Antoine Leroux - B3GD2

Death Canyon

Informations :

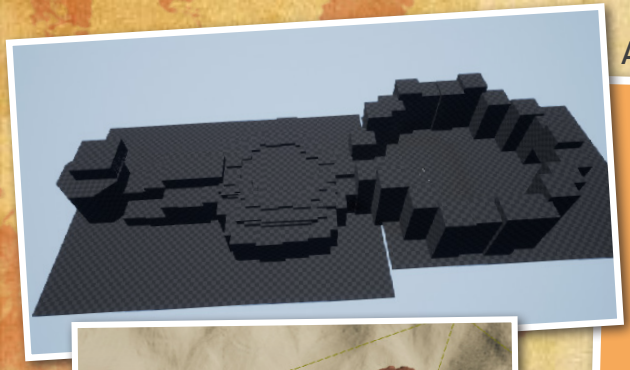
- **Version :** Unreal Engine **4.25.4**
- **Plugins installés :** Apex Destruction
- **Membre :** **Antoine LEROUX**

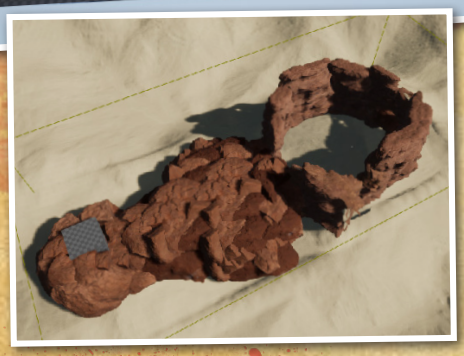
Résumé du développement :

Pour commencer, le projet de conception d'un prototype sur Unreal Engine m'a tout de suite intéressé.
En effet, ayant touché au moteur il y a **3 ans** pour faire un petit jeu, je ne **m'y connaissais pas** encore en programmation et je n'avais **pas les connaissances** de développement de jeux vidéo que j'ai aujourd'hui ; c'est pourquoi avoir **un regard nouveau** sur ce moteur me **plaisait fortement**. De plus, ça a été synonyme de changement pour moi qui avait l'habitude d'utiliser uniquement **Unity** comme moteur de jeu afin de concrétiser mes idées de conception.

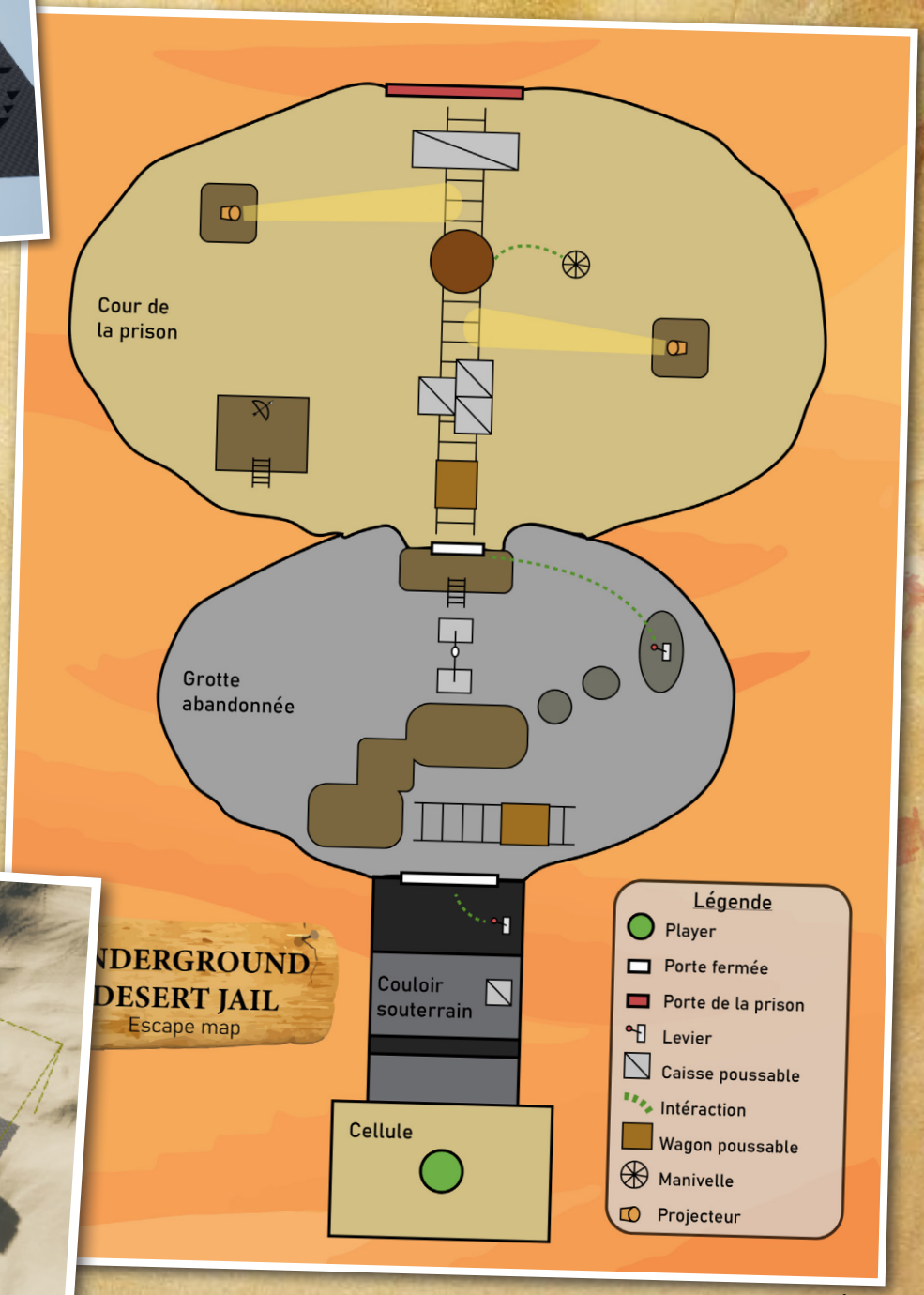
À titre personnel, les cours d'Unreal m'ont permis d'avoir une **base suffisante** pour réaliser l'exercice et ses contraintes. Cependant, **j'ai souhaité pousser l'exercice légèrement plus loin**, notamment au niveau du **blocking** et de **la scénarisation** pour répondre aussi à l'**exercice d'Alexandre** et, également, pour prendre davantage en main le moteur sous tous ses aspects pour **mes futures années** et pour développer ma polyvalence.

C'est ainsi qu'a débuté le développement de mon prototype sur Unreal Engine. J'avais **mon idée de prison du premier semestre** ; mon niveau me semblait réalisable assez facilement car j'avais posé en amont mon Level Design sur croquis. Mais comme notifié ci-dessus, il me fallait enrichir mon expérience de jeu pour l'exercice d'Alexandre. C'est ainsi que j'ai décidé de pousser l'étape du blocking plus loin et de modéliser le plus possible celui-ci. Lié à des éléments de décorations et de scénarisation, mon niveau dans son entièreté pourrait **répondre à la fois aux deux exercices**.



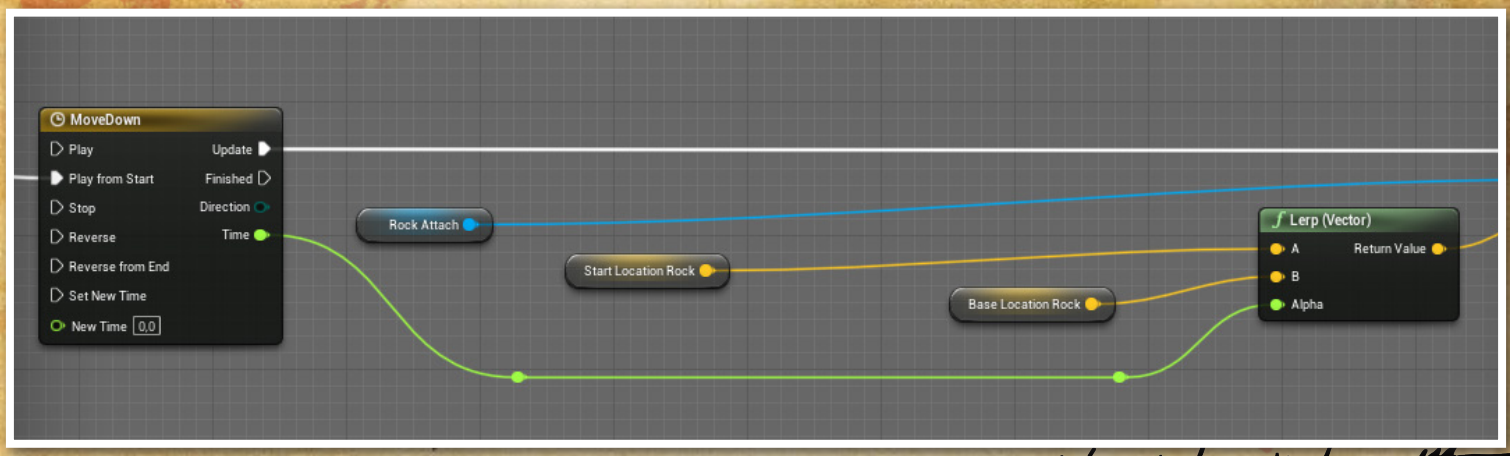


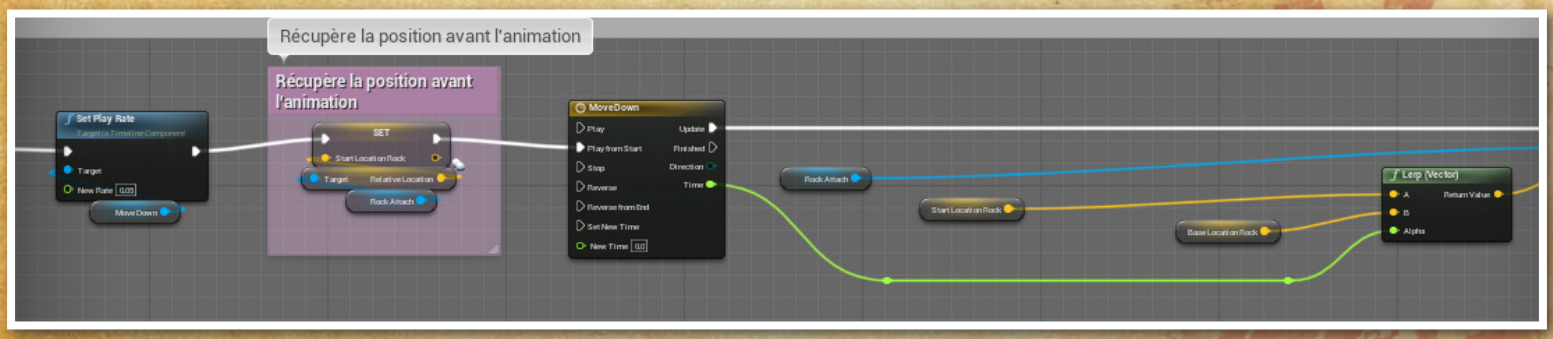
Croquis Level Design



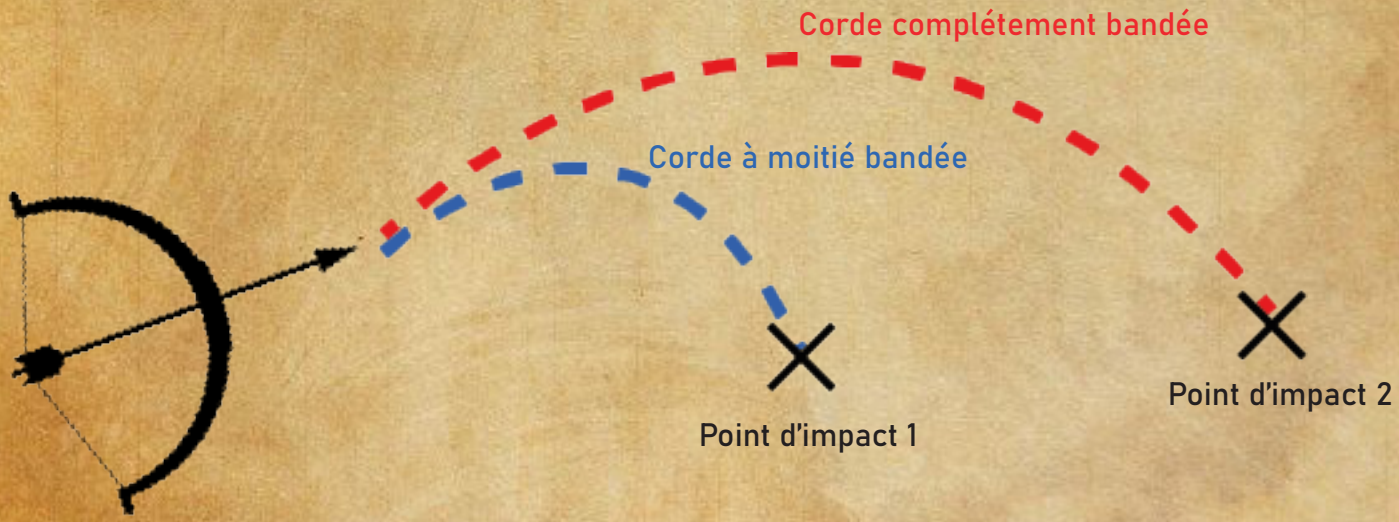
Le développement a commencé et j'ai rapidement compris la **logique du Blueprint**, grâce à son interface très **user-friendly**. J'ai pu réaliser **mon controller** et les **comportement des éléments de jeu** en moins de temps que je ne pensais, tout en enrichissant mes connaissances sur ce langage de programmation en **Visual Scripting**.

Mon **plus gros problème en Blueprint** a été pour **déplacer un objet d'un point A à un point B** en un **temps donnée** spécifique. Bien sûr j'ai utilisé **les timelines** avec une float de 0 à 1 en un temps x pour réaliser une interpolation entre 2 valeurs de coordonnées de déplacement. Mais **aussi grand soit ce temps x**, l'interpolation se réalisait tout-de-même **en 1 seconde**. Néanmoins, j'ai trouvé une solution qui est de **set le play rate** de la timeline avec une faible valeur pour augmenter le temps que prend l'interpolation, mais **je ne suis pas ravi** de cette solution qui fait effet de tampon. J'ai découvert après coup, que je pouvais également utiliser des components pour déplacer un objet (**InterpToMovement**) mais **uniquement pour un actor** il me semble et non pour un mesh qui est enfant de l'actor par exemple.



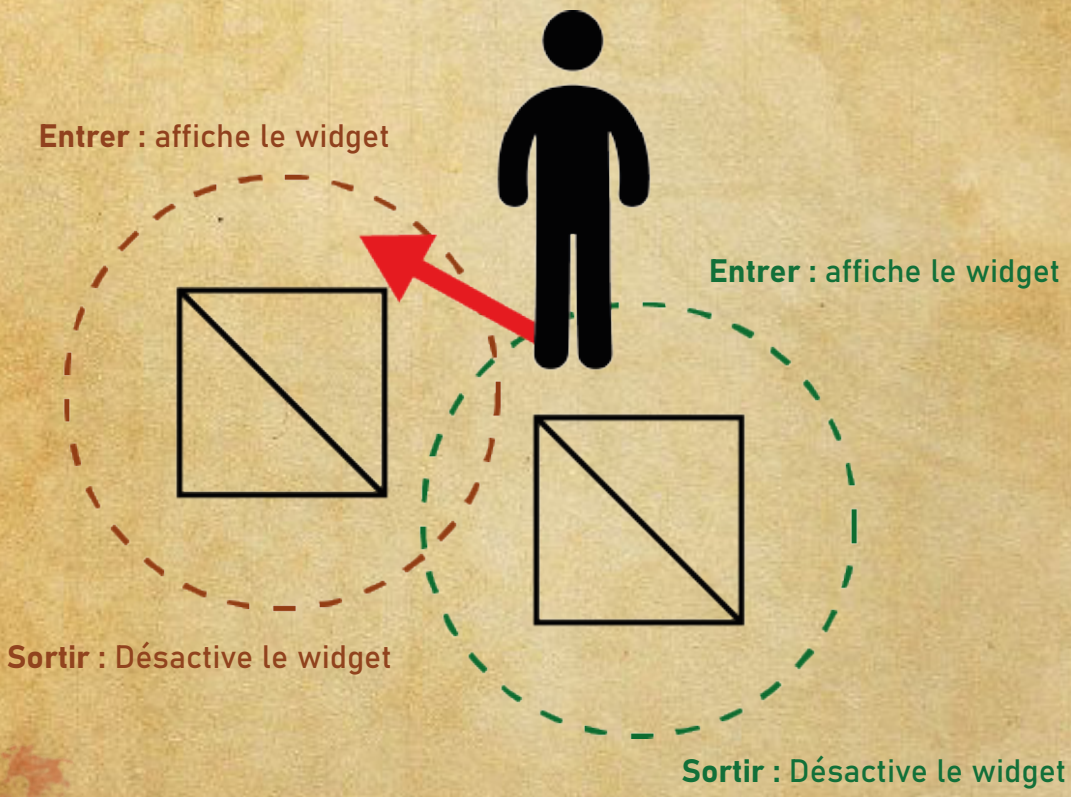


Ma **mécanique principale**, **l'arc**, a été compliquée à développer, **surtout à la 3ème personne** car je devais faire les **animations du personnage** lorsque l'arc est équipé, mais également les **animations de l'arc**, de **la corde** qui se bande, etc... Cette tâche a été assez **fastidieuse** et j'ai du me débrouiller. Le résultat du comportement de l'arc **n'est pas celui que je souhaitais** mais il fonctionne. Voici tout-de-même un croquis de ce que j'aurais souhaité.

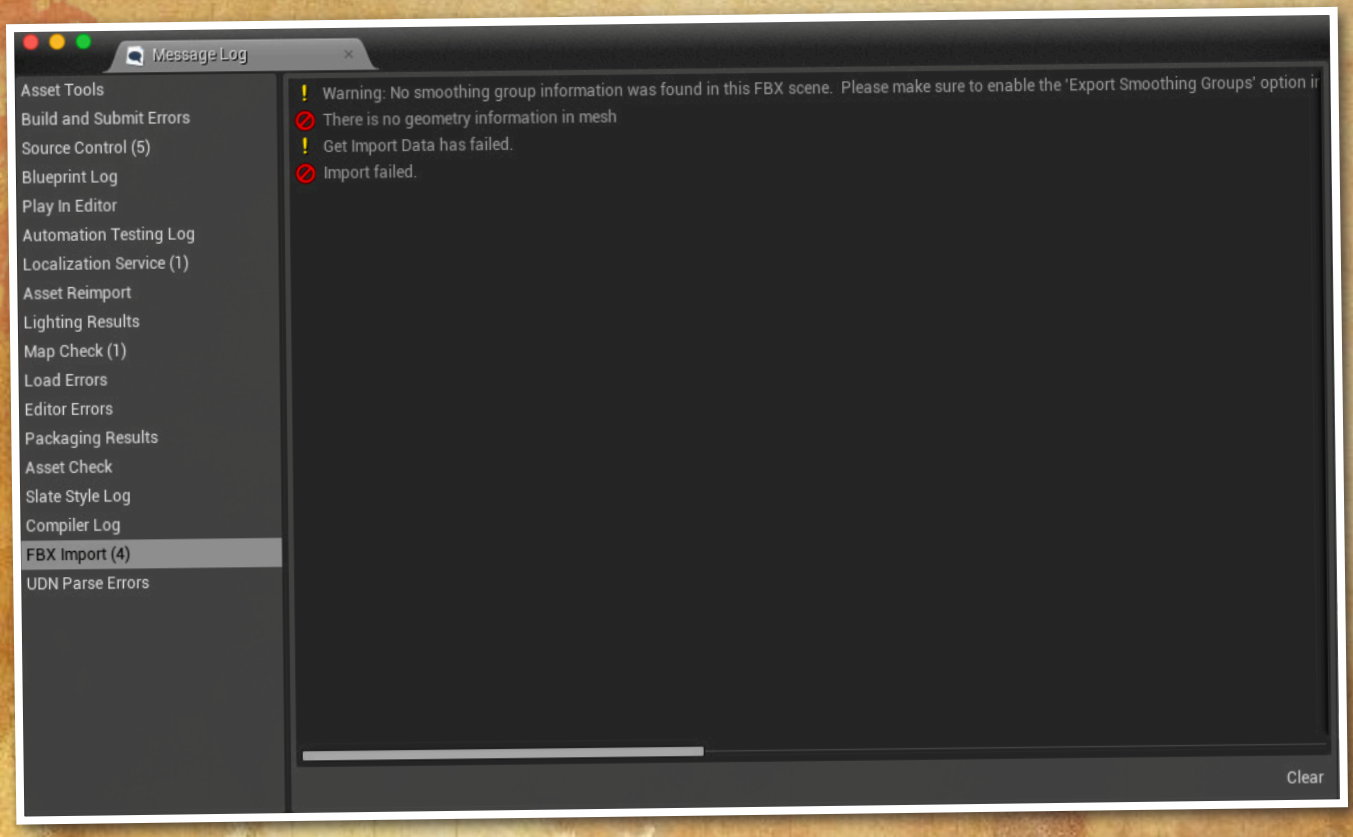


Comme on peut le voir sur le schéma, j'avais pour **intention** que le player puisse tirer avec son arc quand il le souhaite, **peu chargé** ou **complètement chargé**, ce qui représente, en terme d'input, l'**appui plus ou moins long sur le clic gauche** de la souris à l'image d'une barre qui se remplit. Ainsi la **flèche** aurait eu de l'**inertie** et le challenge du joueur aurait été de savoir bien doser son arc au fil du jeu. Le système pour lequel j'ai opté, **par facilité** est le fait de **tracer un raycast** où le player vise et d'envoyer une flèche à cette destination en **ligne droite** ; autrement dit, celle-ci n'est **pas physiquée**.

Enfin, concernant le Blueprint, j'ai eu des **problèmes mineurs** comme afficher uniquement un widget spécifique lorsque le **personnage entre en collision avec des actors d'un certain tag**. En effet cela marche, mais lorsque je **sors de la collision** pour désactiver le widget et que je suis **encore en collision avec un autre actor** du même tag, le **widget se désactive** quand-même ; je ne sais pas quelle aurait été la solution.



Pour la **partie visuelle**, j'ai rencontré beaucoup de problèmes avec **l'importation de models 3D gratuits**. J'ai par exemple eu ce problème à chaque importation qui m'**obligeait à relinker les textures au matérial** moi-même.



J'ai ensuite découvert **Quixel Bridget** qui m'a facilité la vie pour trouver des models et des matérials et leur **importation automatique** sans relink nécessaire ! Mais je n'ai **pas vu les problèmes** que ce dernier a apportés dans mon projet. N'ayant pas fait attention aux paramètres d'exportation au début, j'apportais des **textures 8k** à chaque fois, ce qui **était inutile** dans mon cas. J'ai très vite modifier cela pour la plus basse résolution : tout de même 2k ! Ensuite, j'ai découvert que **chaque model** 3D avait **plusieurs LOD** lors de l'importation, ce n'était également pas utile dans mon cas et j'y ai vite remédié. J'ai quand même eu une situation cocasse lorsque j'ai découvert que les models avaient plusieurs LOD car en effet, je ne comprenais pas pourquoi ils rétrécissaient lorsque j'étais loin d'eux et s'agrandissaient au contact.

Enfin, une **grosse partie de mon temps** s'est déroulée sur le **post-process** que je n'ai pas réussi à fignoler et cela est **dommage** car je trouve que celui-ci et les lights **ne rendent pas hommage à mon niveau**. N'étant déjà pas doué sur Unity pour le post-process, cela se répète sur Unreal ; je ne comprends pas tous les paramètres et c'est plus un **mélange d'équilibrage** de ma part pour obtenir un rendu qui me convient à chaque fois. **Il faut ainsi que je m'améliore sur ce point**.
J'ai par exemple eu le dilemme entre choisir un **template de level avec une directional light "classique"** mais n'éclairant pas l'intérieur de mes grottes et **un template de level avec un time of day** qui donne un excellent rendu en extérieur, mais les ombres n'étaient pas assez présentes à mon goût. Cela provient sûrement de paramètres à modifier, mais vu que le post-process était ma dernière étape de développement, je n'ai clairement **pas eu le temps** de me documenter là dessus.

Pour **résumer le développement,** la **programmation** n'a été qu'une **petite partie** de mon temps car facilitée grâce au Blueprint. L'**importation**, l'**assemblage** et le **placement** des **models 3D** m'ont bien pris **40% de mon temps** avec le **Level Design** et le **décor** du niveau. Enfin, l'**UI**, le **son**, les **lights**, le **post process** et les **Blueprints de transition de niveau**, de **victoire** et de **défaite** ont occupé **le reste de mon temps** de travail.

Aide de jeu :

Echap : Quitter le jeu

Tab : Sortir l'arc

Z : Avancer

S : Reculer

P : Mettre en pause

Clic droit : Bander l'arc / Tirer

Shift : Courir

C : S'accroupir

Q : Aller à droite

D : Aller à gauche

Espace : Sauter