de salles en réalité virtuelle.
Nous devions enchaîner plusieurs scènes dans un ordre précis, tout en respectant plusieurs contraintes clés liées à la VR :

- Transitions fluides : Pas de chargements visibles, pas de coupures brutales, les changements de scène devaient être naturels et intégrés dans l'expérience.
- Lighting indépendant : Chaque salle devait posséder son propre éclairage précalculé pour garantir à la fois performance et qualité visuelle, indispensable en VR.
- Clarté de production : Travailler à plusieurs sur différentes parties du projet exigeait une structure modulaire claire, pour répartir les responsabilités (gameplay /énigmes, observation).
- Flexibilité : Il fallait pouvoir facilement ajouter, retirer ou modifier des salles sans tout recompiler, ni casser l'ordre de progression.

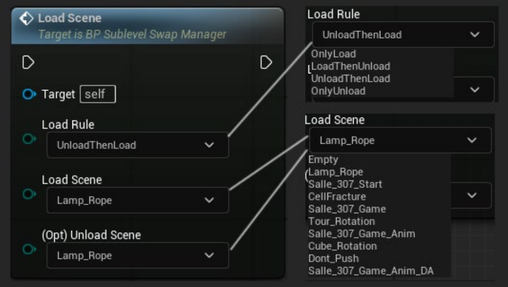
La réponse, un système modulaire et scalable :

Pour répondre à ces besoins, j'ai mis en place une architecture de gestion de scènes basée sur le level streaming dans Unreal Engine 5.5, combinée à une logique de transition orchestrée dans le Level Blueprint principal.

- Utilisation du Level Streaming via Sublevels pour chaque salle (énigmes et galeries).
- Chaque salle = sous-niveau indépendant, chargé/déchargé à la volée.
- Permet un lighting précalculé de manière efficace.



Gestion centralisée des transitions avec un blueprint customisé.
Celui ci comprend une fonction LoadScene avec des règles de chargement (LoadOnly, UnloadThenLoad, LoadThenUnload, OnlyUnload...) avec un choix du niveau à charger ou à décharger très simple d'utilisation.



Les scènes sont répertoriées dans un Enum E_Scene, ce qui permet d'ajouter, charger et décharger dynamiquement n'importe quel niveau.

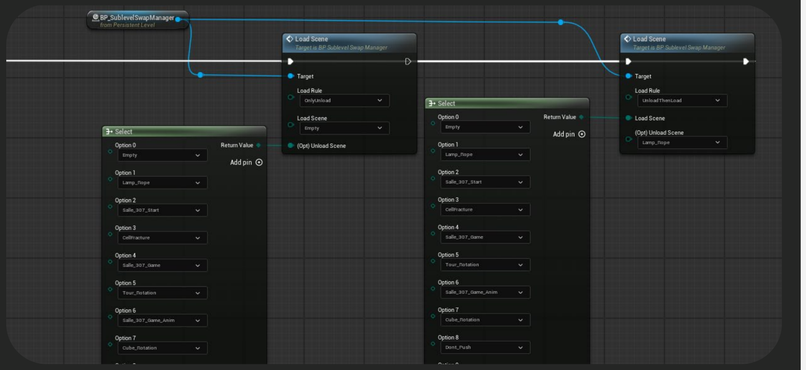
Et en jeu :

À la fin de chaque salle, une chaîne suspendue apparaît devant le joueur.

En la tirant, il déclenche un effet visuel : la lumière s'éteint et un fondu au noir masque la transition.

Ce geste envoie un signal (OnDrag) au Level Blueprint du niveau persistent.

- Le Level Blueprint appelle alors l'événement LoadScene dans le blueprint BP_SublevelSwapManager.
- Ce dernier utilise un index de progression pour déterminer la prochaine salle à charger.
- Il décharge proprement le niveau actuel, puis charge dynamiquement le suivant.
- Une fois le nouveau niveau prêt, un fondu inverse réactive la lumière, pour replonger le joueur naturellement dans l'expérience.



Résultat : Un système fluide, invisible pour le joueur, qui enchaîne les salles sans rupture et conserve l'immersion VR.

e-artsup Vertigo

Coté Gameplay :

Template VR Unreal :

J'ai choisi de partir sur le template VR d'Unreal Engine, qui offre une base solide (tracking des mains, interaction physique, gestion de la tête, etc.). Cela m'a permis de gagner du temps tout en assurant une intégration stable et performante.

J'ai tout de même apportés quelques modifications :

- Suppression de la téléportation, inutile dans notre expérience fixe.
- Ajout d'un input de recentrage du joueur, pour ajuster facilement sa position sans recalibrage.
- Ajout de colliders physiques sur chaque doigt, pour détecter des interactions fines avec les éléments du décor (boutons, objets d'énigme, etc.).

Énigme 1 – Reconstitution de la salle :

Pour cette première énigme, nous voulions une séquence marquante mêlant vertige, observation et interaction directe : le joueur se retrouve isolé sur une plateforme suspendue au-dessus du vide, avec une salle fragmentée en morceaux autour de lui.

Son objectif : Reconstituer progressivement la salle en plaçant correctement des pièces de puzzle physiques sur un plateau central.

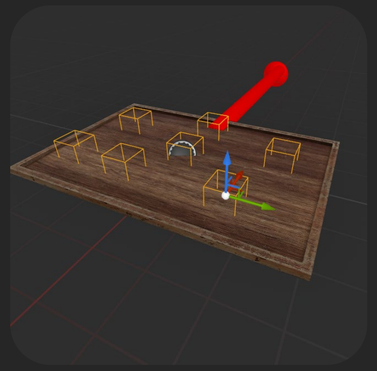
- Pièces de puzzle: Le joueur peut les attraper avec ses mains grâce à des collisions précises sur chaque doigt (ajoutées au VRPawn).
- Contrôle de placement fluide: chaque pièce déposée déclenche une animation qui la positionne proprement sur le plateau.
- Feedbacks visuels et sonores : à chaque succès, un fragment de la salle revient à sa position initiale, créant une boucle de reward claire, immersive et motivante.

L'énigme repose sur une architecture simple mais efficace, articulée autour de trois Blueprints principaux :

BP_PuzzleManager :
Orchestration de l'énigme : logique centrale, état des pièces, déclenchement de fin.

BP_Puzzle :
Plateau interactif : détection de la position et validation de la pose.

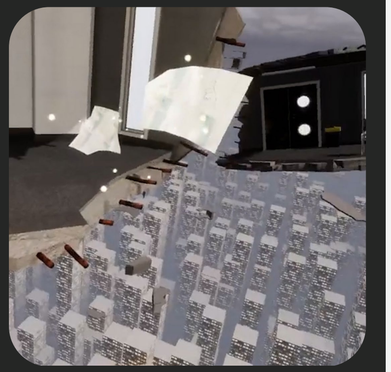
BP_PuzzlePiece :
Pièce manipulable : saisissable, indexée, et animée au placement.



Feedback :

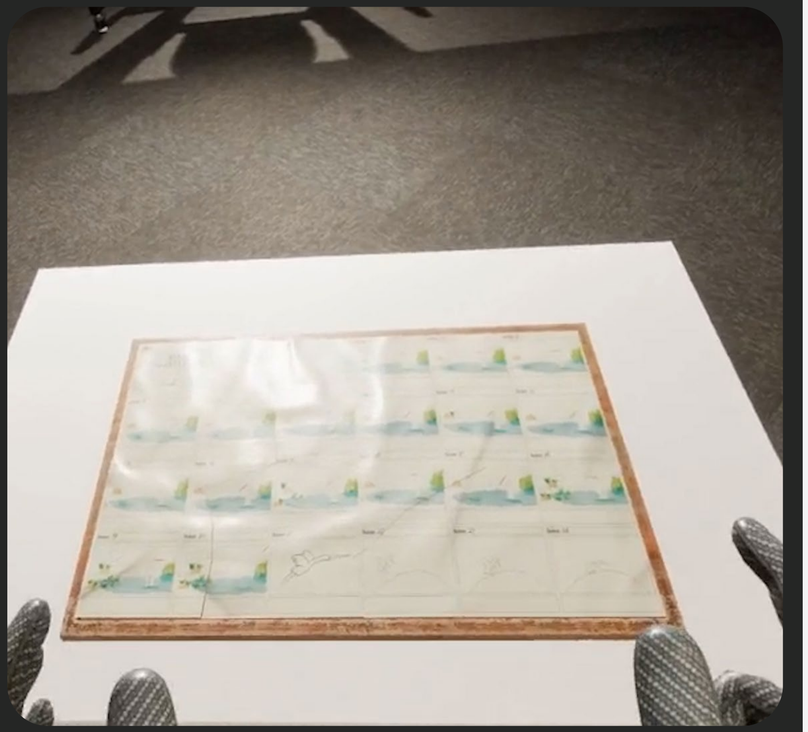
Un système de particules fait par l'equipe Art est visualisable sur chaque objet saissisable.

Celui ci est désactivé lorsqu'il est bien placé ou qu'il est deja en dans les mains du joueur.



Dès que toutes les pièces sont bien positionnées et que la salle est entièrement reconstituée, un signal OnEnigmeEnd est envoyé par le BP_PuzzleManager.

Ce signal déclenche l'apparition de la chaîne suspendue, signature de l'expérience, que le joueur peut attraper pour lancer la transition vers la salle suivante.



Énigme 2 – Système de tour rotative :

Pour cette seconde énigme, j'ai conçu un système basé sur une tour verticale de 8 étages rotatifs que le joueur doit réaligner manuellement.

Fonctionnement :

- Chaque étage est un Blueprint autonome (BP_FloorRotator) attaché à un pivot central.
- Le joueur utilise une console physique interactive composée de deux boutons de naviguer entre les étages (index) ainsi qu'un bouton de validation pour appliquer une rotation à 90° sur l'étage sélectionné.
- Une échelle visuelle (BP_EchelleFeedback) se met à jour en temps réel pour refléter l'étage actif via un système de Material Instances.

La console principale (BP_ConsoleFloorRotator) centralise la logique :
Elle gère l'index courant, envoie le signal de rotation au bon étage via un array de BP_FloorRotation référencé & applique des verrous d'interaction pour éviter les doublons en cas de multiple appuie sur les bouttons.

Interaction & immersion :

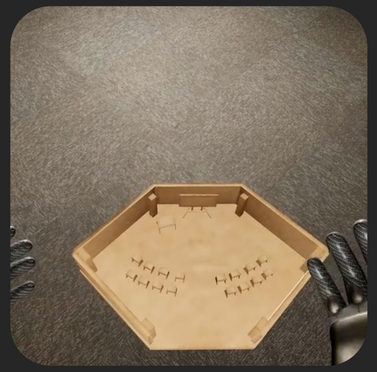
- Une lampe saisissable permet de se repérer dans une scene plongée dans le noir.
- Si elle est jetée dans le vie a la manière d'une flare, elle réapparaît automatiquement dans sa boîte, évitant toute friction ou confusion pour le joueur.

Énigme 3 – Simulation par maquette interactive :

Cette énigme repose sur un concept physique original : une maquette saisissable agit comme contrôleur direct de la salle réelle, que le joueur doit incliner pour en faire tomber les objets et atteindre un score cible.

Fonctionnement :

La salle suit dynamiquement la rotation de la maquette (BP_Maquette), avec un système fluide de suivi interpolé (BP_DuplicateRotation), déclenché dès la saisie.



Tous les objets présents dans la salle héritent d'un BP_ActorBasePoints, Blueprint parent générique :
Un AC_PointGiver (Actor Component) attaché permet de configurer le score de chaque objet de manière indépendante.
Seul le StaticMesh change d'un objet à l'autre (enceintes, néons, chaises, etc.), assurant un design modulaire et extensible.

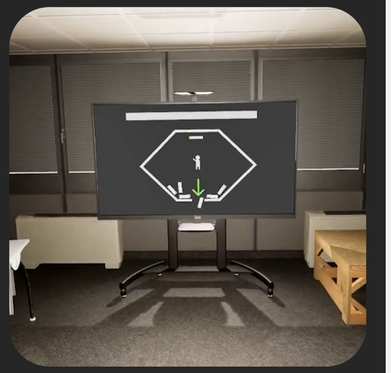
Le BP_PointsManager gère les collisions derière la sortie : chaque objet tombé est détecté, validé et incrémente le score du joueur.

Dynamique & immersion :

L'état du joueur est constamment lisible :

Le score est affiché en temps réel via une progress bar (BP_ScoreBar).

Une animation tutorielle (BP_TutoGif) explique l'objectif via une TV en début de salle.



Certains objets suspendus (enceintes, néons...) ne tombent pas immédiatement, mais se décrochent aléatoirement après un délai suite à l'interaction avec la maquette.
Cela crée des événements imprévisibles, ce qui renforce l'immersion.

Une fois l'objectif atteint, le signal de fin est automatiquement envoyé et l'accès à la salle suivante est libéré.

e-artsup Vertigo

Salle Don’t Push :

Une fausse sortie… ou presque.
Après l’énigme 3, le joueur quitte la salle via une chaîne suspendue, se retrouve dans la salle 307 avec toujours cette même chaine… Mais un bouton rouge attire l’œil… et la curiosité.

Lors de l’appui, une série d’événements se déclenchent :

- Signal d’alarme retentit
- Trois explosions Chaos successif dans la salle (FX réalisés par Bastien Canto)
- Une timeline anime la chute du joueur dans le vide
- Un crash sonor retentit en bas, simulant l’impact
- Le niveau se termine sans tirer la chaine habituelle.

Essayez le jeu :

https://bastini.itch.io/e-artsup-vertigo

La vidéo trailer :

Par Bastien Canto

Mot de la fin :

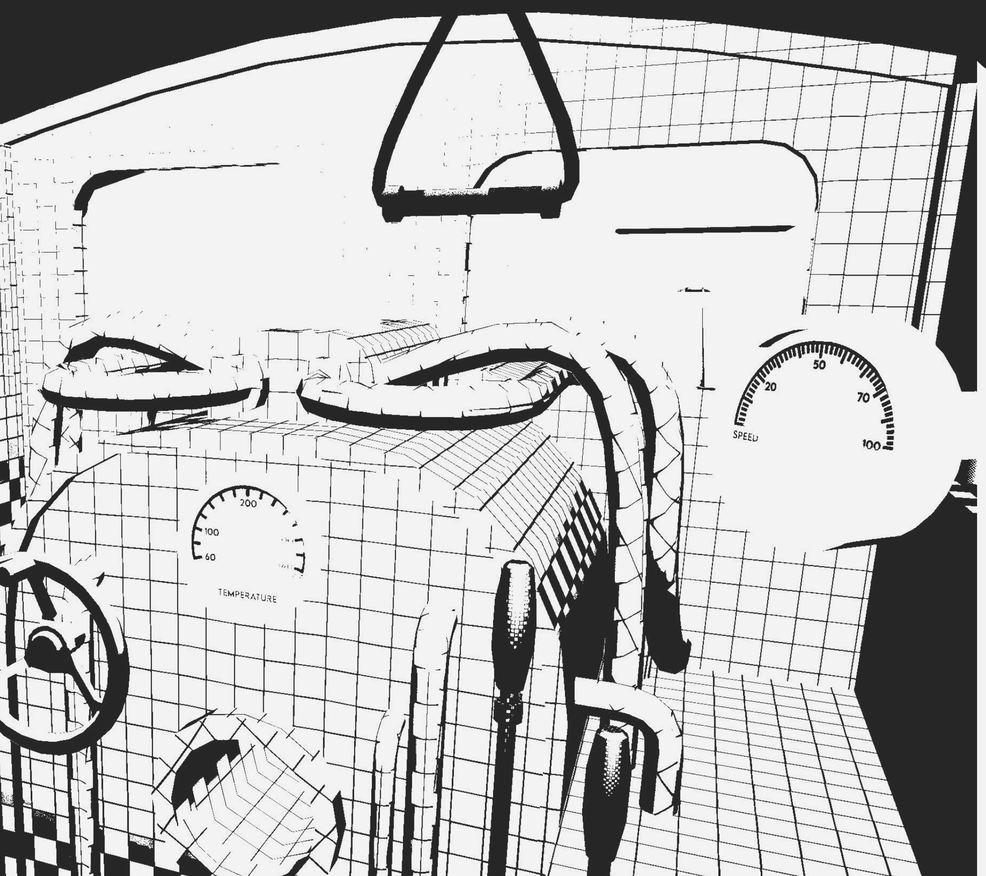
Vertigo a marqué une vraie étape dans ma progression en tant que développeur gameplay, et mon tout premier réél projet VR.

J’y ai découvert les exigences spécifiques à la réalité virtuelle : lisibilité, confort, optimisation du lighting… et appris à adapter mes systèmes à ce type de production immersive.

C’était aussi la première fois que je travaillais aux côtés d’un autre développeur. On a construit ensemble des fondations techniques solides, en échangeant beaucoup sur la structure, les besoins, les priorités.

Ce que je présente ici dans ce portfolio se concentre exclusivement sur mes réalisations personnelles au sein du projet :
architecture modulaire des énigmes, gestion du streaming, interactions physiques, feedbacks, logique de progression…

Et puis parfois, il faut aussi savoir ajouter un faux bouton rouge, trois explosions Chaos, une alarme stridente… et une chute dramatique.



Section
Game Jams

Prototypes réalisés en temps limité (48 à 72h) dans le cadre de Game Jams. L'accent est mis sur l'efficacité, la créativité, et la mise en place rapide de mécaniques de jeu fonctionnelles et originales.

Les Trois P’tits Bouffons

Prototype statégie Coop. développé sur Unity
2024

Language :
C#

Rôles :
Integration
Participation programmation, Game Design et Level Design

Durée :
Prototype réalisé en 3 jours de Global Game Jam sur le thème “Make Me Laugh”

Equipe :
Equipe de 6 personnes

Concept :
“Les Trois Ptits Bouffons” est un jeu en coopération locale pour 3 joueurs, dans lequel vous incarnez une bande de voleurs déguisés en bouffons infiltrés à la cour du Roi.

Votre mission : collaborer pour divertir la foule tout en dérobant ses richesses sans vous faire attraper.
Chaque joueur peut se présenter comme cracheur de feu, équilibriste ou jongleur pour attirer l’attention, pendant que ses complices commettent le larcin.

Mais attention : répéter le même numéro trop longtemps ennuiera la foule et attirera les soupçons ! Coordonnez-vous intelligemment, remplissez votre chariot de trésors et prenez la fuite avant la fin du temps imparti.

Game Jam

Mon premier Character Controller :

Systèmes de déplacement (Multiplayer local) :

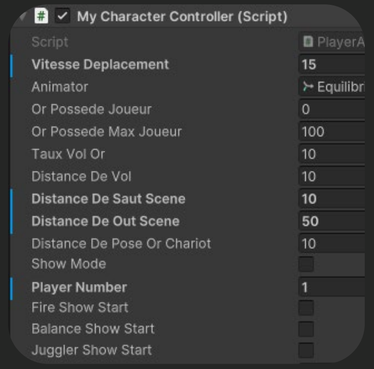
J'ai codé le déplacement pour les trois joueurs, chacun avec ses propres inputs.

Les mouvements sont fluides grâce à une gestion de l'accélération, et intégrés à un système de ralentissement dynamique selon l'or transporté par le joueur.
Plus on est chargé, plus on ralentit !

Détection et interaction avec l'environnement :

J'ai implémenté un système de détection d'éléments proches (PNJ, scènes, chariots) pour interagir de manière intuitive.
Cela permet au joueur de :

- Voler de l'or aux PNJ s'ils sont proches
- Monter ou descendre d'une scène
- Déposer son or dans un chariot si assez proche



Système de vol de pièces :

Si un joueur maintient le bouton d'interaction près d'un PNJ, il déclenche un vol progressif :
Le joueur vole un pourcentage de l'or du PNJ à chaque frame, avec un sprite visuel pour indiquer l'action.
Le vol s'arrête automatiquement à la relâche du bouton ou si la limite d'or est atteinte.

Un chariot centralise l'or volé par tous les joueurs.
J'ai implémenté une mise à jour visuelle dynamique du sprite selon plusieurs seuils permettant un feedback immédiat sur l'avancement de l'équipe.

Gestion des animations :

Chaque joueur est relié à un Animator en 2D basé sur des spritesheets, qui réagit à la direction du déplacement (haut/bas/gauche/droite) et change les poses automatiquement.
J'ai également ajouté des animations contextuelles selon l'état du joueur : idle, vol, spectacle, etc.

Mot de la fin :

"Les Trois P'tits Bouffons", c'est ma toute première Game Jam.
Je découvrais Unity, C#, les contraintes de production rapide... et surtout : le travail en équipe.

J'y ai codé mes premiers systèmes de gameplay :
le déplacement des joueurs, l'intégration de l'Animator, un système de vol d'or, la détection de zones, ou encore la mécanique de scoring via le chariot.

Ça m'a donné le déclic.
J'ai compris que ce métier me passionnait, mais qu'il demandait aussi rigueur, patience, écoute... et une bonne dose de débrouillardise.

C'était imparfait, mais formateur.
Depuis, j'ai envie de créer, progresser, et construire des jeux avec d'autres.

Essayez le jeu :

https://skilda.itch.io/les-trois-ptits-bouffons

Les Trois P'tits Bouffons

Paul_Hugy

Quack and Slash

Prototype survivor développé sur Unity
2025

Language :
C#

Rôles :
Chargé de toute la programmation et de l'integration
Participation Game design

Durée :
Prototype réalisé en 3 jours de Global Game Jam sur le thème "Bubble"

Equipe :
Chef de projet d'une équipe de 8 personnes aux profils variés, comprenant des membres débutants dans le développement de jeux

---

Concept :
Incarnez un canard héroïque dans un monde complètement loufoque !
Dans ce survival unique, affrontez des hordes de créatures sales déterminées à éclater votre précieuse bulle.
Tuez ces monstres pour faire grossir votre bulle, la rendant de plus en plus difficile à défendre, mais débloquez en échange des compétences qui vous aideront à tenir bon.

Game Jam

Mécaniques de base :

Personnage jouable, Quack :

Déplacement fluide (Rigidbody + Accélération) :

Le joueur se déplace en appliquant des forces, avec accélération, décélération, clamp de vitesse et rotation vers la direction du mouvement.

Résultat : un contrôle souple, réactif et agréable à la manette ou clavier.

Attaque automatique orbitale (épée tournante) :

Toutes les X secondes, une épée est instanciée et tourne automatiquement autour du joueur.
Sa rotation accélère exponentiellement pour un effet dynamique, et s'accompagne d'animations + sons.

Résultat : feedback satisfaisant et mécanique défensive passive.

Animation contextuelle :

Différentes animations sont jouées afin que le joueur comprenne ce que se passe via l'animator.

- isMoving: joue les animations de déplacement / Idle .
- isAttacking: joue les attaques automatiquement pendant l'orbitation de l'arme

Builds Progressifs :

Système de compétences évolutives :

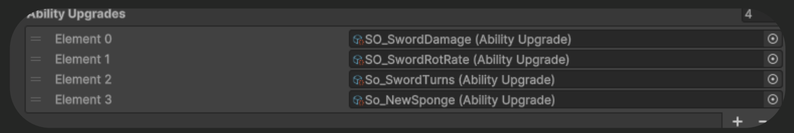
J'ai conçu un système de perks, qui récompense le joueur à chaque niveau par des choix d'améliorations aléatoires :

L'idée étant de :

- renforcer la rejouabilité avec des builds variables.
- donner un sentiment de progression gratifiante
- ajouter une couche de décision rapide sous pression
- et bien sûr, enrichir la boucle de gameplay (tuer - survivre - améliorer - recommencer).

Fonctionnement :

- Tous les X ennemis tués, le joueur gagne un niveau d'expérience.
- Un menu de sélection s'ouvre avec 2 perks tirées aléatoirement.
- Il en choisit une, immédiatement appliquée. Si elle est stackable, elle pourra revenir jusqu'à atteindre son niveau max.
- Un UpgradeManager central gère le tirage aléatoire, l'application, et le retrait du pool lorsque le niveau max est atteint.
- Le GameManager écoute les upgrades et applique leurs effets en déléguant aux systèmes concernés (ex : SwordManager).



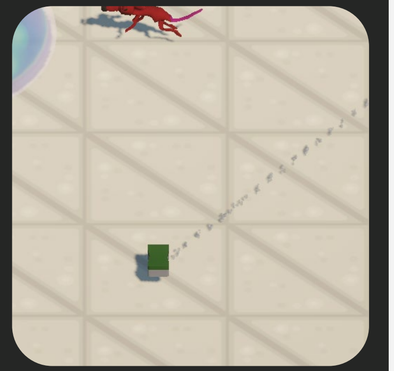
Chaque perk est un ScriptableObject contenant son ID, nom, description, icône et nombre max de stacks.

Ce système m'a permis de poser des bases pour de la progression dynamique, tout en rendant le code modulaire : il suffit de créer un nouveau ScriptableObject et d'écouter son ID pour ajouter une nouvelle compétence.

Exemple d'upgrades pour la Sponge :

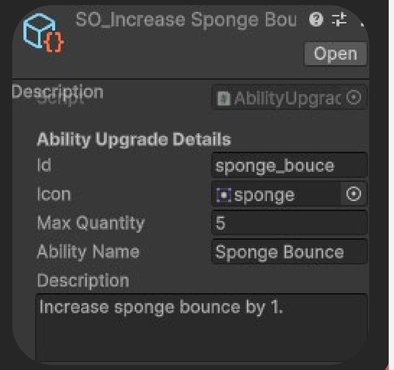
L'éponge est une unité défensive que le joueur peut faire apparaître via une compétence.

- Elle cherche automatiquement un ennemi à proximité et se propulse dessus à grande vitesse.
- Elle rebondit sur les murs (jusqu'à X fois) avant de disparaître.
  Elle inflige des dégâts à l'impact et déclenche des effets sonores pour renforcer l'impact.



Les upgrades permettent :

- D'augmenter le nombre de rebonds avant destruction (IncreaseSpongeBounce()),
- D'en faire apparaître plus souvent, en relançant automatiquement un spawn après un cooldown de 3 secondes.



Quack and Slash

Système d’ennemis & équilibrage dynamique :

Logique de spawn évolutive :

J’ai conçu un système de spawn circulaire dynamique qui entoure progressivement le joueur à distance variable.

- Spawn en cercle autour de la bulle, entre un rayon min et max.
- Randomisation du type d’ennemi à chaque spawn (rat ou poulpe) avec poids.
- Intervalle de spawn qui se réduit automatiquement toutes les 20 secondes, rendant la partie de plus en plus tendue.

Rat (ennemi de base) :

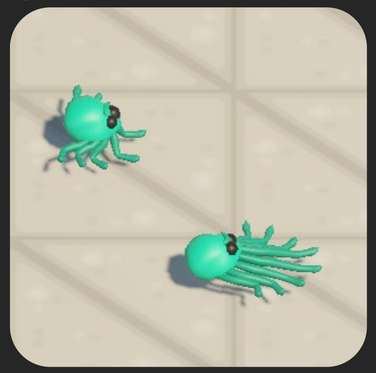
Comportement :

- Se dirige en continu vers la bulle.
- Simple, rapide, toujours actif.
- Sert de pression constante sur le joueur.

Poulpe (ennemi rythmique) :

Comportement :

- Synchronisé à une fenêtre d’animation précise.
- Se déplace par à-coups rythmés, uniquement pendant une plage de frames définie.
- Crée un rythme de menace imprévisible.



Système de dégâts :

Tous les ennemis partagent :

- Un système de santé unifié (Damageable.cs) avec callback onDeath pour spawn d’XP.
- Un feedback visuel à chaque dégât via un tween de scale (DoTween).
- Une mort animée avec FX et son.

La bulle, à la fois cœur, score… et faiblesse :

Une mécanique centrale :

La bulle représente l’objectif principal du joueur : la protéger à tout prix. C’est une zone fragile, symbole d’équilibre entre progression et danger.

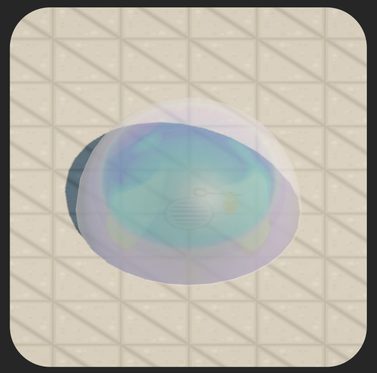
- Chaque ennemi éliminé génère une orbe d’xp qui est absorbé par la bulle
- En collectant un XP, la bulle grandit visuellement.

Plus elle grossit, plus :

- Sa hitbox augmente, donc plus difficile à défendre.
- Elle occupe de l’espace à l’écran, réduisant la lisibilité.

C’est un dilemme volontairement conçu : progresser rend la partie plus tendue.

- Si un ennemi touche la bulle : Game Over.
- L’explosion est visuelle, sonore, brutale : un rappel que la bulle, par nature, finit toujours par éclater.
- C’est une boucle fermée qui est voulue : progresser, s’enhardir, devenir vulnérable, perdre.



Mot de la fin :

Même en Game Jam, j’ai voulu poser des bases solides.
Le système de perks, par exemple, a été pensé dès le départ comme un système modulaire et évolutif, pour pouvoir intégrer facilement les idées de l’équipe et leur donner vie sans bloquer le flow de prod.

Ce que j’ai adoré dans Quack and Slash, c’est la diversité des niveaux et profils dans l’équipe. Ça crée des échanges hyper riches. Tu te retrouves à expliquer des trucs que tu fais un peu par réflexe… et tu réalises que ce réflexe, tu l’as acquis avec le temps, mais qu’il mérite d’être partagé.

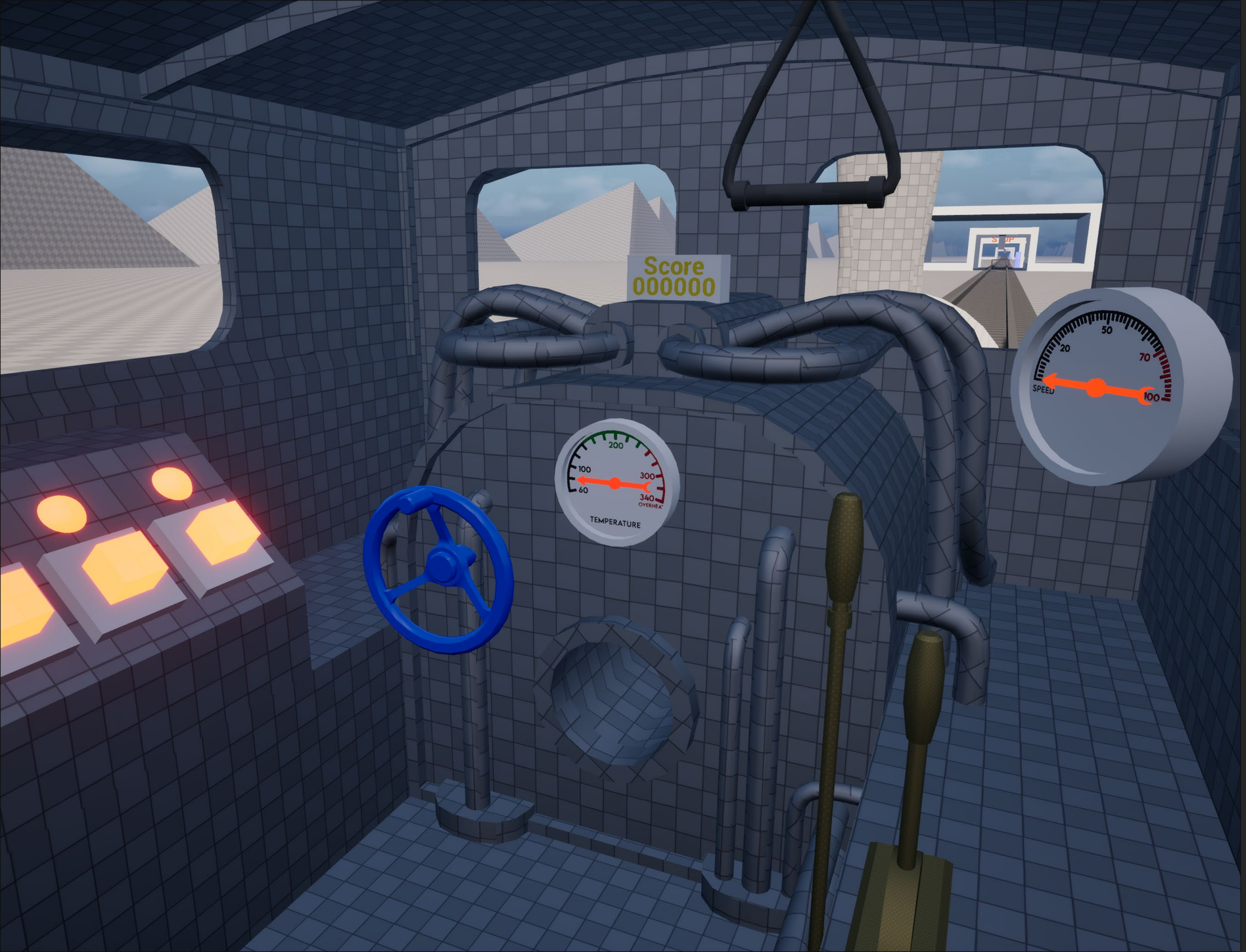
C’est dans ce genre de projet que tu te rends compte que le game dev, au-delà d’être une passion, c’est un vrai métier. Complexe, exigeant, mais ultra stimulant. Et il te fait progresser même quand tu crois “juste faire une jam”.

Essayez le jeu :

https://paul-hugy.itch.io/quack-and-slash

Quack and Slash

Paul_Hugy



Tchou T’chwo

Prototype en réalité virtuelle développé sur Unreal 5.5
2025

Language :
Blueprint

Rôles :
Programmation
Game design

Durée :
Prototype réalisé en 2 jours de Game Jam

Equipe :
Equipe de 2 personnes, réalisé en collaboration
avec Bastien Canto

Concept :
Tchou t’chwo est un prototype en réalité virtuelle dans lequel vous incarnez un conducteur de train de marchandises.
Votre objectif : conduire le plus rapidement possible, tout en gardant votre train en bon état et en livrant les bonnes marchandises à chaque arrêt.
Entre gestion de la vitesse, freinage, manœuvres de livraison et pression du temps, le joueur doit faire preuve d’efficacité pour maximiser son score.

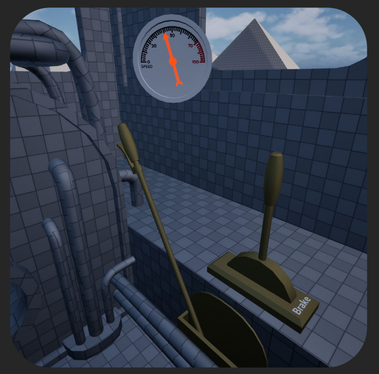
Game Jam

Réalité Virtuelle = Un maximum d'interactions pour le joueur :

Leviers (Accélération / Freinage) :

Deux leviers à saisir en VR, basés sur une lecture continue de la position du Motion Controller par rapport au point de pivot du levier :

- Rotation contrôlée en temps réel via SetTimer by Function, avec une fonction UpdateRotation.
- Clamp de la rotation pour éviter les valeurs extrêmes et bien bloquer le levier dans la zone de valeurs voulue.
- Permet une modulation fine de l'accélération et du freinage.



Boutons de dechargement :

Boutons interactifs à presser avec retour visuel (déplacement relatif en Z) + callback (Event Dispatcher) a l'appuie / relâchement :

- Activation via détection d'overlap avec la main.
- Reset automatique après interaction.

Valve de température :

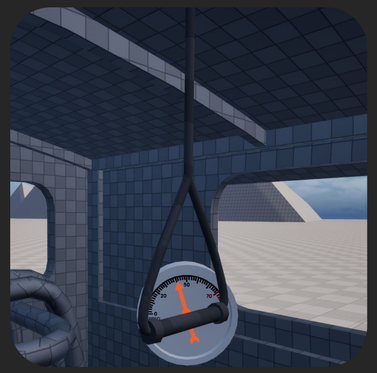
Un composant tournant dynamiquement autour de son axe, une fois saisi :

- Calcul via dot product et angle relatif.
- Rotation fluide avec cumul des tours, pour déclencher des effets selon l'amplitude.
- Valeur finale utilisée pour alimenter la jauge de température.

Klaxon suspendu :

Un composant suspendu à tirer (corde avec poignée) :

- Détection de mouvement vertical vers le bas.
- Joue un son lorsque l'amplitude est suffisante.
- Feedback visuel avec retour automatique.



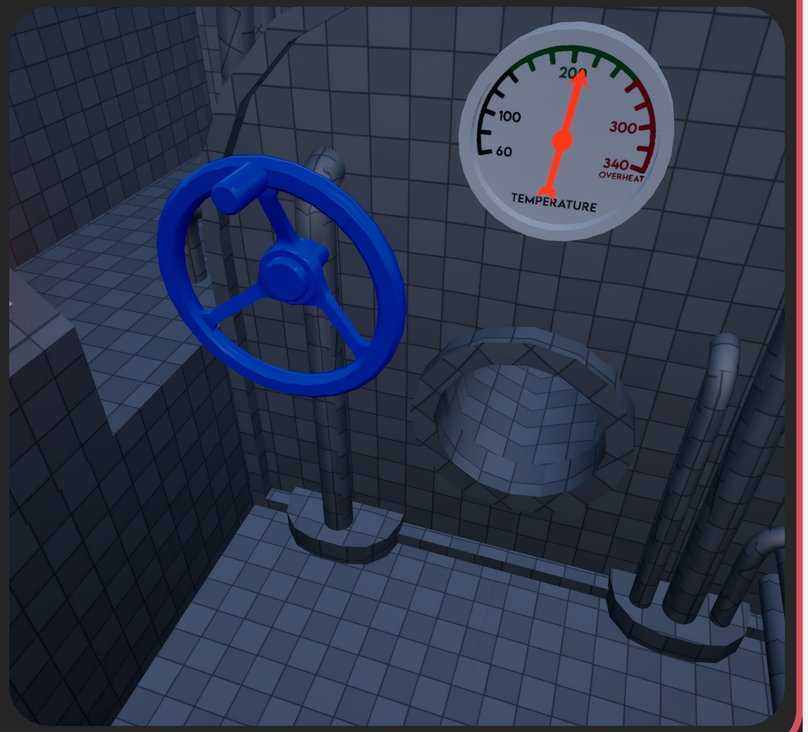
Système de chaudière dynamique :

Ce système simule une chaudière instable que le joueur doit réguler en temps réel à l'aide d'une roue rotative.
La température est représentée par une aiguille analogique sur un compteur visible dans la cabine.

- Toutes les 3 secondes, une variation aléatoire vient perturber la température cible, simulant les soubresauts d'un moteur sous pression.
- Le joueur peut alors corriger en tournant la roue (gauche : refroidir / droite : chauffer), ce qui ajuste dynamiquement la température cible.
- Une inertie thermique est simulée : la température réelle ne suit pas immédiatement la cible, ce qui demande anticipation et précision.

Résultat :

- Une chaudière bien chaude = vitesse max + gain de score.
- Une surchauffe prolongée = perte de points, et explosion si elle est ignorée.
- Une température trop basse = vitesse du train réduite, livraisons ratées.



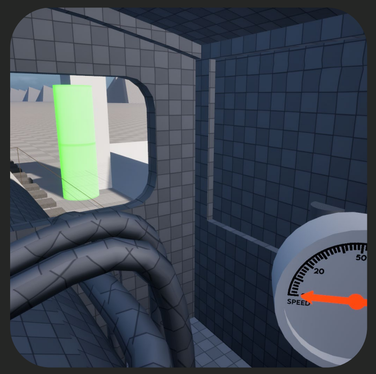
Tchou T'chwo

Les arrêts en gare :

Détection des arrêts :

Cette zone permet de détecter l'arrivée et le départ du train dans une gare et permet au système de livraison de se déclencher uniquement quand le train est à l'arrêt complet dans la zone dédiée.

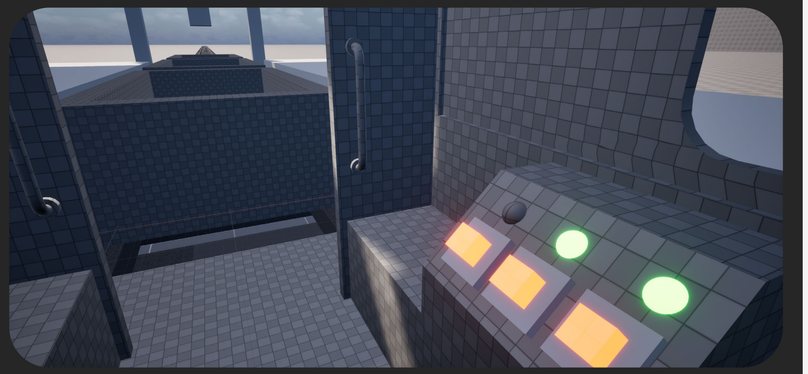
- Une box de collision détecte l'entrée/sortie du train (via Tag "Train").
- Un matériau dynamique sur le cylindre central change de couleur (vert = train en gare / bleu = vide), assurant un feedback visuel clair.



Système de livraison :

Une mécanique simple mais efficace, conçue pour renforcer la boucle de gameplay : vitesse - arrêt - interaction - récompense.

- Lorsqu'un train s'arrête complètement dans une Zone d'Arrêt, les quais génèrent aléatoirement une commande de ressources parmi trois types représentée en cabine par trois voyants lumineux : un voyant allumé indique un besoin actif.
- Le joueur doit alors appuyer sur le(s) bon(s) bouton(s) pour lancer la livraison depuis le bon wagon.



Mot de la fin :

Ce prototype a été réalisé en 2 jours de Game Jam avec Bastien Canto, juste avant le projet Vertigo e-artsup.
L'idée : découvrir la VR dans Unreal et comprendre ce que ça change vraiment pour le gameplay.

C'est sur ce projet que j'ai eu le déclic. En VR, le joueur veut tout attraper, tout tester. Chaque objet est une interaction potentielle et tout doit être clair, fluide, satisfaisant.

J'ai pris en main le VR Template d'Unreal, conçu plusieurs interactions physiques précises et compris à quel point le game feel est essentiel en réalité virtuelle.

Une petite jam, mais une vraie prise de conscience.

Section
Expérimentations

Expérimentations techniques centrées sur le gameplay pur. Tests de systèmes, IA, mécaniques de combat ou locomotion. Ces prototypes sont pensés pour apprendre, explorer et repousser mes limites actuelles.

Proto K2 Crown

Prototype développé sur Unreal 5.5
2025

Language :
Blueprint

Rôle :
Programmation

Durée :
Prototype réalisé en 3 jours

Equipe :
Equipe de 2 personnes, réalisé en collaboration avec Bastien Canto

Objectif :
Notre objectif était de prototyper les principales mécaniques de gameplay de Kingdom Two en un temps très court.

Expérimentation

Des entités autonomes :

Système d'héritage :

L'un des objectifs principaux de ce prototype était de créer un système d'IA modulaire et évolutif, capable de simuler des comportements similaires à ceux des villageois dans Kingdom Two Crowns.
Pour cela, j'ai conçu une classe parent BP_Entity, utilisée comme base commune à toutes les unités autonomes du jeu (Vagabond, Villageois, Ouvrier, Archer).
Cette classe contient les fonctions partagées (comme la patrouille autour d'un point), et chaque enfant hérite et spécialise son comportement selon son rôle.

Ouvrier, construction et gestion des structures :

- Les ouvriers cherchent en permanence les structures marquées comme incomplètes (Tag : Construction) grâce à une détection basée sur la distance et les tags.
- Une fois proche d'un bâtiment, ils invoquent une interface BPI_ Constructable pour appeler la fonction Build, réalisant ainsi la construction.
- Ils sont aussi responsable de récolter les arbres marqués par le joueur.

Le système repose sur une interface générique, permettant d'étendre très facilement la logique de construction à d'autres types de bâtiments.

Vagabond, près du feu :

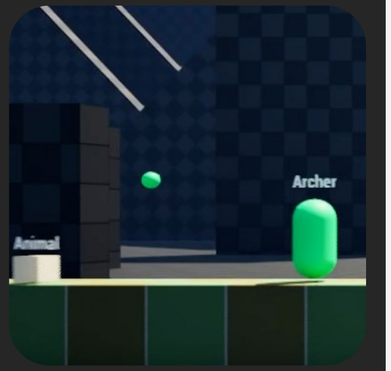
- Les vagabonds patrouillent automatiquement autour d'un feu de camp grâce à la fonction PatrolAroundBase, qui sélectionne aléatoirement un point dans un rayon défini autour de leur position initiale.
- Chaque vagabond possède un SphereTrigger qui détecte les pièces déposées par le joueur.
- Si une pièce est détectée à proximité, le vagabond est détruit et remplacé dynamiquement par un villageois via un SpawnActor.

Ce comportement crée une boucle de recrutement autonome et intuitive, inspirée du gameplay de Kingdom, tout en restant simple à étendre.

Archer, défense et chasse, selon le cycle jour/nuit :

- Les archers adaptent leur comportement selon la variable IsNight (transmise par la GameInstance).
- Ils cherchent une tour libre et s'y postent automatiquement pour défendre le camp.
- Le jour, ils partent en chasse, choisissent une direction, recherchent les animaux dans leur zone, et tirent à vue.
- La nuit, ils rentrent au camp pour le défendre.

Ce système met en avant un comportement contextuel, réactif et spécialisé, dépendant à la fois de l'environnement et du moment de la journée.



Villageois, état transitoire en attente d'un métier :

- Lorsqu'un villageois est généré, il retourne automatiquement à la base via une commande MoveTo.
- Une fois sur place, il entre dans un état d'attente (WaitingForTool), et surveille les outils disponibles à proximité.
- Lorsqu'un outil est détecté (arc ou marteau), il se transforme en unité spécialisée (BP_Archer ou BP_Ouvrier) et est remplacé dans le monde.

Ce fonctionnement rend l'unité entièrement dépendante de l'écosystème du joueur, renforçant la sensation d'un monde vivant.

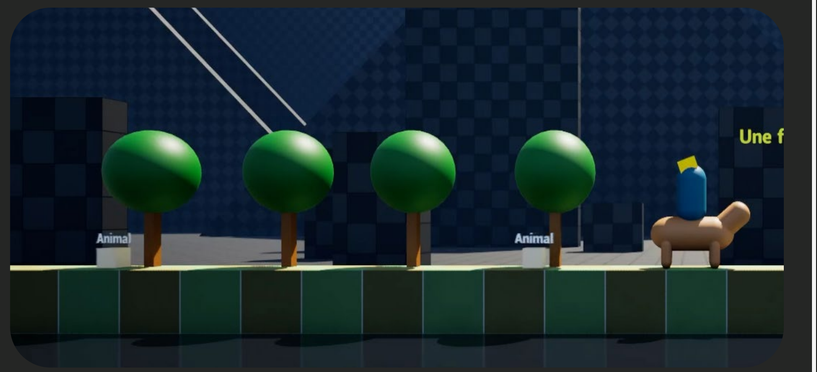
Grâce à ce système autonome et modulaire, le joueur n'a qu'un seul levier stratégique à gérer : l'investissement de pièces.
C'est le système qui s'occupe de tout le reste : recrutement, attribution des rôles, déplacements et comportements.
Une approche simple à jouer, mais profonde à designer !

Proto K2 Crown

Écosystème, les animaux :

J'ai intégré le système d'animaux mobiles pour enrichir le gameplay économique :
les archers peuvent les chasser la journée, et chaque animal détruit génère une pièce.

- Chaque BP_Animal hérite de la classe BP_Entity, ce qui permet de réutiliser le système de patrouille existant.
- Les animaux se déplacent aléatoirement autour de leur point d'apparition, à la manière des villageois.
- Lorsqu'un archer entre en collision avec un animal, celui-ci est automatiquement détruit, ce qui déclenche la génération d'une pièce et la destruction de l'animal.



Système des Ennemis

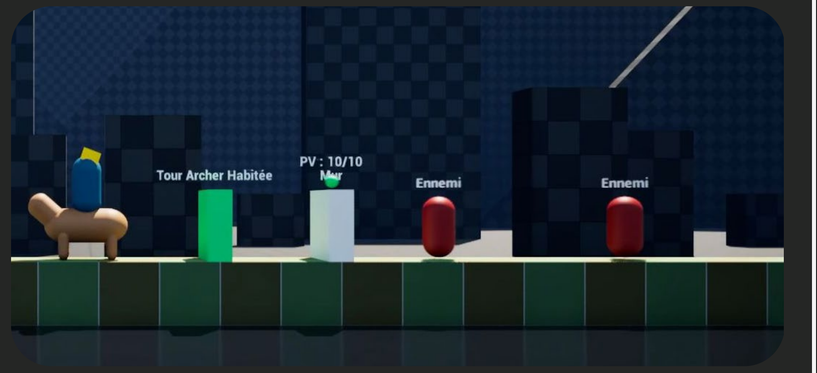
Les ennemis sont eux aussi dérivés d'un BP_Entity. Contrairement aux unités alliées, ils ne sont pas autonomes mais suivent une logique simple et déterministe, conçue pour mettre en tension les défenses du joueur.

À leur apparition (BeginPlay), les ennemis sont automatiquement envoyés vers le village, en passant par un portail d'apparition.

Leur état (EnemyState) est géré par une énumération et conditionne leur comportement (aller au village, retourner au portail, ou attaquer un mur).

Lorsqu'un ennemi entre en collision avec un autre acteur, on vérifie les tags de l'objet touché pour déclencher la bonne logique via un Switch on Name :

- Archer ou Ouvrier : l'unité subit un downgrade, et l'ennemi récupère une arme et part avec.
- Villageois: Redevient Vagabond
- Mur : l'ennemi entre en état DestroyingWall et déclenche une destruction progressive.
- Joueur ou pièce : Vol une pièce et part avec celle-ci..



Essayez le jeu :

https://paul-hugy.itch.io/prototype-two-crown

Mot de la fin :

Dans ce prototype, je me suis concentré principalement sur la gestion des entités IA. Le défi était de concevoir un système simple, modulaire et efficace, sans recourir aux outils classiques comme les Behavior Trees.

J'ai adoré réfléchir à une logique claire, orientée états et transitions, capable de s'intégrer facilement dans un système global.
Ce qui m'a vraiment plu, c'est de voir comment mes entités (vagabonds, villageois, ouvriers, archers, ennemis...) venaient naturellement s'imbriquer dans la mécanique d'économie et de progression gérée par Bastien Canto.
Une vraie synergie de prototypage rapide !

Proto K2 Crown

Paul_Hugy

Octopus Rifle

Prototype développé sur Unreal 5.5
2025

Language :
Blueprint

Rôle :
Chargé de toute la programmation

Durée :
Experimentation réalisée sur 1 semaine

Equipe :
Equipe de 2 personnes, réalisé en collaboration avec Bastien Canto

Objectif :
Expérimenter des systèmes de locomotion avancés dans UE5.

Travailler sur un setup complet de personnage jouable, incluant animation, armes, IK et feedback.

S'approcher du niveau de polish technique d'un projet type Lyra (Epic).

Système de gameplay du joueur :

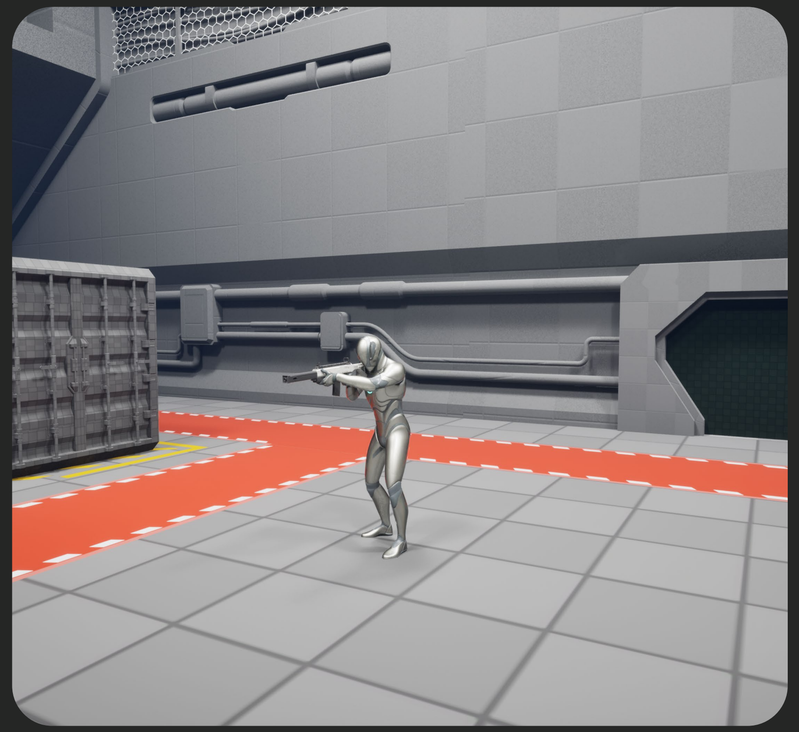
Setup Général – Un Blueprint pensé pour la modularité :

Parti d'un blank template, j'ai construit un personnage jouable entièrement en Blueprint, structuré et évolutif.
L'objectif était double :

- Comprendre en profondeur chaque brique système du personnage.
- Me doter d'une base claire, scalable et facilement itérable pour des prototypes futurs.

Le personnage principal, BP_Ice, regroupe :

- La gestion des inputs via l'EIS.
- Un système d'animation modulaire inspiré de Lyra.
- Un Actor Component dédié aux armes.
- Un contrôle complet du déplacement et de l'état d'animation.



Système d'Armement – Centralisé dans un Actor Component :

J'ai créé BPC_Weapons, un Actor Component chargé de centraliser tout le système de tir et de gestion d'arme.
Il repose sur un Enum E_Weapons (Unarmed, Rifle), et peut être étendu très facilement à de nouvelles armes.

Ce component gère :

- L'arme actuellement équipée (EquippedWeapon).
- Le nombre de balles, les tailles de chargeur, la réserve.
- Les booléens de rechargement, de tir possible (CanFire), etc.
- Le swap d'arme, les montages, et les mesh associés.

Tout le système est conçu pour être autonome, réutilisable dans d'autres personnages, et déconnecté de la logique du BP principal.

Lors du tir, plusieurs feedbacks visuels et sonores sont déclenchés pour renforcer l'impact et la sensation d'un système de tir réactif et crédible :

- Une animation de recul est jouée directement sur le skeletal mesh de l'arme, apportant du réalisme au visuel.
- Un raycast (line trace) est lancé pour simuler la trajectoire de la balle.
- À l'impact, deux systèmes Niagara sont déclenchés : un tracer pour visualiser la trajectoire ainsi q'un effet d'impact contextuel (étincelles, fumée...)
- Le son de l'impact est également déterminé dynamiquement, en fonction du Physical Material touché (métal, sol, chair, etc.).

Ce système permet de renforcer le feedback du tir avec un setup modulaire, extensible à d'autres armes ou types de projectiles.

Système de Déplacement – Gait & Data dynamiques :

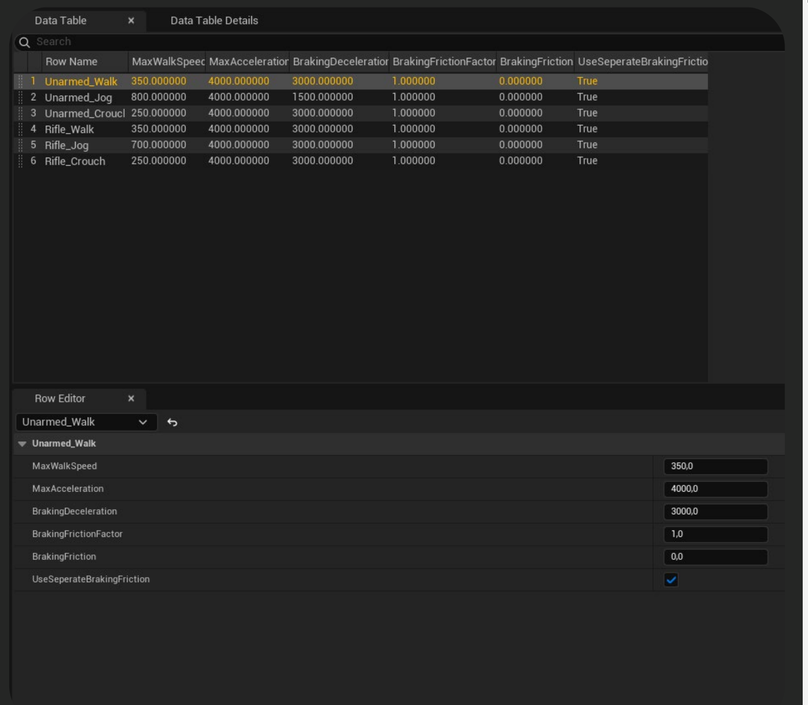
Le système de locomotion est piloté par un Enum E_GaitIce qui gère 3 types de déplacements :

- Walking
- Jogging (valeur par défaut au démarrage)
- Crouching

Le CurrentGait est utilisé pour piloter dynamiquement le CharacterMovementComponent.
J'ai mis en place une fonction UpdateGaitSettings qui ajuste tous les paramètres de mouvement selon le Gait ET l'arme équipée.

Cela permet de réduire la vitesse quand on porte une arme ou d'adapter la friction ou l'accélération selon le type de déplacement

Cette fonction récupère dynamiquement ses données via une DataTable et elle est appelée à chaque changement d'arme ou de Gait, garantissant un comportement fluide et contextuel.



Octopus Rifle

Systeme d’animation complet :

Structure générale & architecture :

- ABP_Ice :

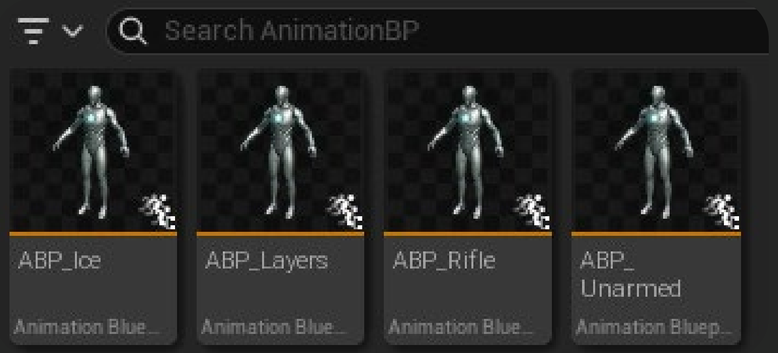
Blueprint principal, contenant la state machine de locomotion, le blend pose final et l’ensemble des contraintes IK.
Il orchestre toutes les données gameplay et les redistribue aux layers via un système thread-safe.

- ABP_Layers (Linked Anim Layer) :

Cœur de l’animation contextuelle. C’est dans cette couche que se font les blendspaces directionnels, les transitions de gait, et les animations spécifiques à l’arme ou à l’état (saut, pivot, idle...).
Les layers sont interchangeables à runtime via Link Anim Class Layers, permettant de switcher dynamiquement entre ABP_Rifle, ABP_Unarmed, etc.

- ABP_Rifle / ABP_Unarmed :

Ces enfants d’ABP_Layers contiennent les poses, montages, et paramètres propres à chaque type d’équipement.
L’animation est sélectionnée dynamiquement via des Switch On Enum, conditionnés par l’arme équipée (stockée dans l’ActorComponent BPC_ Weapons).

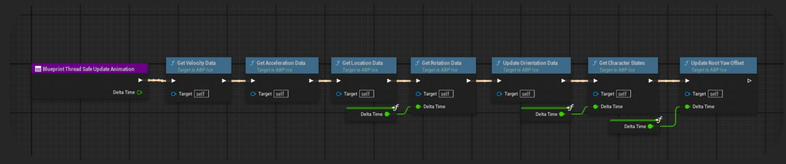


Gestion des datas – Thread-Safe Update :

Pour garantir un Update Animation robuste, toutes les datas de locomotion sont récupérées dans une fonction dédiée Blueprint Thread Safe Update Animation, appelée une seule fois par frame :

- Vitesse, accélération, direction, distance au sol, orientation du joueur, état de gait, en l’air ou non, etc.
- Ces infos sont injectées dans les layers via une structure de données partagée (Anim Data Struct simulée à la Lyra).

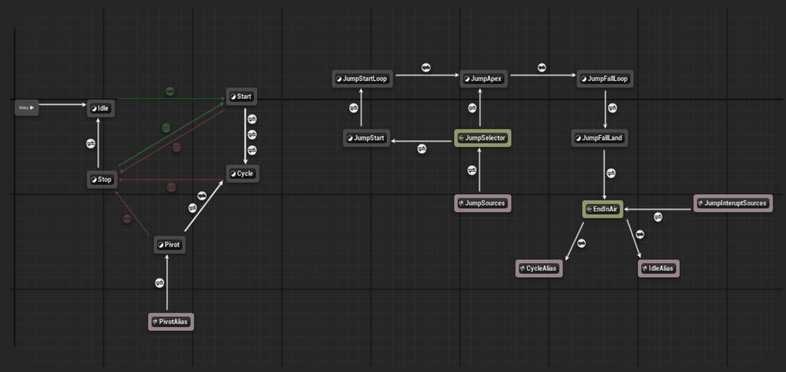
Ce modèle évite les calculs redondants dans chaque layer et centralise la logique.



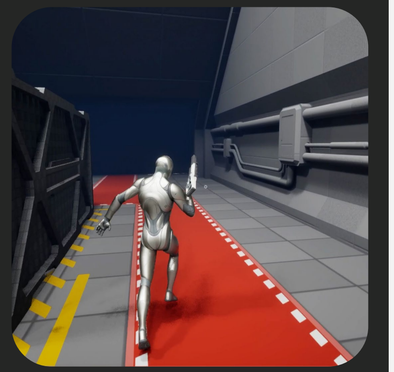
ABP_Ice – Blueprint d’Animation Principal du Joueur :

Ce blueprint contient l’ensemble du système d’animation centralisé du personnage jouable.

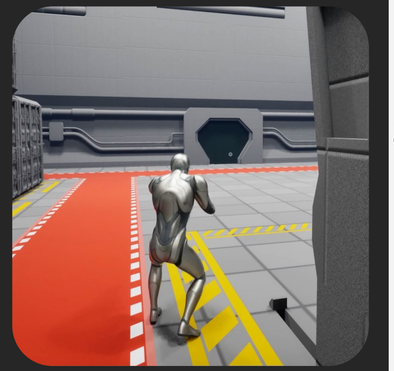
1. State Machine – Locomotion
La state machine principale Locomotion gère toutes les transitions de déplacement (Idle, Walk, Jog, Crouch, Jump...).
Les règles de transition ont été pensées pour garantir une fluidité maximale, sans à-coups ni coupures visuelles.



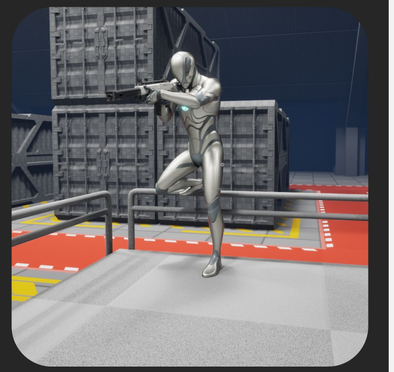
2. Systèmes de Slot – Gestion par parties du corps
Plusieurs slots sont utilisés pour permettre des animations indépendantes sur certaines parties du corps.
Par exemple : le joueur peut recharger une arme tout en continuant à courir.



3. Additive Layer – Animation de Recovery
Une animation additive est jouée à l’atterrissage d’un saut.
Sa puissance est calculée dynamiquement en fonction de la hauteur du saut, pour simuler une récupération plus ou moins intense.



4. IK
Le blueprint intègre un système de Hand IK (placement des mains sur le fusil) ainsi qu’système de Foot IK est utilisé lorsque le personnage est au sol pour ajuster dynamiquement la jambe d’appui selon l’inclinaison du terrain.
Cela permet un ancrage visuel



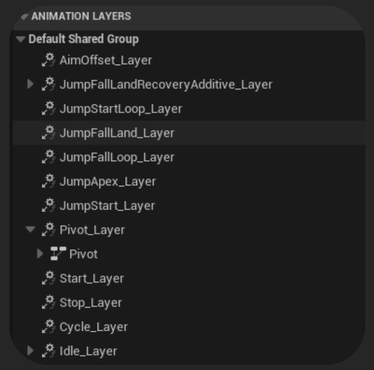
Octopus Rifle

ABP_Layers – Cœur du système d'animation contextuelle :

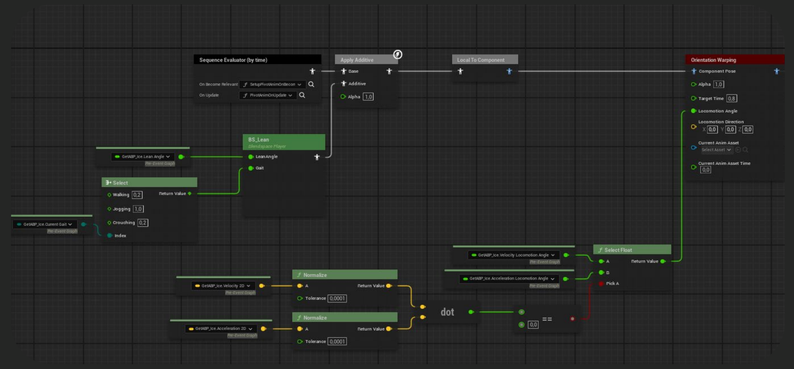
Ce blueprint est responsable de toute la logique d'animation dynamique du personnage.
Il orchestre les poses clés et les blends en fonction de l'état du joueur et de son mouvement (current_gait & direction).

Gère toutes les Linked Anim Layers :

- Idle, Cycle, Stop, Start, Pivot
- JumpStart, JumpApex, JumpFallLoop, JumpLand, JumpRecovery (Additive)
- AimOffset Layer (visée fluide avec blend selon direction & inclinaison)



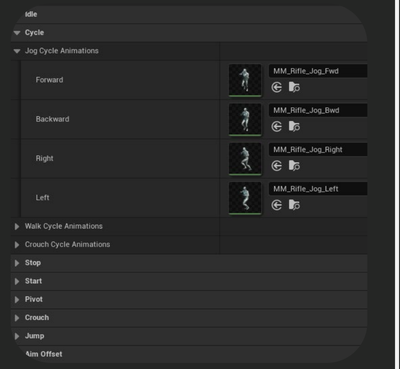
- Initialise et gère les Sequence Evaluators pour des transitions ultra réactives et sans popping.
- Pose des effets dynamiques :
  Lean Additive : pour simuler l'inertie des mouvements.w
  Orientation Warping : pour tourner le haut du corps selon la caméra.
  Stride Warping : pour adapter naturellement la longueur des pas à la vitesse réelle.

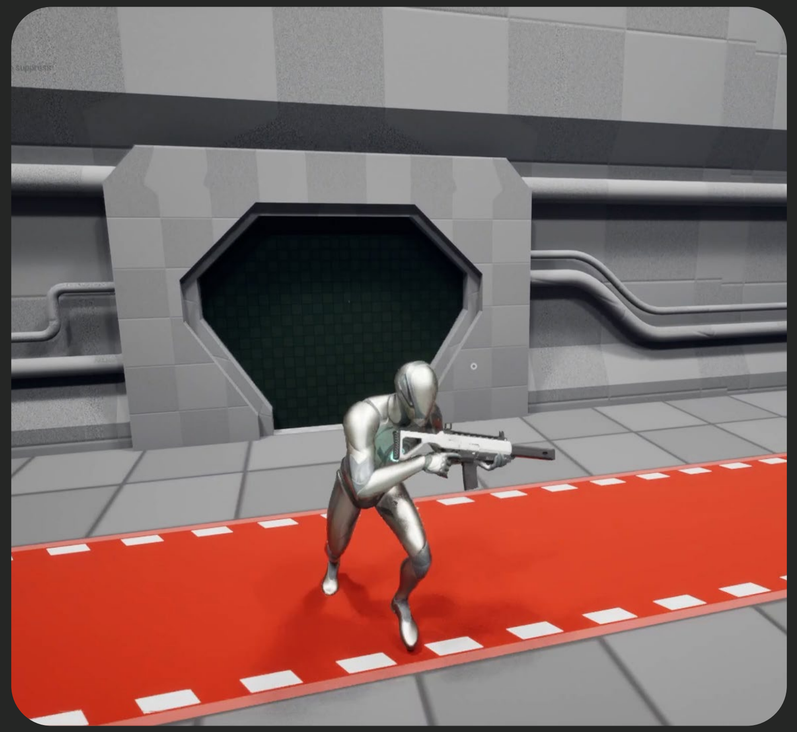


ABP_Rifle & ABP_Unarmed – Spécialisation par équipement :

Ces deux children d'ABP_Layers représentent des variations d'animation spécifiques selon l'équipement porté.

Ces variations sont très simple d'utilisation car il suffit de glisser / déposer les animations voulues dans les différents slots et ABP_ Layers s'occupe de les gérer.





Mot de la fin :

Ce projet a été pour moi une véritable opportunité d'apprentissage.
En m'inspirant directement de projets techniques de référence comme Lyra (Epic Games), j'ai cherché à reproduire un niveau de qualité élevé.

Recréer un setup de personnage complet — intégrant un système de locomotion dynamique, une architecture modulaire d'animation via Linked Anim Layers, et une gestion avancée de l'armement — m'a permis de m'approprier des logiques professionnelles, tout en découvrant en profondeur les subtilités d'Unreal Engine 5.5.

L'exercice a été exigeant, mais extrêmement formateur :
j'ai consolidé mes compétences en animation Blueprint, tout en développant une approche plus rigoureuse de la modularité.

Répliquer des systèmes pensés par des équipes expérimentées est un excellent moyen de progresser, et ce type d'expérimentation restera au cœur de ma démarche de développeur gameplay.

Octopus Rifle

Paul_Hugy

Octopus SF

Prototype développé sur Unreal 5.5
2025

Language :
Blueprint

Rôle :
Experimentation seul

Durée :
Experimentation réalisée sur 1 semaine

Equipe :
Experimentation seul

Objectif :
Itérer sur un système de combat hybride (corps-à-corps + distance) dans un jeu à la troisième personne, avec une gestion avancée des ennemis via IA comportementale.

L'objectif était d'explorer des mécaniques inspirées d'action-RPG modernes, avec une base solide pour enchaîner combos, dash, et interactions IA fluides.

Système de gameplay du joueur – Posture Corps à Corps :

Gestion de la posture Cac (épée) – Automatisation et fluidité UX :

Le système de combat est centralisé dans un Actor Component (AC_ Combat).
Celui-ci gère tout le système de combat et donc également la posture du joueur (Unarmed, Corps à corps ou bien Magique).

Fonctionnement de la posture corps à corps :

- Le joueur peut dégainer ou rengainer son arme manuellement via la touche Tab.
- Ces changements déclenchent des animations spécifiques, et déplacent dynamiquement le mesh de l'arme entre deux sockets du personnage : Holster_Socket (rangement) et Equipped_Socket (combat).
- Pour favoriser une expérience utilisateur fluide, si le joueur tente d'attaquer alors que l'arme est encore rangée, celle-ci est automatiquement dégainée et l'attaque est directement déclenchée.

Objectif : Supprimer les frictions inutiles dans l'enchaînement des actions pour renforcer la réactivité et le ressenti d'attaque.

Système de combo – Gestion dynamique via Animation Notifies :

Le système de combo repose sur une séquence d'animations stockée dans une liste de montages (3 attaques de base).

- À chaque input (clic gauche), si le joueur est en posture épée, l'animation correspondante à l'index courant (AttackIndex) est jouée.
- En cours d'animation, une Notify (AN_ContinuAttack) déclenche une interface vers le AC_Combat, signalant qu'un combo peut être enchaîné.
- Si le joueur appuie à nouveau sur l'input pendant cette fenêtre, AttackIndex est incrémenté et la prochaine animation remplace directement la précédente.
- Si aucun input n'est détecté avant la fin de l'animation, une seconde notify (AN_ResetCombat) remet l'AttackIndex à zéro, réinitialisant le combo.

Cette gestion fine des notifies permet un système précis, réactif et lisible, où chaque attaque ouvre une fenêtre d'enchaînement intuitive pour le joueur.

Système de gameplay du joueur – Posture Magique (distance) :

Posture Magique – Lancer de projectiles & UX fluide :

Le joueur peut entrer en stance magique à l'aide de la touche E.

Lors de cette action :

- La posture précédente est automatiquement enregistrée (Unarmed ou Cac).
- Si le joueur était en posture Cac, l'épée est rengainée automatiquement avant de passer en posture magique.
- Un curseur de visée apparaît à l'écran, signalant le mode distance activé.

À la sortie de la posture magique, le système réactive automatiquement la posture précédente :
Soit en repassant en unarmed, soit en ressortant l'épée si le joueur était en posture Cac.

Une fois en posture magique, le joueur peut lancer des boules de feu dans la direction du curseur.

- Si une cible ou un obstacle est détecté dans l'axe de visée, le projectile s'oriente dynamiquement vers ce point.
- Sinon, il suit la direction par défaut du curseur

Ce système permet d'éviter les décalages visuels entre la position de la main et la cible réelle.

Octopus SF

Système de gameplay du joueur – Mécaniques Complémentaires :

Dash – Mobilité & Cancel offensif :

Le système de dash est pensé comme un outil central de mobilité, permettant au joueur d'esquiver, de se repositionner, ou même de canceller une attaque en cours.

Pendant les premières frames du dash, le personnage bénéficie d'une invulnérabilité temporaire, lui permettant d'éviter des attaques au timing précis.

Dash – Mobilité & Cancel offensif :

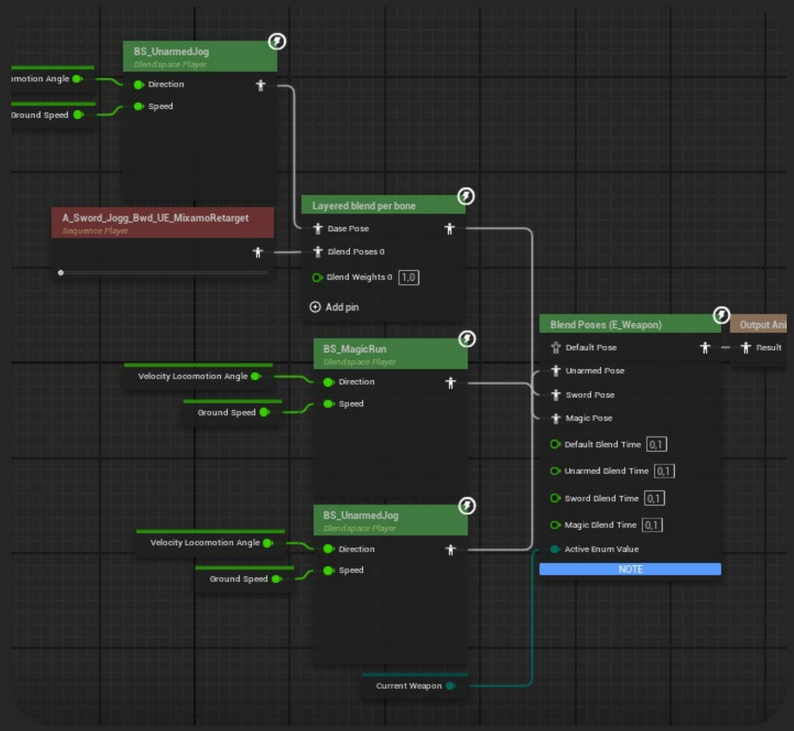
Le système de ciblage est entièrement géré via un Actor Component dédié : AC_LockOn, afin d'être réutilisable et indépendant.
Lors de l'input du joueur :

- Un Multi Sphere Trace est émis depuis la position du joueur.
- L'ennemi le plus proche est automatiquement sélectionné comme cible.
- Un widget de lock est affiché au-dessus de la cible.
- La caméra s'oriente dynamiquement pour suivre la cible tout au long du lock.

Gestion des animations :

Blend Spaces :

Les animations de déplacements sont gérés dans des blend spaces en fonction de la vitesse et la direction du joueur, puis distibué dans l'animation blueprint en fonction de la posture en cours.



Systèmes d'IA comportementale – Prototypes de Behavior Trees :

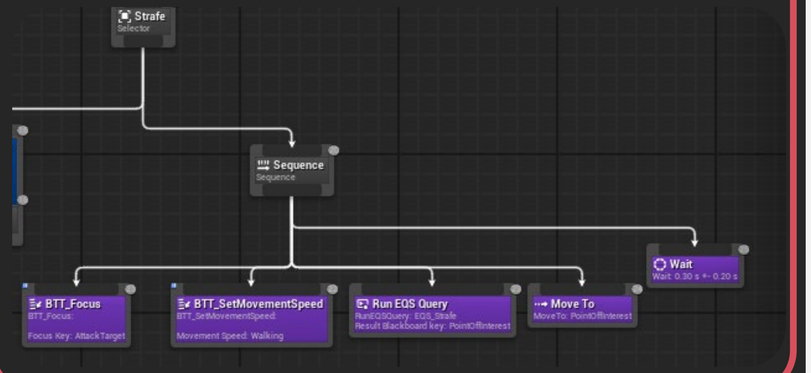
Structure générale :

Dans ce projet, j'ai conçu une architecture comportementale modulaire pour les ennemis, basée sur un Behavior Tree parent générique et des sous-arbres spécialisés pour chaque type d'ennemi (corps-à-corps et distance).

- BT_Enemy_Base : Arbre principal qui gère les états globaux (Passive, Investigating, Attacking, Frozen…) à l'aide d'un système d'enum dans le Blackboard.
- À partir de cet arbre parent, plusieurs sous-arbres sont déclenchés dynamiquement selon l'état courant, permettant une logique claire, évolutive, et facilement maintenable.

IA Melee :

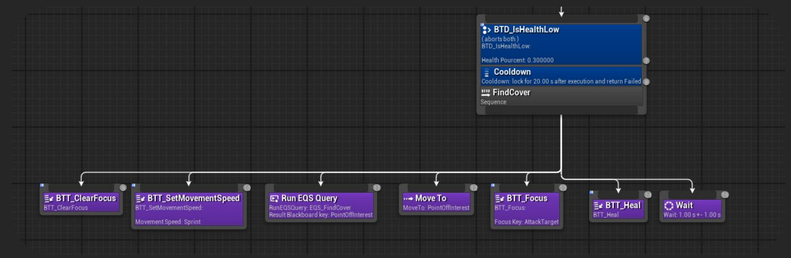
- Implémente un système de focus et de détection de portée idéale.
- Gère l'équipement et le déséquipement de l'arme selon la situation.
- Attaque en séquence, avec une gestion du cooldown et un système de strafes autour du joueur.
- Capable d'interrompre ses actions si la cible meurt ou devient invisible (via services et décorateurs conditionnels).



Octopus SF

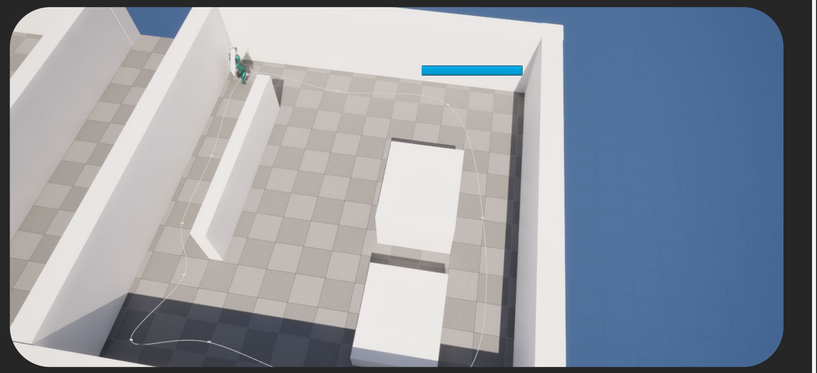
IA Range :

- Se positionne pour maintenir une distance optimale (line of sight, EQS).
- Lance des attaques à distance avec cooldown et variation de pattern.
- Capable de prendre des décisions dynamiques comme se replier et se soigner si ses PV tombent en dessous d'un certain seuil (condition + EQS pour trouver une cover).
- Peut chercher une nouvelle position de tir ou se repositionner si le joueur sort de son champ de vision.



Patrouille :

J'ai mis en place un système de patrouille : l'ennemi suit une spline de point en point. Lorsqu'un de ses sens est activé (vue, dégâts), il se rend sur les lieux pour enquêter, puis engage le combat avec l'opposant



Système de Tokens – Régulation des Attaques IA :

Pour éviter que tous les ennemis attaquent le joueur en même temps, j'ai mis en place un système de tokens d'attaque.
Lorsqu'un ennemi souhaite attaquer, il demande un token à un gestionnaire central (ou conserve le sien s'il est encore en cours d'attaque).

- S'il y a un token disponible, l'attaque est lancée.
- Sinon, l'ennemi patiente ou se repositionne.

Mot de la fin :

L'objectif principal de ce prototype était d'explorer la mise en place d'un système de combat basé sur deux postures complémentaires : corps-à-corps et distance.
Je voulais comprendre comment structurer ce type de gameplay de manière fluide et modulable, avec une UX cohérente et des transitions naturelles entre les postures.

En parallèle, j'ai saisi cette opportunité pour approfondir des notions complexes comme les Behavior Trees, l'EQS, et la logique d'IA comportementale dans Unreal Engine.
Je me suis appuyé sur les enseignements de développeurs expérimentés et de formateurs en ligne pour construire une IA capable de prendre des décisions crédibles et adaptatives.

Ce projet m'a permis de prototyper, tester et itérer rapidement, tout en consolidant des bases solides sur l'architecture de gameplay et les systèmes d'IA.
Une expérimentation précieuse, aussi bien sur le fond que sur la méthode.

Octopus SF

Paul_Hugy

Parcours Dynamics

Prototype développé sur Unreal 5.5
2025

Language :
Blueprint

Rôle :
Experimentation seul

Durée :
Experimentation réalisée sur 3 jours

Equipe :
Experimentation seul

Objectif :
Cette section présente mes recherches sur les mécaniques de traversée et d'interaction avec l'environnement. Escalade, franchissement, déplacements fluides : chaque prototype vise à renforcer l'immersion et la liberté de mouvement du joueur.

Parcours Dynamics

Vaulting avec Motion Warping :

Objectif: Permettre au personnage de franchir automatiquement un obstacle détecté devant lui, avec une animation synchronisée sur les points d'entrée, de passage et de sortie, pour un rendu fluide et réaliste.

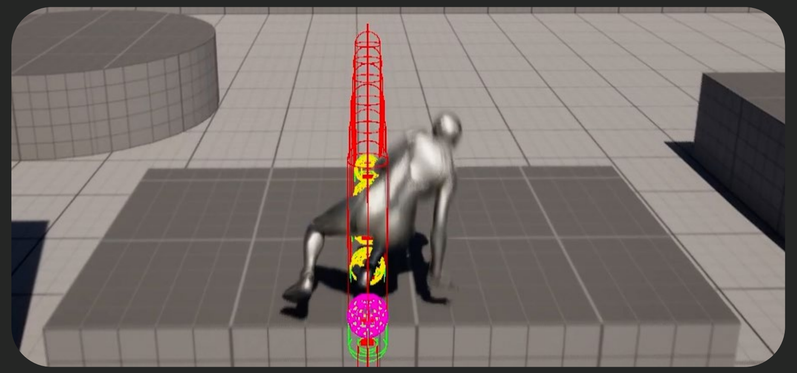
Fonctionnement :
Détéction de Vault :

- Lors de l'appui sur la touche LeftShift, un Sphere Trace est lancé en avant pour vérifier la présence d'un obstacle franchissable.
- Le système vérifie plusieurs conditions (distance, hauteur, validité du hit).
- Si les conditions sont remplies, trois points sont calculés : VaultStart, VaultMiddle, VaultEnd

Motion Warping :

- Ces trois positions sont définies comme des Warp Targets.
- Une animation de vault est ensuite jouée, et automatiquement "warped" sur les cibles grâce au système de Motion Warping Component.
- Cela garantit une animation adaptable à la géométrie rencontrée, quel que soit l'obstacle.

Ce prototype montre comment combiner traces intelligentes et Motion Warping pour créer des déplacements dynamiques, naturels et adaptatifs. Il permet de prototyper rapidement des systèmes de franchissement réalistes sans devoir multiplier les animations précises par situation.



Système de Grimpette – Accroche aux surfaces dynamique :

Ce système permet au joueur de s'agripper à une paroi en face de lui, en vérifiant plusieurs conditions (distance, orientation, statut). Lorsqu'un input est déclenché (IA_Climb), une fonction personnalisée (AttachToSurfaceCalculation) effectue un Line Trace dans la direction du joueur, afin de détecter une surface valide.

Une fois accroché, le joueur peut se déplacer librement sur la surface verticale.

Le mouvement est autorisé via Add Movement Input, et une rotation douce (Lerp + SetActorRotation) aligne le joueur à la paroi pour maintenir l'immersion.

Ce système permet de garantir un contrôle fluide, réactif, tout en assurant la validité physique des déplacements. En cas de perte de surface ou d'input de saut, un appel à StopClimbing restaure la physique normale du personnage.

Dash – Mobilité & Cancel offensif :

Détection de rebord via trace linéaire, suivie d'un calcul de position cible (vault start / middle / end).
Trois points de warp sont générés dynamiquement pour guider précisément l'animation de montée via Motion Warping.

Objectif : rendre la montée d'un obstacle naturelle, lisible, et systématiquement réutilisable dans d'autres contextes (sauts contextuels, escalade libre...).

Mot de la fin :

J'ai toujours été attiré par les jeux offrant une grande liberté de déplacement, comme Assassin's Creed ou Dying Light.
C'est donc naturellement que je me suis lancé dans l'exploration de ces systèmes de traversal.

À travers de nombreux tutoriels, prototypes et expérimentations, j'ai cherché à comprendre les fondations de ces mécaniques : détection de surfaces, orientation dynamique, animation contextuelle, motion warping...

Ce que je présente ici, c'est à la fois un terrain de test et un espace de progression.

Paul_Hugy