

1.5em0pt

École Polytechnique de l'Université de Tours
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr



Mise en œuvre collective

Compte-rendu

Enseignant

Sébastien Aupetit

sebastien.aupetit@univ-tours.fr

Étudiants

David Boas

david.boas@etu.univ-tours.fr

Nicolas Fontaine

nicolas.fontaine@etu.univ-tours.fr

Guoxiang Liu

guoxiang.liu@etu.univ-tours.fr

Charlie Mathonnet

charlie.mathonnet@etu.univ-tours.fr

Bastien Meunier

bastien.meunier@etu.univ-tours.fr

Armand Renaudeau

armand.renaudeau@etu.univ-tours.fr

Li Wenlong

li.wenlong@etu.univ-tours.fr

DI5 2014 — 2015

Version du 16 mars 2015

Table des matières

1	Introduction	2
1.1	Mise en œuvre collective de réalité virtuelle	2
1.2	Slender : The Eight Pages	2
1.3	PolySlender	2
1.4	Mise en place du PolySlender	3
2	Modélisation	4
2.1	Modélisation des étages et des objets	4
2.1.1	Organisation des graphes de scène	4
2.1.2	Modélisation des objets	5
2.1.3	Modélisation des étages	6
2.2	Matériaux et textures	6
2.2.1	Création des images de texture	7
2.2.2	Application d'une texture à un objet : l'UV Mapping	8
3	Scène Unity	10
3.1	Environnement et ambiance	10
3.1.1	Ambiance lumineuse	10
3.1.2	Particules et effets visuels	11
3.1.3	Ambiance sonore	14
3.2	Animations	14
3.2.1	Les différents types d'animations	14
3.2.2	Cinématique d'introduction	16
3.2.3	Effet de grésillement de l'écran	17
3.2.4	Robots	17
3.3	Les interactions avec l'environnement	18
3.3.1	Objets gérés par le moteur physique	18
3.3.2	Portes	18
3.3.3	Collisions	20
3.4	LOD	20
3.4.1	Simplification des meshes	21
3.4.2	Le script du DLOD	21
4	Le gameplay	22
4.1	Contrôle du personnage	22
4.1.1	Caméra	22
4.1.2	Gestion de la vie	23
4.2	Fonctionnement du Slenderman	23
4.2.1	IA	23
4.2.2	Du côté d'Unity3D	25
4.3	Les objets à ramasser	27
4.3.1	Apparence des objets	27



4.3.2	Le placement des objets	28
4.3.3	Récupération des objets et gestion de l'inventaire	29
4.3.4	Conditions pour ramasser un objet	29
4.3.5	Gestion de l'inventaire	29
5	Gestion de projet	30
5.1	Travail en collaboration	30
5.2	Outils de gestion de projet	31
5.2.1	Trello	31
5.2.2	Google Drive	31
5.2.3	Guides	32
5.3	Outil de versioning	32
6	Conclusion	33
A	Annexe A : Guide de modélisation	34
B	Annexe B Guide d'utilisation de Git	40
C	Annexe C : Guide de pose des textures	45

Table des figures

2.1	Graphe de scène d'un même objet sur Blender et sur Unity	5
2.2	Système métrique ("Metric") sur Blender	6
2.3	Plan du rez de chaussee	6
2.4	Exemples de textures créées et utilisées dans le cadre de ce projet	7
2.5	Traitement d'une image pour créer une texture répétable	8
2.6	Etapes de l'UV Mapping	9
3.1	Comparatif entre une lumière directionnelle et spotlight	10
3.2	Comparatif entre une lumière avec et sans cookie	11
3.3	Cookie de la lampe torche	11
3.4	Capture d'écran du système de particules "pluie"	12
3.5	Configuration du système de particules "pluie"	12
3.6	Capture d'écran du système de particules "brouillard"	13
3.7	Capture d'écran du système de particules pour les objets	13
3.8	Capture d'écran de l'outil "Timeline"	14
3.9	Capture d'écran de l'outil "Dope Sheet"	15
3.10	Capture d'écran de l'outil "Animation"	15
3.11	Capture d'écran d'un exemple d'Animator	16
3.12	Aperçu de la cinématique d'introduction	16
3.13	Création de la cinématique d'introduction	17
3.14	Capture d'écran du composant physique "Box Collider"	18
3.15	Capture d'écran du composant physique "Rigidbody"	19
3.16	Capture d'écran d'un exemple d'Animator	19
3.17	Visualisation de l'animation d'ouverture de porte avec collision entre BV	20
3.18	LOD	21
4.1	Personnage jouable du jeu	22
4.2	Zone d'action du slender	25
4.3	Detection du champ de vision	25
4.4	NavMesh	26
4.5	Objets à ramasser dans le jeu original et dans PolySlender	28
5.1	Etat de notre Trello vers la fin du projet	31
5.2	Etat du GitHub vers la fin du projet	32

16 mars 2015

Introduction

1.1 Mise en œuvre collective de réalité virtuelle

Dans le cadre de notre 5ème année d'études d'ingénieurs en informatique à Polytech Tours, nous avons choisi l'option de dernière année "Réalité Virtuelle". Cette option prévoit la réalisation en groupe d'un projet interactif en 3D dont le sujet est libre.

Nous avons choisi de créer un jeu dans la thématique de la réalité virtuelle. Nous avons donc voulu créer un jeu mettant l'accent sur l'immersion du joueur. Le jeu se déroulerait dans un lieu réel et connu des utilisateurs potentiels du jeu : le Département Informatique. Comme nous souhaitions développer au maximum l'immersion dans le jeu, notre choix s'est porté sur un jeu d'horreur. C'est ainsi que nous avons décidé de créer une nouvelle version du jeu Slender : The Eight Pages.

1.2 Slender : The Eight Pages

Slender : The Eight Pages est un jeu d'horreur devenu très populaire voire même viral lors de sa sortie en 2012 car c'est un jeu gratuit, dont l'histoire est basée sur une légende urbaine, et particulièrement immersif.

Dans ce jeu, on incarne un personnage à la première personne qui doit visiter une forêt très sombre, munie uniquement d'une lampe torche. Le but du jeu est de trouver 8 objets répartis aléatoirement dans la forêt. À part le déplacement du personnage principal, les seules actions possibles sont de se mettre à courir, ce qui abaisse la lampe torche, ou de ramasser une page qu'on a trouvé. L'ambiance du jeu est de plus en plus oppressante au fil de la progression, grâce à une ambiance sonore stressante et qui évolue, une luminosité quasi inexistante et la présence d'un personnage mystérieux : Slenderman, issu d'une légende urbaine. Il ne faut pas se faire attraper par Slenderman, qui a la possibilité de se déplacer vers le joueur, de se téléporter discrètement, et qui produit divers effets dérangeants (un son brusque qui fait sursauter, un grésillement de l'écran qui indique qu'il est proche même si on ne le voit pas).

1.3 PolySlender

Notre objectif était de recréer le jeu Slender : The Eight Pages de façon fidèle : nous voulions garder les mêmes contrôles, la même ambiance, les mêmes principes de jeu. Cependant nous avons modifié l'univers dans lequel se déroule l'histoire pour l'adapter au lieu dans lequel nous avons cours tous les jours de la semaine, peut-être dans l'espoir d'occasionner quelques frayeurs à un étudiant de Polytech Tours qui aurait le malheur de se promener dans les couloirs après la tombée de la nuit après avoir joué à notre jeu. Le personnage qu'on incarne est un étudiant qui doit absolument récupérer ses affaires, qu'il a oubliées à l'école, avant le lendemain. Il a un entretien très important pour lequel il doit apporter son bulletin de notes, et son TOEIC. Il doit aussi récupérer son sac de cours, son disque dur, son certificat de scolarité et sa ventoline. Et pour cela, il lui faudra également sa carte d'étudiant, car certaines portes du bâtiment sont badgées.

1.4 Mise en place du PolySlender

Notre but pour notre projet est donc de recréer le jeu du Slender dans l'univers du Département Informatique de Polytech Tours. Pour se faire, nous sommes passés par plusieurs étapes que nous allons vous présenter au travers de ce rapport.

Tout d'abord, nous commencerons par vous parler de toute la partie concernant la modélisation de l'école. C'est une partie qui a représentée une grosse partie de notre projet car nous voulions que l'immersion soit la meilleure possible. Nous avons donc tenté de modéliser le bâtiment en étant le plus précis possible.

Ensuite, nous vous présenterons l'univers que nous avons mis en place au travers de l'outil Unity3D. Nous vous parlerons donc des différentes méthodes que nous avons mis en place afin de réaliser notre environnement de jeu et nous vous parlerons des principaux éléments qui composent le jeu.

Nous continuerons ensuite sur la présentation et le fonctionnement du gameplay de notre jeu. Pour cela, nous commencerons par décrire les principales caractéristiques du personnage et ensuite, nous ferons de même pour l'ennemi. Pour finir, nous vous parlerons de comment nous avons réparti les tâches au cours du projet et comment nous l'avons gérer.

Modélisation

2.1 Modélisation des étages et des objets

2.1.1 Organisation des graphes de scène

Sur Blender

Fonctionnement Un fichier au format Blender (.blend) contient une ou plusieurs scènes qui organisées par un graphe de scène. Pour ce projet il ne nous était pas utile d'inclure plus d'une scène par fichier Blender.

Les graphes de scène sur Blender sont classiques : chaque noeud du graphe de scène contient une transformation locale, un éventuel objet (mesh, armature...) et les données qui y sont associées, et des éventuels noeuds enfants. C'est l'utilisateur qui choisit l'organisation du graphe de scène lors de la modélisation de la scène. Nous avons donc organisé nos graphes de scène de façon à faciliter notre travail.

Organisation Pour les objets, nous avons en règle générale préféré fusionner notre objet en un seul mesh, même s'il était composé de plusieurs parties non connectées. Les raisons pour lesquelles nous avons fait ce choix sont les suivantes :

- nous souhaitions simplifier les graphes de scène en réduisant le nombre de noeuds dans lesquels nous pouvions naviguer
- nous avons remarqué que les performances de l'éditeur Unity et du jeu étaient meilleures lorsque le graphe de scène comprenait moins de noeuds, à géométries égales

Pour les étages, nous avons créé un noeud comprenant toute la géométrie des sols et des escaliers. Nous avons choisi cette organisation afin de facilier la création d'un NavMesh sur Unity (référence). Pour les autres objets, nous les avons organisés spatialement, afin de tirer bénéfice des volumes englobants et des différents types de culling.

Sur Unity

Fonctionnement Unity utilise le même type de graphe de scène que Blender, et lors de l'importation d'un fichier au format Blender (.blend) dans un projet Unity, Unity crée un nouveau noeud dans son graphe de scène dont les enfants sont les noeuds de la scène importée. On conserve donc la même architecture.

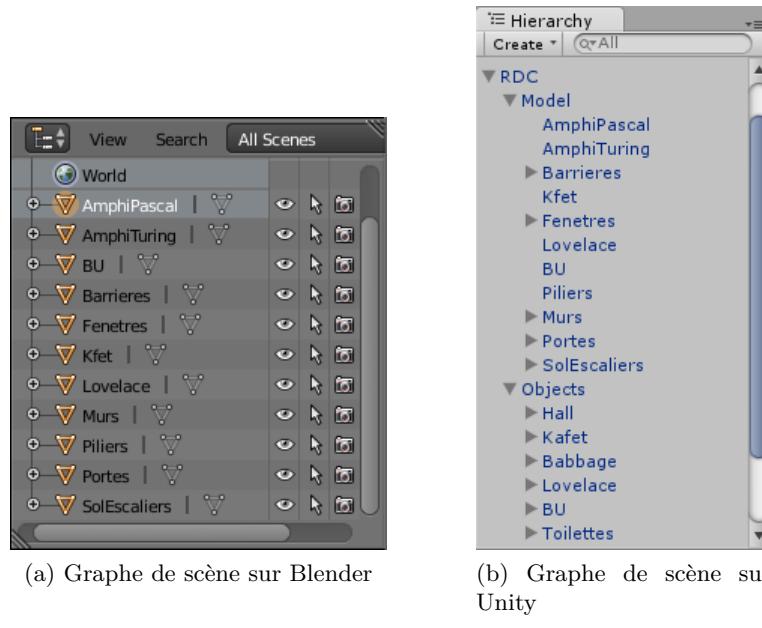


FIGURE 2.1 – Graphe de scène d'un même objet sur Blender et sur Unity

Organisation Nous avons tiré profit de la gestion du graphe de scène que propose Unity pour différentes choses :

- organiser logiquement notre scène afin de trouver rapidement les objets que nous cherchons
- séparer des éléments indépendants afin de pouvoir travailler en parallèle sur plusieurs objets (référence, gestion de projet ? prefabs git...)
- grouper des éléments qui sont indissociables dans le jeu. Par exemple, la lampe de poche est un enfant du personnage principal, celle-ci subit donc les mêmes transformations que son possesseur
- séparer les éléments importés à partir du fichier Blender et les éléments rajoutés sur Unity. Nous avons été préféré effectuer cette séparation car nous avons rencontré des problèmes lors de la mise à jour de la scène sur Blender : il était parfois nécessaire de supprimer puis réimporter l'objet sur la scène. Séparer le modèle Blender des objets ajoutés sur Unity nous a permis d'effectuer ces remplacements sans perdre le travail effectué sur Unity.

2.1.2 Modélisation des objets

Nous avons modélisé nous-mêmes une partie des objets, comme des chaises, des tables, des tableaux, etc... et nous avons récupéré une partie des modèles sur Internet, comme la machine à café, les toilettes, les poubelles... Pendant cette phase de modélisation, nous avons utilisé un système de mesures réelles. Pour cela, nous avons utilisé une fonctionnalité de Blender qui permet de spécifier les dimensions des objets en mètres. Cela nous a permis de modéliser tous les objets à l'échelle, en utilisant la même échelle pour tout le monde sans ambiguïté.

Nous avons utilisé les paramètres métriques ("Metric") suivants : l'unité de base est "1 meter" (1 mètre) et l'échelle ("scale") est de 1.000, ce qui équivaut aux unités Blender suivantes :

- 0.100 = 1cm (1 centimetre)
- 1.000 = 1m (1 mètre)
- 1000.000 = 1km (1 kilomètre)

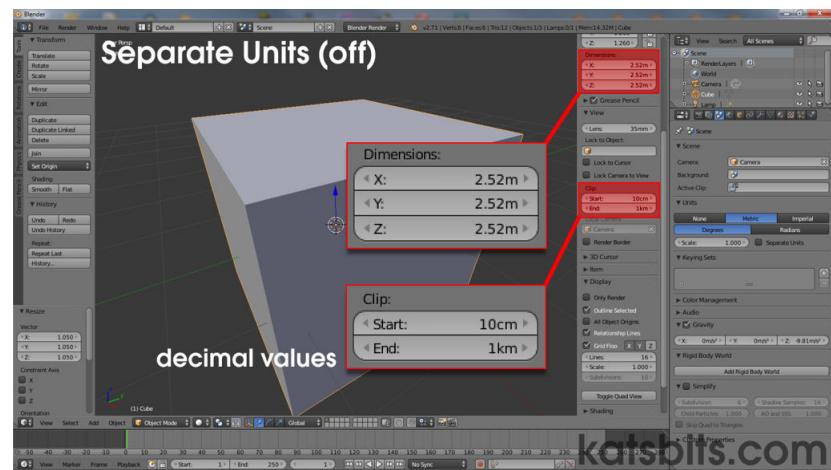


FIGURE 2.2 – Système métrique ("Metric") sur Blender

2.1.3 Modélisation des étages

Nous avons modélisé le bâtiment à partir de plans réels tels que le plan ci-dessous. Nous avons ensuite pris des mesures dans le bâtiment pour compléter les informations dont nous disposions. Cela nous a permis, d'une part, de modéliser le bâtiment avec plus de facilité pour respecter les échelles, mais aussi de coordonner notre travail afin que les trois étages, modélisés par des personnes différentes, se superposent sans incohérences.

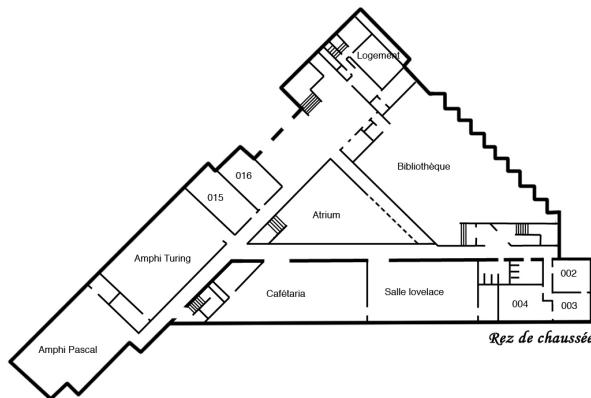


FIGURE 2.3 – Plan du rez de chaussee

La méthode que nous avons utilisé pour modéliser le bâtiment à partir des plans a été de construire d'abord les murs en 2D en les calquant sur le plan, puis d'effectuer une extrusion en hauteur afin d'obtenir le volume des murs. Nous avons ensuite, à l'aide de subdivisions et d'ajustement des points, ajouté les détails aux murs, tels que les encadrements de porte, les renforts de murs...

2.2 Matériaux et textures

La modélisation des objets est la première étape de leur création. Il faut ensuite leur appliquer un matériau et/ou une texture. Dans notre cas, la majorité des surfaces que nous souhaitons représenter

n'étaient pas de couleur unie, nous avons donc presque exclusivement utilisé des textures avec un matériau simple et peu coûteux à rendre.

Une texture est une image représentant une surface offrant la possibilité de simuler l'apparence de celle-ci quand on la colle sur un objet 3D. Les textures sont particulièrement utilisées généralement dans les jeux-vidéos, et les textures offrent un aspect proche de la réalité. Dans notre projet, nous avons pris des photos comme la texture.

2.2.1 Crédit des images de texture

Récupération de l'image d'origine

Afin de rendre le plus réaliste et fidèle possible l'environnement, nous avons créé la grande majorité de nos textures à partir de photos des différentes surfaces du Département Informatique. Ainsi, nous avons répertorié les principaux motifs que l'on pouvait voir au sol, sur les murs, au plafond, les différents panneaux... puis nous avons pris des photos de chaque motif à partir desquelles on a créé nos textures.



FIGURE 2.4 – Exemples de textures créées et utilisées dans le cadre de ce projet

Pour certaines textures, telles que les panneaux récurrents du bâtiment (interdiction de manger ou de boire, plans d'évacuation...) ou les images des documents à ramasser, il nous a suffit de prendre une photo et d'en ajuster l'exposition pour pouvoir ensuite directement l'utiliser comme texture. Mais pour d'autres surfaces telles que les murs, sols plafonds, nous avions besoin de textures répétables. Il nous a donc fallu traiter les images que nous avions prises afin que leur répétition paraisse naturelle.

Aussi, afin de simplifier la pose des textures, nous avons utilisé comme norme des images représentant une surface mesurant 1m x 1m. Cela nous a évité de devoir faire de nombreux ajustements de taille lors de l'affectation des textures aux coordonnées UV.

Traitement de l'image

N'importe quelle image peut être utilisée pour produire une texture "répétable". Cependant, la démarcation entre chaque répétition de l'image risque d'être visible, ce qui ne rend pas naturelle du tout cette répétition. Par abus de langage, nous avons utilisé le terme "répétable" pour qualifier un motif que l'on peut répéter pour créer une texture uniforme, sans effet "patchwork". Pour éviter cet effet, on a suivi les étapes suivantes :

- prendre des photos dans des zones assez éclairées et où l'éclairage est uniforme. Cela n'a pas toujours été possible car nous avons pris ces photos en intérieur, il a donc fallu corriger les inégalités d'exposition
- recadrer et redimensionner l'image afin qu'elle représente 1m x 1m de surface et qu'elle ait une résolution adaptée à l'objet sur lequel elle sera posée. Une texture qui sera visible de près aura une bonne résolution, alors qu'une texture dont un ne peut s'approcher peut avoir une résolution plus faible pour économiser des ressources lors du rendu de la scène.
- corriger l'exposition : grâce à un logiciel de retouche photo, on applique un filtre à l'image afin que l'exposition de celle-ci soit uniforme sur toute l'image
- lisser les bords de l'image afin de ne pas voir d'effet "patchwork"

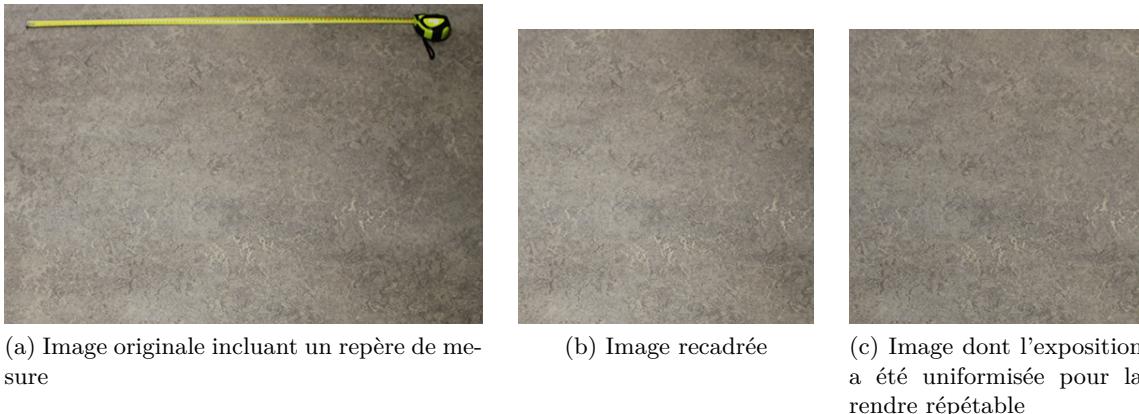


FIGURE 2.5 – Traitement d'une image pour créer une texture répétable

2.2.2 Application d'une texture à un objet : l'UV Mapping

L'application d'une texture à un objet 3D nécessite de passer par 3 étapes : la création de la texture, le dépliage des UVs ou UV Mapping, et l'application de la texture.

L'UV mapping est un procédé qui permet de représenter en 2D la surface d'un modèle 3D. Cela permet de projeter une image sur un objet 3D. Les lettres «U» et «V» représentent les axes de la texture 2D. En effet, les lettres "X", "Y" et "Z" sont déjà utilisés pour désigner les axes de l'objet 3D dans l'espace objet. Les coordonnées UV sont stockées, pour chaque point 3D, dans les données du mesh.

Nous avons effectué le dépliage des UVs et l'affectation des textures sur Blender. Ces données sont ensuite importées par Unity en même temps que les données du mesh.



FIGURE 2.6 – Etapes de l'UV Mapping

Scène Unity

3.1 Environnement et ambiance

3.1.1 Ambiance lumineuse

Éclairage général Tout comme dans le jeu original Slender : The Eight Pages, nous avons choisi de paramétrier l'éclairage global du jeu de la sorte :

- pas de sources de lumière autres que la lampe torche du personnage
- une lumière ambiante existante mais très faible

Afin de mettre en oeuvre cette ambiance lumineuse, il nous a fallu ajuster plusieurs paramètres dans Unity :

- réduction de la lumière ambiante dans les "render settings" du projet
- le seul objet de type "light" présent sur la scène est la lampe torche
- assombrissement du brouillard ainsi que de la couleur d'arrière-plan des caméras de la scène. Cette couleur d'arrière-plan est visible au niveau du far plane pour chaque caméra, et est bleu par défaut.

La lampe torche

Choix du type de la source lumineuse Pour représenter le faisceau lumineux d'une lampe torche, nous avions le choix entre utiliser une lumière directionnelle avec un masque en forme de disque, ou bien une lumière de type "spotlight", en forme de cône. L'avantage du spotlight était de permettre une atténuation de l'intensité lumineuse en fonction de la distance, alors que la lumière directionnelle permettait de produire des ombres dynamiques et donnait une meilleure impression de faisceau lumineux. Finalement, nous avons opté pour le spotlight, comme dans le jeu Slender : The Eight Pages.



(a) Spotlight



(b) Lumière directionnelle

FIGURE 3.1 – Comparatif entre une lumière directionnelle et spotlight

Utilisation d'un cookie Pour rendre la lampe torche plus réaliste, nous lui avons appliqué un cookie. Un cookie est une image en niveaux de gris qui représente un canal alpha. C'est un masque que l'on applique à une source lumineuse pour en moduler l'intensité, d'une façon similaire à l'application de texture à un objet 3D.



(a) Avec cookie



(b) Sans cookie

FIGURE 3.2 – Comparatif entre une lumière avec et sans cookie

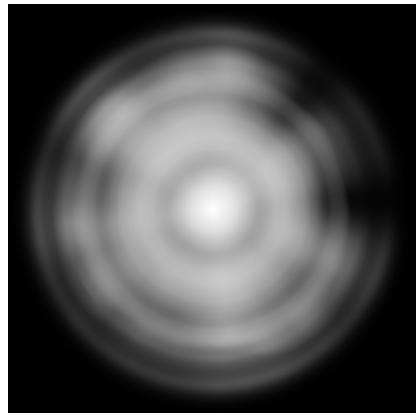


FIGURE 3.3 – Cookie de la lampe torche

3.1.2 Particules et effets visuels

Introduction Afin d'ajouter du réalisme dans notre projet, nous avons ajouté un certain nombre d'effets de particules. Nous avons choisi d'utiliser un système de particules afin de représenter de la pluie ou encore du brouillard. Ces effets nous permettent aussi de cacher en partie le décors autour du bâtiment, que nous n'avons pas modélisé. En outre, nous avons aussi décidé de mettre des effets de particules pour créer des effet visuels permettant de repérer les éléments importants du jeu. Par la suite, nous détaillerons donc chacun de ces systèmes afin de pouvoir comprendre comment nous les avons créés.

La pluie

Afin de réaliser notre système de particules représentant la pluie, nous avons utilisé le gameObject "Particle System" proposé par Unity3D. De ce fait, nous avons pu configurer ce gameObject afin d'obtenir l'effet voulu.

Pour commencer à créer notre pluie, nous devons commencer par configurer l'émetteur de particules. Nous lui indiquons donc la façon dont vont être générées les particules de notre système. Dans notre cas, nous devons par exemple indiquer au gameObject que le système de particules devra tourner en boucle car on ne veut pas que la pluie s'arrête. Nous devons le configurer de manière à avoir une pluie assez dense

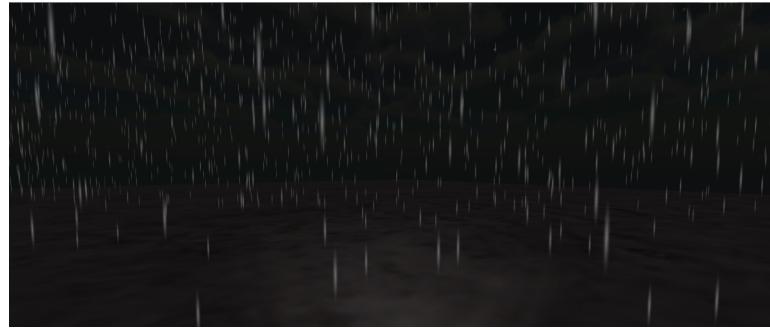


FIGURE 3.4 – Capture d'écran du système de particules "pluie"

car le but est de masquer en partie l'extérieur. Nous lui indiquons aussi la forme et la taille du système de particules et le nombre de particules qui doivent être émises.

La deuxième phase de configuration du système de particules est la phase de rendu, c'est-à-dire l'aspect que prendra le système. Pour cela, nous devons utiliser les stretched billboard pour l'affichage. Ces billboards permettent d'appliquer une texture étirée verticalement sur nos particules, ce qui donne un effet de gouttes d'eau tombantes.

Afin de rajouter un effet de réalisme à notre pluie, nous avons décidé d'y ajouter un effet de collision. Ainsi, lorsque la pluie touche le sol, elle détecte la collision et déclenche un autre effet de particules qui représente un léger éclaboussement.

Sur la figure ci-dessous, nous pouvons voir comment nous avons configuré le système de particules.

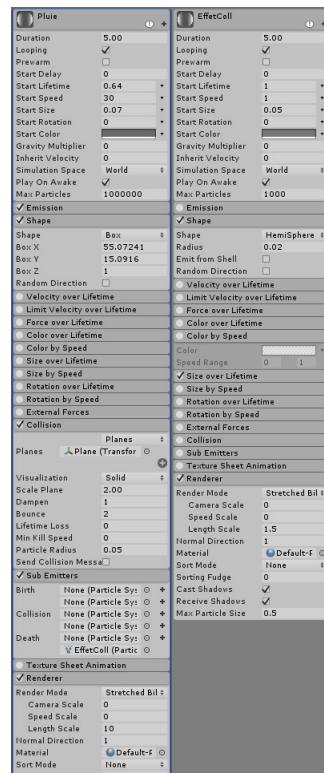


FIGURE 3.5 – Configuration du système de particules "pluie"

L'effet de brouillard



FIGURE 3.6 – Capture d'écran du système de particules "brouillard"

Afin de nous simplifier la tâche, nous avons décidé d'utiliser un "prefab" provenant de l'asset store de Unity. Nous avons donc téléchargé l'ensemble "Sky FX Pack" et nous avons adapté le système de particules servant à faire une tempête de sable afin d'obtenir un brouillard qui nous permettrait d'avoir l'effet voulu.

Objets



FIGURE 3.7 – Capture d'écran du système de particules pour les objets

Nous avons utilisé un troisième type de système de particules afin de mettre en évidence les objets qu'il faut ramasser. En ce qui concerne ces particules, nous avons configuré l'émetteur de façon à ce que les particules soient émises en boucle. On leur applique une gravité négative afin qu'elles semblent s'en voler, et une transformation en fonction de leur cycle de vie pour créer un effet d'apparition et de disparition plus naturel. Puis, lorsque nous ramassons les objets, nous détruisons simplement le système en même temps que l'objet.

3.1.3 Ambiance sonore

Sur Unity, il est possible d'ajouter des sons 2D et des sons 3D. Chaque son doit être émis d'un composant AudioSource et ne peut être "entendu" que par un composant AudioListener. La différence entre un son 2D et un son 3D est qu'un son 3D aura son volume et sa balance stéréo affectés par la position de l' AudioSource par rapport à l' AudioListener.

Nous avons utilisé les deux types de sons dans le jeu.

Exemple de sons 2D :

- Les musiques du menu et musiques d'ambiance du jeu
- Les bruits de pas du joueur, car la source de ce bruit est toujours la même relativement à la caméra, qui est à la première personne
- Les effets sonores joués lorsque l'écran se brouille

Exemple de sons 3D :

- Le bruit que produit un objet lorsqu'on le ramasse
- Le bruit que produit un objet qui tombe ou rentre en collision avec un autre objet. Attention, entendre ce genre de bruit pourrait trahir la présence du Slenderman à proximité du joueur !

3.2 Animations

3.2.1 Les différents types d'animations

Les animations permettent de mettre les objets que nous avons modélisé et se réalise par l'interpolation de keyframes. Dans ce projet, nous avons réalisé nos animations de deux manières différentes. La première manière a été en utilisant l'outil pour créer des animations sous Blender et la deuxième manière consiste à coder directement les animations au sein de l'EDI (MonoDevelop) proposé par Unity.

Blender

Blender met à la disposition des utilisateurs un outil permettant de réaliser des animations sur les mesh que nous sommes en train d'éditer grâce au outils "Timeline" et "Dope Sheet".

Timeline L'outil "Timeline" permet de créer les frames de nos animation ainsi que de spécifier le nombre de frames au sein de notre animation. Il permet également de réaliser des keyframes animation comme nous l'avons vu en cours. Pour cela, il suffit de placer les keyframes et l'outil va automatiquement réaliser l'interpolation entre les différentes keyframes grâce à courbe d'interpolation nommée "IPO".



FIGURE 3.8 – Capture d'écran de l'outil "Timeline"

Dope Sheet La "Dope Sheet" est un outil dans blender qui permet de voir de l'évolution des mouvements des objets au sein de la scène au cours de l'animation. C'est aussi avec cet outil que l'on décide de faire plusieurs animation pour le même mesh.

Unity

Il est possible de réaliser également des animations de deux manières différentes au sein de Unity. La première consiste à créer directement les animations au sein de Unity grâce à l'outil "Animation" à la

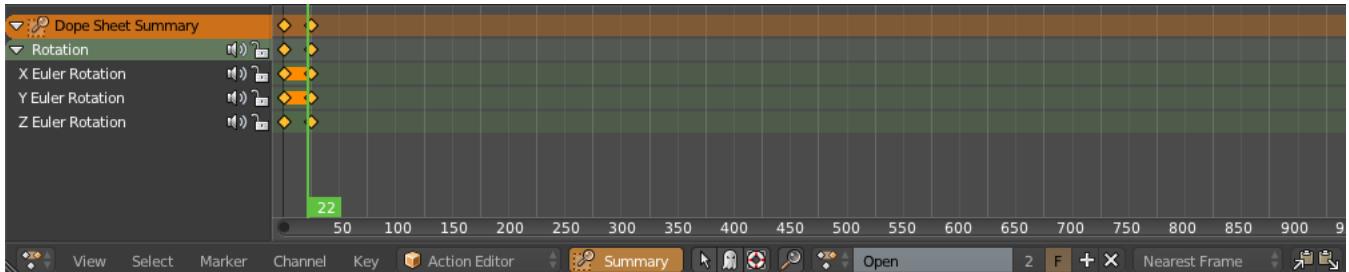


FIGURE 3.9 – Capture d'écran de l'outil "Dope Sheet"

condition que le modèle que l'on veut animer soit présent dans le graphe de scène.

Cet outil est assez similaire à l'outil "Timeline" de Blender dans le sens où il est capable de réaliser des keyframes animations et permet également de réaliser plusieurs animations sur le même objet.

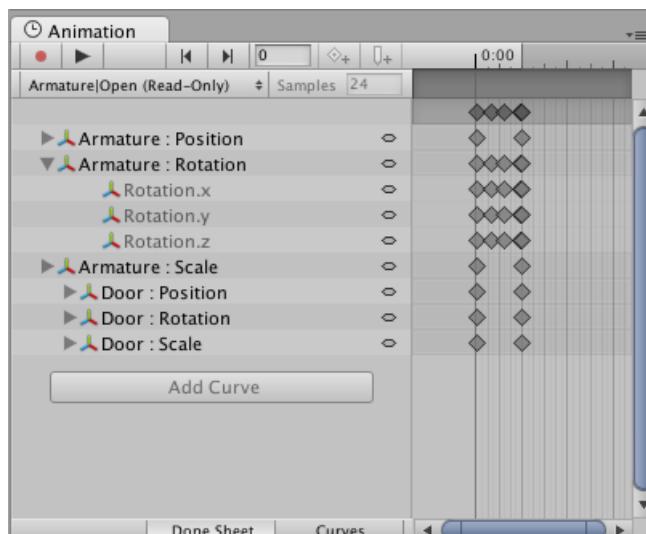


FIGURE 3.10 – Capture d'écran de l'outil "Animation"

Unity propose également un autre système afin de réaliser des animations. Ce système se nomme "Animator" et permet de réaliser des transitions entre les différentes animations importées sur l'objet en représentant l'objet sous la forme d'une machine à état (Chaîne de Markov) où les noeuds représentent une animation ou un état de l'objet et les arcs représentent une transition entre animation ou états.

Grâce à cet machine à état, il est possible de définir les paramètres de transitions en les différentes animations. Les transitions entre les différentes animations sont réalisées grâce à une interpolation par le moteur d'animation de Unity.

Dans la version 4.6 de Unity, il est préférable d'utiliser le système "Animator" car habituellement les animations des objets ou des humanoïdes présents à l'intérieur du graphe de scène sont réalisées par d'autres personnes et/ou dans des outils autres que Unity. Par conséquent, il est plus simple d'importer directement celles-ci que de les recréer.

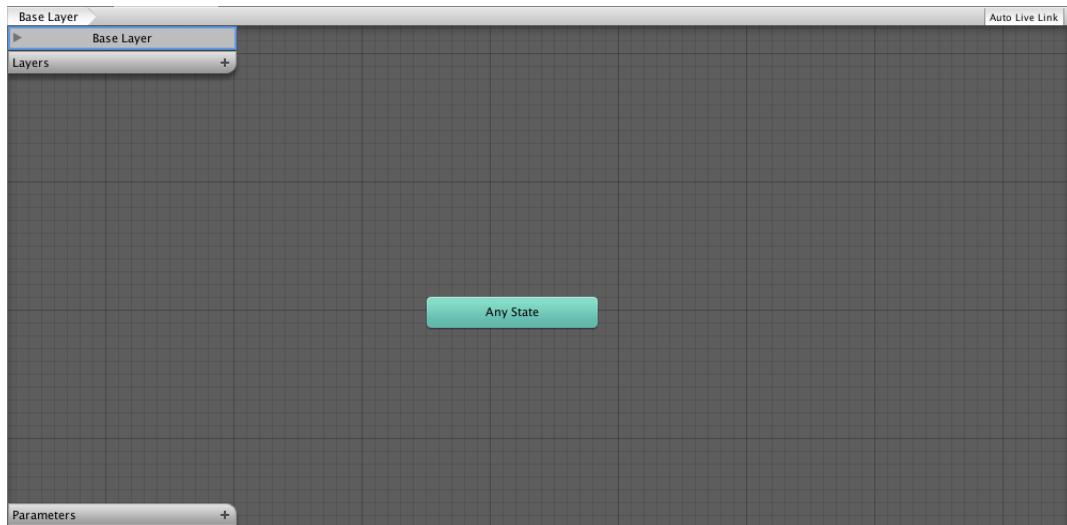


FIGURE 3.11 – Capture d'écran d'un exemple d'Animator

3.2.2 Cinématique d'introduction



FIGURE 3.12 – Aperçu de la cinématique d'introduction

L'une des animations principales présente dans le jeu est la cinématique d'introduction. Cette animation se déclenche à chaque nouvelle partie, mais il est possible de la sauter en appuyant sur une touche.

Choix du type d'animation Cette animation repose sur l'exécution d'une série d'actions sur un objet 3D particulier qui possède une armature. Comme cette animation comprend également plusieurs plans et mouvements de cameras, il faut avoir un contrôle sur l'enchaînement de chaque animation et mouvement de camera pour que le tout soit synchronisé indépendamment du framerate. Pour réaliser cette animation nous avons donc choisi de découper les actions du personnage en plusieurs animations et de contrôler l'enchaînement de chaque plan par un script.



FIGURE 3.13 – Création de la cinématique d'introduction

Le script se déclenche lors du début de la partie. Celui-ci :

- met en pause les acteurs du jeu (le personnage principal et le Slenderman) et contrôle plusieurs cameras
- enchaîne une série de plans possédant chacun un mouvement de camera une condition nécessaire pour passer au plan suivant et une action à effectuer lors du passage au plan suivant. Par exemple, on attend que le personnage ait effectué l'animation d'ouverture de la porte tout en effectuant une rotation de la camera dans sa direction. On effectue ensuite un zoom sur son visage, et on enchaîne avec le plan suivant.
- lorsque tous les plans ont été joués, on réactive les éléments du gameplay, puis on supprime l'objet représentant l'animation d'introduction. Le jeu peut commencer !

3.2.3 Effet de grésillement de l'écran

Nous avons utilisé un script afin de produire sur l'écran un grésillement lorsque le Game Over est imminent.

Le grésillement consiste en une texture attachée à la caméra dont on modifie l'opacité pour simuler un brouillage de l'écran.

A chaque frame, l'opacité de la texture est déterminée aléatoirement. Elle apparaît et disparaît donc très rapidement plusieurs fois par seconde, donnant un effet de grésillement. L'opacité moyenne de la texture est déterminée par le nombre de points de vie du joueur, qui sont faibles lorsque le Slenderman est proche, et qui sont restaurés lorsqu'on s'en éloigne.

Le grésillement de l'écran est accompagné d'un son mêlant bruit blanc et basses fréquences qui rappelle les interférences d'un écran TV. Le volume de ce son est déterminé à chaque frame de la même façon que l'opacité de la texture de grésillement.

3.2.4 Robots

De petits robots-fourmis Arduino déambulent de façon aléatoire dans la salle TP Systèmes. De la même façon que pour la cinématique d'introduction, l'animation de ces robots a été faite grâce à un script.

Comportement Le comportement des robots est assimilable à une machine à états. Lorsqu'un robot avance, il possède à chaque instant une petite chance de se mettre à tourner sur lui-même. Lorsqu'il tourne sur lui-même, il possède à chaque instant une petite chance d'avancer de nouveau en ligne droite. Cela produit un mouvement simple et aléatoire.

Choix du type d'animation Comme les robots se comportent comme une machine à états, il aurait pu être judicieux d'utiliser l'Animator pour mettre en place cette animation. Cependant, l'Animator permet uniquement de jouer des animations, et non de déplacer des gameObjects. Or, nous voulions que le déplacement des robots ne soit pas uniquement un effet graphique mais que leur déplacement soit aussi physique, et qu'ils gèrent les collisions. C'est la raison pour laquelle nous avons géré leur déplacement et leur animation par un script.

3.3 Les interactions avec l'environnement

3.3.1 Objets gérés par le moteur physique

Après les modélisation des étages et objets, nous avons placé les objets dans chaque pièce du bâtiment. Nous avons voulu tirer profit du système de gestion de la physique intégré à Unity pour ajouter du réalisme au jeu : ainsi, le joueur peut déplacer ou renverser les objets qui ne sont pas trop lourds (chaises, tables). Pour permettre cela, il faut ajouter deux composants Unity aux GameObjects représentant chaque objet : un rigidBody et un Collider. Le Collider est un volume englobant dont on choisit la forme et qui sert à effectuer la détection des collisions avec cet objet. Les volumes englobants sont des "boîtes" que l'on utilise afin de délimiter un objet. Nous avons choisi des Box Colliders pour des raisons de performance.

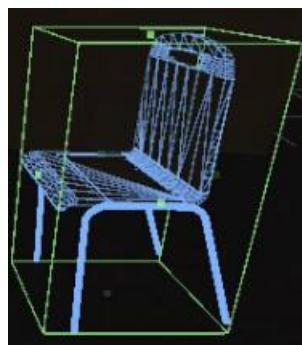


FIGURE 3.14 – Capture d'écran du composant physique "Box Collider"

Le rigidBody permet, lorsqu'un collider est présent, de soumettre les objets à la gravité et de réagir aux collisions avec les autres objets selon la force avec laquelle ils entrent en collision. Pour que le joueur puisse interagir avec ces objets, il a fallu ajouter un rigidBody ainsi qu'un Collider à l'objet représentant le joueur.

3.3.2 Portes

Dans notre jeu PolySlender, nous avons mis en place des animations sur les portes pour que lorsque le personnage s'approche ou s'éloigne d'une porte, celle-ci s'ouvre ou se ferme. Afin de mettre en place ces animations au sein du jeu, nous les avons d'abord réalisé au sein de Blender comme expliqué plus haut. Nous avons deux types de portes :

- Une porte que l'on peut ouvrir sans avoir besoin d'un objet particulier.
- Une porte qui nécessite d'avoir la carte étudiante en tant qu'objet possédé afin d'ouvrir la porte.

L'animation d'ouverture consiste en la rotation de l'objet sur l'axe vertical de +80° et celle de fermeture consiste à faire tourner la porte du même angle mais dans le sens inverse.

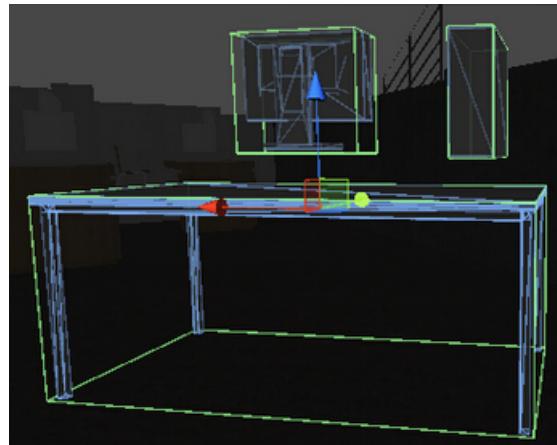


FIGURE 3.15 – Capture d'écran du composant physique "Rigidbody"

Une fois que les animations des portes ont été réalisées, nous les avons implémentées dans Unity et affectés pour chaque objet porte au sein de la scène principale un composant "Animator".

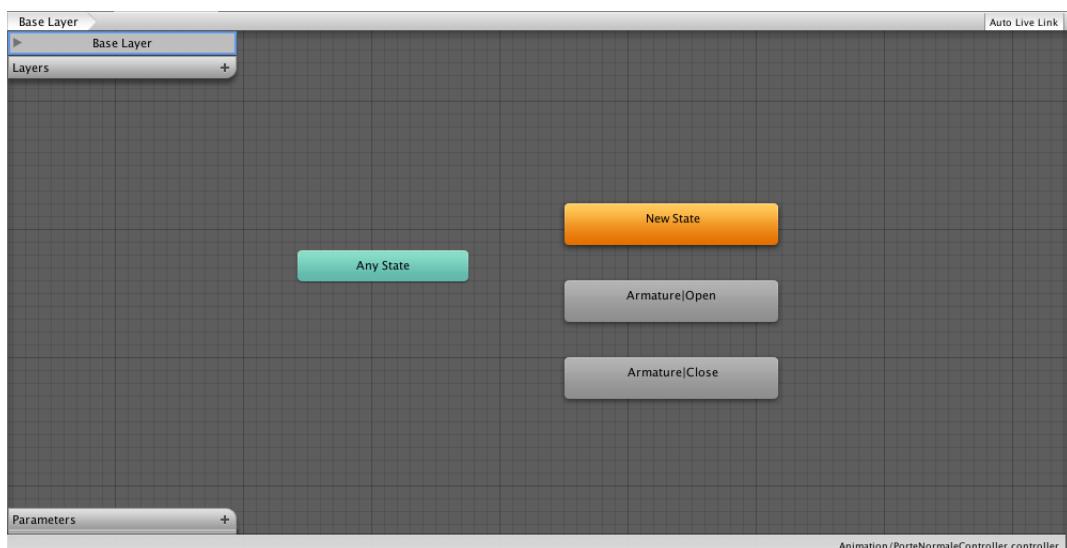


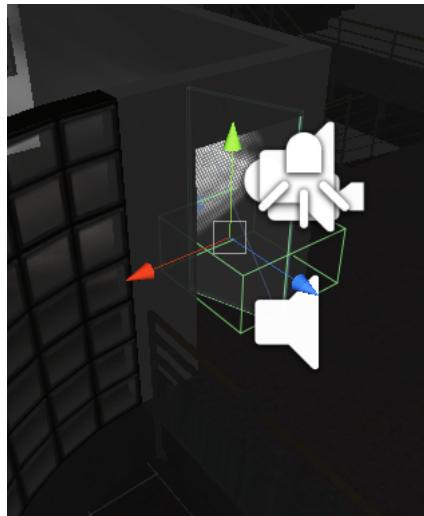
FIGURE 3.16 – Capture d'écran d'un exemple d'Animator

On peut voir que la machine à état est composée de quatre noeuds :

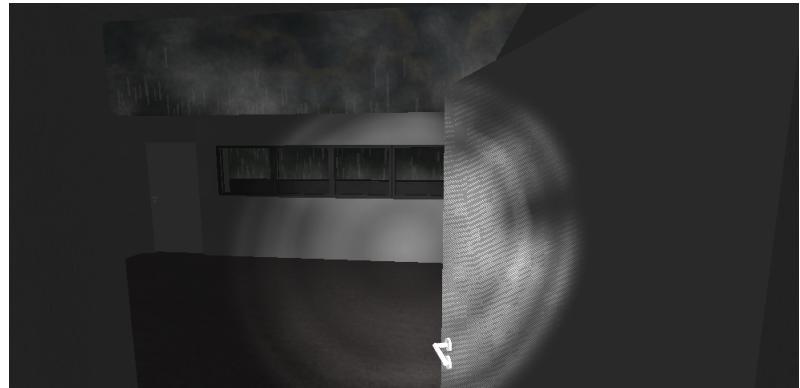
- Any state : Noeud permettant de représenter n'importe quel état de la porte.
- New State : Noeud permettant de représenter l'état au repos.
- Armature|Open : Noeud représentant l'animation d'ouverture.
- Armature|Fermeture : Noeud représentant l'animation de fermeture.

Cette machine à état ne comporte pas d'arcs car nous n'avons pas besoin de transitions entre les animations de fermeture et d'ouverture.

Pour déclencher les animations des portes qui ne nécessitent pas d'objet, nous utilisons le système de collision de Unity. On définit un volume englobant sur la porte de telle manière que lorsque les volumes englobant de la porte et du personnage du jeu se rencontrent, l'animation de d'ouverture ou de fermeture se déclenche.



(a) Image d'origine



(b) Après une détection des contours de Laplace

FIGURE 3.17 – Visualisation de l'animation d'ouverture de porte avec collision entre BV

Pour la porte qui requiert la carte étudiante, nous utilisons le même système que précédemment avec en plus lors de la collision des volumes englobant entre la porte et le personnage, la vérification que le personnage possède la carte via un script réalisé en C#.

3.3.3 Collisions

Afin de gérer les collisions dans notre jeu, nous utilisons deux systèmes différents : l'un pour gérer les collisions avec le joueur et l'autre pour gérer les collisions avec le Slender. Cette seconde méthode utilise le Navmesh qui sera présenté dans la partie 4.2.2.

Pour ce qui est de la partie concernant les collisions avec le personnage principal, nous utilisons le système des volumes englobants. Unity possède un certains nombre d'outils très pratiques qui permettent de gérer et de définir ces volumes englobants. Par exemple, nous avons utiliser ces outils afin d'ajouter un composant "Mesh Collider" sur les différentes parties qui ne peuvent pas être traversées par notre joueur.

Parallèlement à cela, nous avons dû définir un volume englobant pour le joueur. Nous avons ainsi utilisé un "Capsule Collider" afin de délimiter le joueur. En procédant ainsi, les collisions entre le joueur et tout élément possédant un volume englobant défini comme pour ceux des murs seront gérées.

Unity propose aussi un outil permettant de gérer la gravité. Ainsi, grâce à cela, le joueur ne passera pas en dessous du sol.

3.4 LOD

Durant ce projet, on nous a demandé d'implémenter du niveau de détail (Level Of Detail, LOD). Celui ci permet de simplifier l'affichage de certain élément afin de simplifier le dessin de la scène. Il existe plusieurs type de LOD. Celui que nous avons implémenté et le Discret LOD (DLOD).

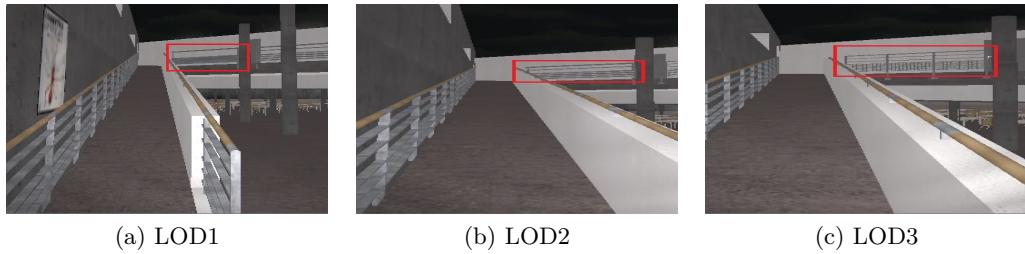


FIGURE 3.18 – LOD

3.4.1 Simplification des meshes

3.4.2 Le script du DLOD

Son principe est le plus intuitif. Plus on se rapproche plus on veux voir du détail. Pour cela il faut posséder plusieurs implémentations du même objet mais avec plus ou moins détails. Puis nous échangeons l'implémentation visible avec une autre selon le niveau de détail voulu.

Voici le pseudo code du LOD :

Input:

- tabDistance : le tableau des distance seuil par ordre croissant
- pltabObjet : le tableau des objets
- camDistance : la distance de l'objet à la camera
- currentLevel : le niveau de détail actuel

Output: l'objet du bon LOD s'affiche

Begin

```

i, level := Entier
level = -1 ;
for i = 0 ; i < tabDistance.size ;i++ do
    if (camDistance < tabDistance[i]) then
        level = i
        break ;
    end if
end for
if (level < 0) then
    level = tabDistance.size ;
end if
if (level ≠ currentLevel) then
    Afficher(tabObjet[level] ;
    Cacher(tabObjet[currentLevel] ;
    currentLevel = level ;
end if

```

End

Le gameplay

4.1 Contrôle du personnage

4.1.1 Caméra

Notre jeu étant un jeu d'immersion et d'horreur, nous avons opté pour une vision à la première personne. De ce fait, la caméra principale du jeu fait partie intégrante de notre joueur. Afin d'obtenir le rendu voulu, nous avons utilisé le prefab "First Person Controller" qui est fait partie des assets basiques de Unity3D. Suite à cela, nous avons dû configurer ce préfab de façon à avoir l'ensemble des fonctionnalités dont nous avions besoin.

Tout d'abord, nous n'avons que très peu modifié les scripts servant à la caméra. Tout ce que nous avons fait a été de cacher le curseur de la souris de manière à ce que la caméra serve elle-même de curseur. Néanmoins, une fois que le personnage et sa caméra correctement configurés, il était nécessaire de gérer les collisions de la caméra avec les éléments du décor. Pour faire cela, nous utilisons un système de détection de collisions défini par Unity (cf. 3.3.3). Tout ce qu'il nous suffit de faire c'est d'ajouter un composant "rigidbody" à notre personnage.

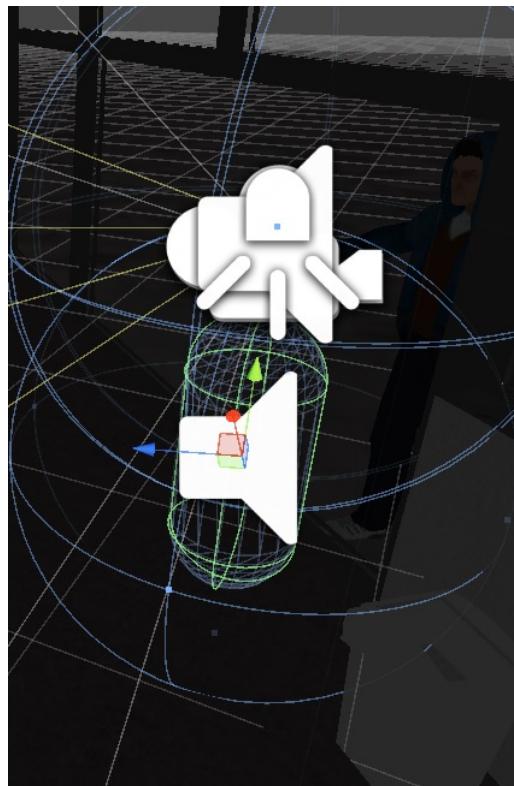


FIGURE 4.1 – Personnage jouable du jeu

Paramètres de la camera Pour que la vue à la première personne soit la plus réaliste possible, nous nous sommes intéressés à l'influence de la hauteur de la camera par rapport au sol, et à son angle (field of view). En théorie, l'œil humain possède un angle de vision proche des 120°, mais en pratique, il n'est pas réaliste d'utiliser un tel angle de vue dans un jeu à la première personne. Nous nous sommes limités à un angle de 60° qui permet de ne pas avoir un effet de zoom trop prononcé mais qui reste acceptablement étroit pour un jeu d'horreur (la restriction du champ de vision contribue à faire monter la peur). Pour la hauteur de la camera, nous l'avons positionnée 1m20 du sol. Cela peut sembler étrange car on représente alors un personnage mesurant environ 1m20, mais nous avons constaté que si l'on augmentait la hauteur de la camera au delà de ce seuil, malgré que tout le bâtiment ait été modélisé à l'échelle, cela donnait l'impression que le personnage était très grand, plus grand qu'un encadrement de porte. Nous avons donc laissé cette hauteur à 1m20, car l'effet était plus vraisemblable.

4.1.2 Gestion de la vie

Durant notre jeu, il est possible de mourir. Pour se faire, il y a deux possibilités :

- Le Slender reste trop longtemps à portée du joueur.
- Le joueur regarde le slender pendant trop longtemps

Dans ces deux cas, on a une notion de temps qui intervient. Afin de représenter cette notion, nous avons utiliser un système de points de vie pour le joueur.

Afin de gérer la vie de notre joueur, nous commençons donc par récupérer des informations provenant de notre antagoniste : le Slender. L'information dont nous avons besoin dépend de l'état d'agressivité du Slender :

- Si on estime que le joueur a trop regardé le Slender ou que celui-ci est resté à porté trop longtemps, nous récupérons une information signifiant la défaite immédiate de notre personnage
- Si le Slender se contente de pourchasser sa cible, on récupère alors la distance qui se trouve entre le Slender et le joueur. Cette distance se calcule à partir du Navmesh (cf. 4.2.2) et nous permettra de définir des seuils qui font que le joueur perd de plus en plus de vie quand le Slender se rapproche.
- Enfin on récupère une autre information si le Slender ne poursuit pas le joueur. Dans ce cas, on a décider de faire en sorte que la vie du joueur remonte progressivement en vue d'une prochaine rencontre avec l'ennemi.

Concernant l'affichage de la vie du joueur, nous avons utilisé un système existant dans le jeu d'origine. Ce système n'est autre que le grésillement de l'écran (cf. 3.2.3). Le principe est simple. Plus l'écran est brouillé et moins on a de vie. Donc si on arrive à échapper à notre poursuivant, l'écran redeviens progressivement normal.

4.2 Fonctionnement du Slenderman

4.2.1 IA

Généralités de l'IA

Le slender est l'antagoniste du jeu. Il ne peut faire que quatre actions :

- chasser le joueur
- s'immobiliser
- se téléporter
- tuer le joueur

Voici le pseudo code du slender :

Input:

- slender : la position du slender
- player : la position du player

Output: distance : la distance entre les deux**Begin**

```

distance := float;
if (le joueur à regardé trop longtemps le slender) then
    Tuer le joueur
    distance = -1;
else if (le slender est vu par le joueur) then
    stopper le slender
    distance = distance(slender,joueur)
else if (le joueur est vu) then
    Chasser le joueur
    distance = distance(slender,joueur)
else if (le joueur est senti) then
    slender se tourne vers le joueur
    distance = ∞
else
    téléporter le slender
    distance = ∞
end if
return distance;

```

End

Afin de réaliser ces actions nous avons défini différentes zones permettant de donner des limites à ces actions.

- La première zone est celle de téléportation, elle donne une sphère autours du joueur dans laquelle le slender peut se téléporter.
- La seconde est une sphère sensoriel ; si le joueur rentre dans cette sphère, le slender le sens et ce retourne vers lui.
- La dernière est la zone de vue. Il y a deux zones de vue, une pour le joueur et une pour le slender.

Afin d'augmenter la difficulté, au fur et à mesure que le joueur ramasse des objet, la zone de téléportation et le temps entre chaque téléportation se réduisent, de plus la zone sensorielles du slender augmente. Le slender à donc de plus en plus de chance de trouver le joueur.

Gestion de la vue

Le principe du slender se base essentiellement sur la gestion de la vue. Comme vu précédemment selon que le joueur voit le slender ou non les actions effectuées sont différentes. DéTECTé un objet dans le champs de vision est donc un point important du jeu.

Pour pouvoir detecter un objet, trois données doivent être connues :

- le vecteur position de l'"oeil"
- l'angle de vision
- le vecteur position de l'objet

Le principe de détection est le suivant. A partir des deux position, on calcule le vecteur de direction de l'oeil a l'objet. Puis nous trouvons l'angle entre ce vecteur et la normale à l'oeil. Si l'oeil est inférieur à l'angle de vision, alors l'objet est dans le champs de vision. Puis il ne reste plus qu'à tester si aucun mur ou objet empêche de voir l'objet. Cette étape est réalisée grâce au lancé de rayon.

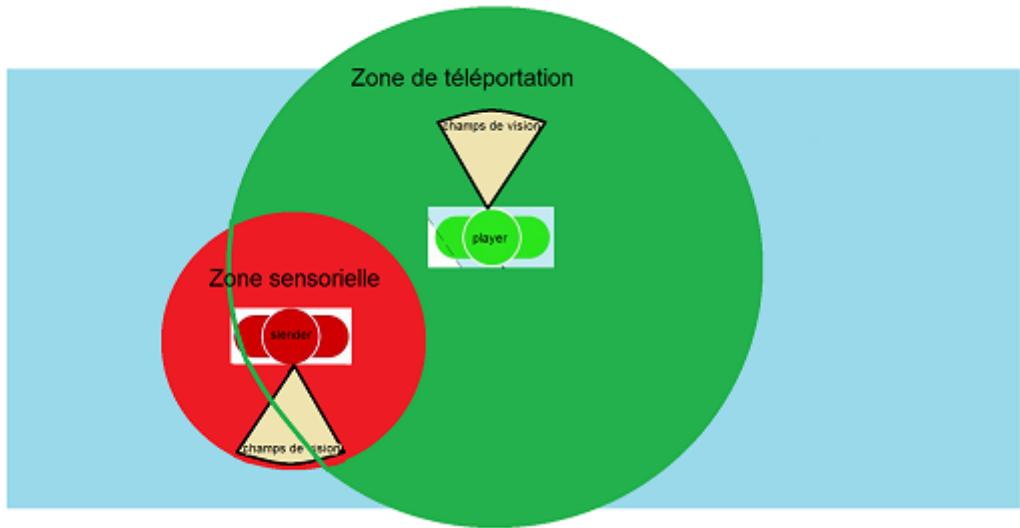


FIGURE 4.2 – Zone d'action du slender

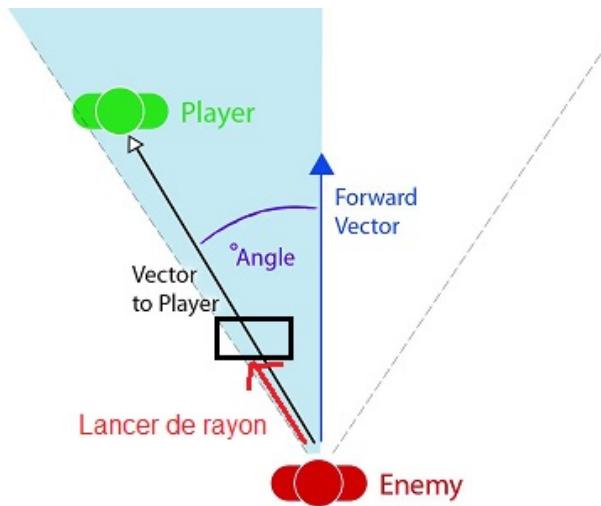


FIGURE 4.3 – Détection du champ de vision

Gestion du déplacement

Le déplacement du joueur est simple. Si le joueur est vu, il le poursuit en utilisant une fonction d'unity. Si le slender doit se téléporter, il sélectionne une position au hasard dans la sphère et s'y téléporte. Si le joueur doit être tué, le slender se téléporte juste à côté du joueur.

4.2.2 Du côté d'Unity3D

Pour pouvoir réaliser l'IA, nous avons utilisé des composants d'unity. Ces composants, simple d'utilisation nous ont permis d'implémenter les déplacements et les vue ainsi que simuler un sixième sens pour le slender. Ces outils sont :

- Le navmesh
- Le raycast et le dot
- Les collider

Le NavMesh

Le NavMesh, l'abréviation de Navigation Mesh, est une structure de données appartenant au système de navigation d'unity. Il décrit les surfaces navigable par un objet. Le navmesh permet de trouver le chemin d'un endroit à un autre. Le principe du NavMesh est simple à comprendre. Hors ligne, Unity calcule les surface navigable en la subdivisant en polygones convexes. Les contours des polygone ainsi que leurs frontière l'un à l'autre sont donc sauvegarder.

Puis lors du jeu, le chemin entre deux points est calculer grâce à l'algorithme de plus court chemin A*, chaque sommet correspondant à un noeud et chaque frontière à un arc. A partir de la une suite de polygone forant le chemin est créée. Cette suite est appeler un corridor.

Des lors, un objet peut se déplacer sur le navmesh. Cet objet pour pouvoir accéder au navmesh doit posséder le component NavMeshAgent. Ce component permet de ne marcher que sur le navmesh et ainsi éviter les obstacles. Voici le navmesh de notre projet.

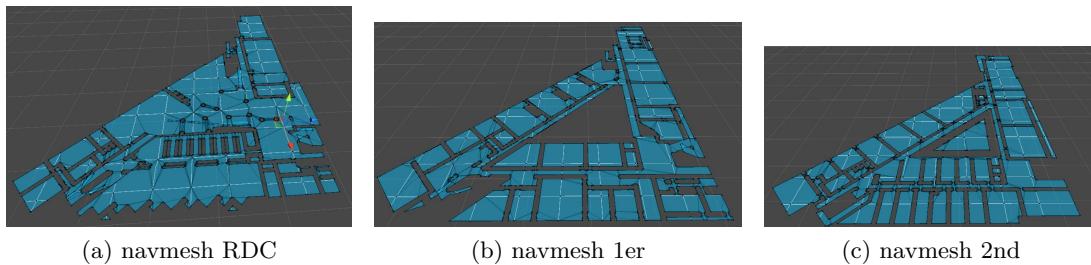


FIGURE 4.4 – NavMesh

Nous attachons donc à notre objet slender un component navmeshAgent.

Le navmeshAgent et le navmesh possèdent plusieurs fonctions de déplacement. Dans notre projet nous utilisons ces fonctions.

- NavMeshAgent.SetDestination(Vector3 target) : cette fonction calcule ou met à jour le chemin vers la position cible et bouge l'agent à la position suivante sur le chemin.
- NavMeshAgent.Warp(Vector3 target) : cette fonction téléporte l'agent à la position donnée.
- NavMeshAgent.CalculatePath(Vector3 target,NavMeshPath path) : cette fonction calcule le chemin vers la cible, le sauvegarde dans le path.
- NavMesh.SamplePosition(Vector3 sourcePosition,NavMeshHit hit, float maxDistance) : trouve la position sur le NavMash qui se rapproche de plus du spourcePosition

Dot et RayCast

Le DOT et le RayCast est sont les outils d'unity utilisés pour simuler le champs de vision.

Le Dot est une fonction qui permet de calculer le produit scalaire de deux vecteurs. Ainsi :

— Vector3.Dot(Vector3 v1, Vector3 v2) : $\|v1\| * \|v2\| * \cos(\theta)$ où θ est l'angle entre les vecteurs . Ainsi en utilisant dot sur deux vecteur normalisés (dont la norme est 1), le résultat donnera les cosinus. Nous pouvons donc faire la comparaison avec le cosinus de l'angle choisi comme demi champs de vision. Une autre manière aurai été d'utiliser la fonction :

- Vector3.Angle(Vector3 v1, Vector3 v2) : θ où θ est l'angle entre les vecteurs.

Cependant le calcule du dot est plus rapide que l'angle car c'est ne fais que de simples multiplications de matrice.

Nous pouvant à présent vérifier si un objet est dans le champs de vision. Il faut donc vérifié si l'objet n'est pas caché. Pour cela nous utilisons le lancer de rayon.

Le RayCast est une application de script permettant le lancer de rayon. Pour détecter si un objet n'est pas caché par un mur, nous avons attribuer un tag au mur. Lors du lancer de rayon, nous vérifions si le rayon touche un mur. Si un mur est touché, l'objet est caché. La fonction utilisé pour simuler les raycast est :

- Physics.Linecast(Vector3 start,Vector3 end,RaycastHit hitInfo) : cette fonction retourne un booléen si un objet à été toucher sur le chemin entre start et end

Les information sur l'objet trouver est mis dans hitInfo. Nous pouvons ainsi vérifier quel objet est touché.

Collider

Le dernier outil utilisé est le collider de type trigger. Ce collider permet de simuler la zone sensorielle. Lorsque le joueur y rentre il déclenche un évènement qui informe le slender de la présence du joueur.

4.3 Les objets à ramasser

Dans le jeu Slender : The Eight Pages, le joueur doit trouver et récolter huit objets. Ces objets sont des feuilles de papier placardées sur un mur ou un objet qui sont récoltées par le joueur lorsqu'il appuie sur la touche action à proximité de celles-ci. Dans PolySlender, nous avons repris et adapté ce principe.

4.3.1 Apparence des objets

Dans le jeu Slender : The Eight Pages, malgré que les pages à récupérer soient petites, elles sont faciles à repérer car :

- le joueur sait déjà quel type d'objet rechercher, tous les objets sont semblables
 - les pages sont positionnées verticalement
 - les pages sont blanches et le matériau utilisé reflète bien la lumière que la lampe de poche émet
- Cependant, dans notre jeu, les objets peuvent être bien plus difficiles à repérer pour différentes raisons :
- les objets sont de formes et de couleurs différentes
 - on ne peut placer certains objets au mur, il faut donc les poser à terre ou sur une surface, qui sont des emplacements moins évidents
 - certains objets sont petits, ce qui les rend difficilement repérables si on en est éloigné

Afin de faciliter la recherche des objets, on a donc ajusté les mécanismes du jeu de la façon suivante :

- mise en évidence des objets à ramasser par un système de particule
- faciliter le ramassage des objets en augmentant la distance à partir de laquelle on peut les ramasser
- ramassage automatique des objets lorsqu'on en est suffisamment proche

Nous allons voir plus en détail comment nous avons mis en oeuvre ces différents points.

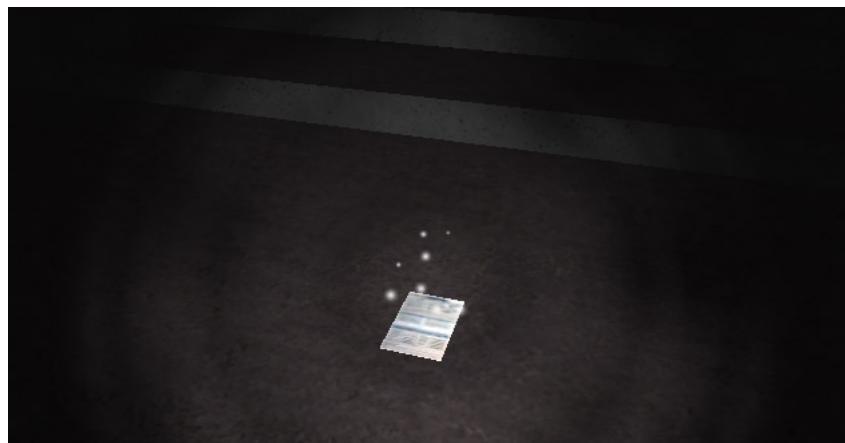
Effet de particules

Nous avons choisi d'utiliser un système de particules pour attirer l'attention sur les objets à ramasser car l'émission de particules permet de donner un indice sur la présence d'un objet indépendamment de sa taille et de sa forme : si un objet est caché on pourra deviner sa présence grâce aux particules qui se dispersent autour de l'objet. De plus, l'effet que nous avons appliqué n'est pas sans rappeler l'effet que produisent généralement les objets ramassables dans les jeux vidéos de type MMORPG, c'est donc un code répandu et connu des joueurs réguliers que nous avons repris.

Pour plus de précisions sur la mise en place de ce système de particules, se référer à la partie correspondante.



(a) Slender : The Eight Pages



(b) PolySlender

FIGURE 4.5 – Objets à ramasser dans le jeu original et dans PolySlender

4.3.2 Le placement des objets

Fonctionnement

Le fonctionnement du jeu original est le suivant : il existe au total 8 objets différents à ramasser. Il faut ramasser les 8 objets au cours de la partie. Il existe une liste d'endroits où peuvent potentiellement apparaître les objets. Au démarrage de la partie, 8 emplacements sont choisis au hasard et chaque objet est positionné selon l'un de ces emplacements. On notera qu'il existe plus de 8 emplacements possible, ce qui fait que l'on ne peut pas savoir à l'avance si un objet sera présent à un certain endroit ou non.

Dans PolySlender, nous avons utilisé exactement le même mécanisme. Nous avons choisi manuellement environ 20 emplacements possibles répartis sur les 3 étages du bâtiment modélisé.

Système mis en place

Le système qu'on a mis en place se compose de deux objets principaux :

- un script contenant les données et les opérations nécessaires pour placer aléatoirement nos 8 objets
- une liste de gameObjects représentant chacun l'emplacement possible d'un objet

Le script contient une liste de références vers les objets qu'il faut instancier dans le jeu. Pour cela nous avons créé un prefab par objet à ramasser et répertorié chaque préfab dans le script. La fonction principale du script est appelée lors du chargement de la scène de jeu. Celle-ci choisit un emplacement au hasard, y

instancie le premier prefab, puis supprime l'emplacement de sa liste. Cela évite qu'il soit possible d'instancier plusieurs objet au même emplacement. Cette opération est répétée pour chaque prefab répertorié.

Les emplacements possible sont représentés par un GameObject chacun. Le fait de les représenter ainsi nous donne deux avantages principaux : cela permet d'utiliser l'Editor d'Unity pour les positionner dans la vue 3D, ce qui est plus comode que de manipuler leurs coordonnées. Cela permet aussi de récupérer la liste de ces objets dans le script décrit ci-dessus en effectuant dans la scène une recherche par tag. Les GameObjects représentant un emplacement d'objet portent le tag "ObjectSpawn". Remarque : les GameObjects que l'on place sont vides, ils ne contiennent aucun Component et aucun noeud fils, car la seule information qui importe est leur transformation, et notamment leur position.

4.3.3 Récupération des objets et gestion de l'inventaire

4.3.4 Conditions pour ramasser un objet

Comme on l'a décrit plus haut, un objet est automatiquement ramassé par le joueur si celui-ci s'en approche suffisamment. Pour mettre en oeuvre ce mécanisme, nous avons ajouté à chaque objet un Component de type Collider. Ce Collider est de type Trigger, c'est à dire qu'il n'est pas utilisé pour calculer les collisions du système de gestion de la physique d'Unity, mais pour détecter si un autre objet entre en collision avec cet objet, et exécuter une fonction le cas échéant.

La fonction qui est appelée lorsqu'une collision est détectée entre un objet et le volume englobant de notre objet effectue les opérations suivantes :

- jouer un effet sonore indiquant la récupération de l'objet
- afficher un texte indiquant quel objet a été récupéré et le nombre d'objets restant
- ajouter l'objet à l'inventaire du joueur
- supprimer le GameObject qui représente l'objet

4.3.5 Gestion de l'inventaire

Gestion de projet

Dans le cadre de ce projet, nous étions une équipe de sept personnes. C'était seulement la deuxième fois (la première fois étant le PIL) que nous avions l'occasion de travailler en un aussi gros groupe. Pour nous aider dans l'organisation, les décisions d'utiliser un outil de versioning, un outil de gestion de projet, et la présence de multiples réunions s'est imposée. Nous allons détailler chacun de ces points ici.

5.1 Travail en collaboration

Pour travailler en collaboration, nous nous sommes répartis les tâches de la façon suivante :

- Boas David
 - Modélisation / Textures du 1er étage
 - Réunion des trois étages
 - Scripts Divers
 - Gestion du Git
- Fontaine Nicolas
 - Modélisation / Textures des objets communs
 - Effets de particules
 - Gestion de la vie du joueur
- Guoxiang Liu
 - Modélisation / Textures des objets communs
 - Placement des objets au sein de l'école
 - Création des objets à récupérer
- Mathonnet Charlie
 - Modélisation / Textures du Rez de chaussée
 - Prise et préparation des textures du bâtiment
 - Animations automatiques : Robots, introduction et grésillement
 - Gestion de l'inventaire et des objets à ramasser
 - Scripts Divers
- Meunier Bastien
 - Modélisation / Textures des objets communs
 - Comportement du SlenderMan
 - NavMesh
- Renaudeau Armand
 - Modélisation / Textures du 2ème étage
 - Animations déclenchées des portes
 - Scripts Divers
- Wenlong Li
 - Modélisation / Textures des objets communs
 - Placement des objets au sein de l'école
 - Création des objets à récupérer

Scripts divers comprend la gestion de la course, des collisions, de la gravité, et le placement des objets ramassables. L'objectif était que chacun puisse apprendre à maîtriser Blender et Unity.

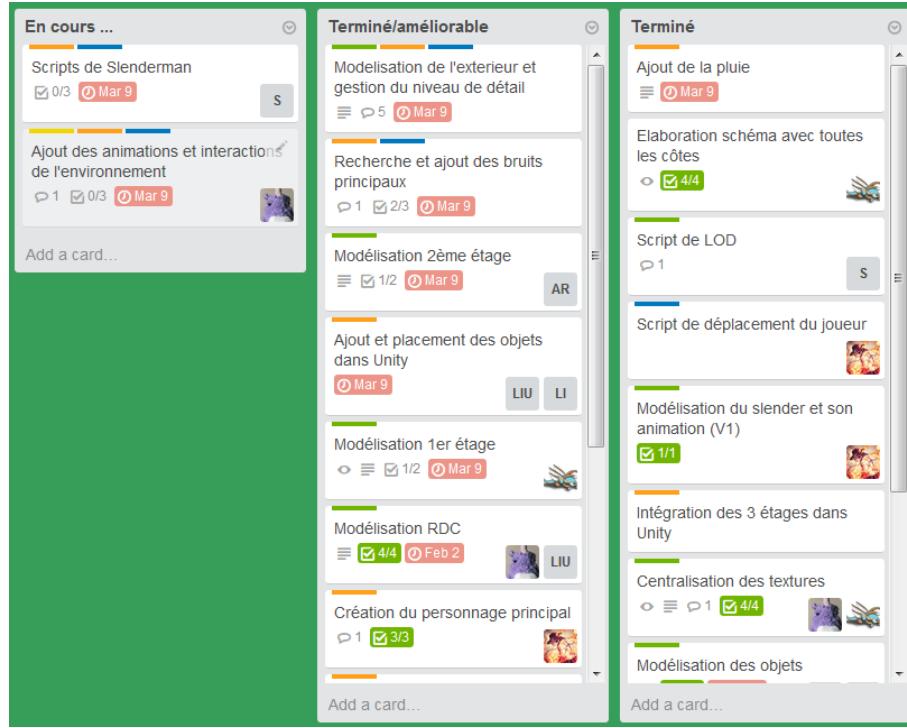


FIGURE 5.1 – Etat de notre Trello vers la fin du projet

Pour faire état de l'avancement de chacun et décider ensemble des décisions à prendre, des réunions étaient fréquemment organisées (presque toutes les semaines). Chacun pouvait alors parler des problèmes qu'il avait rencontrés, et ainsi redéfinir ensemble les tâches importantes, en affectant plus de personnes sur une tâche particulière par exemple. C'est ainsi que voyant que la modélisation et le remplissage de toutes les salles allait nous prendre beaucoup de temps, nous avons décidé de ne pas faire la salle "Chaine de Production" ainsi que l'espace du gardien, initialement prévu pour placer des "EasterEgg" (nous comptons cependant toujours les mettre pour certains ...).

5.2 Outils de gestion de projet

5.2.1 Trello

La répartition des tâches était généralement décidée par le chef de projet. Chaque tâche avait préalablement été définie en commun sur le service de gestion de projet Trello. Les tâches étaient réparties en cinq listes : Principales / Améliorations / En cours / Terminé-améliorable / Terminé. Les différentes personnes étaient alors répartis sur ces tâches, et un code couleur était utilisé pour identifier rapidement l'objet de la tâche : Vert pour la modélisation, Jaune pour les animations, Orange pour Unity et bleu pour la programmation de scripts. Des dates de fins étaient alors estimées pour chacunes d'elles.

5.2.2 Google Drive

Pour la centralisation des fichiers, Google Drive a été utilisé dans un premier temps. Cela nous a permis d'échanger facilement des documents et des modèles, en particulier des photos du bâtiment pour aider à la modélisation (le plan n'étant pas toujours exact à la réalité) et des photos des murs qui deviendront nos textures.



FIGURE 5.2 – Etat du GitHub vers la fin du projet

5.2.3 Guides

Au début du projet, presque tous les membres en étaient au même niveau, et n'avait pas fait de modélisation / Utilisation de Moteur en dehors des cours. La personne la plus expérimentée (Charlie) a ainsi pour booster l'effort collectif rédigés des guides disponibles en annexes pour éviter de buter sur les mêmes points et avancer le plus efficacement possible. Cela a été grandement utile pour apprendre rapidement les bases des compétences que réclamait ce projet ambitieux. Ensuite, lorsque la décision d'utiliser un outil de versioning s'est imposé, le chef de projet a rédigé un guide d'utilisation du dépôt Git pour que tout le monde puisse l'utiliser en rencontrant le moins de problèmes possible.

5.3 Outil de versioning

L'outil de versioning utilisé fut ainsi Git. La raison derrière ce choix est tout simplement qu'il s'agissait globalement de l'outil que tout le monde avait au moins utilisé une fois. Le projet fut alors hébergé sur Github, car nous avions la volonté de partager notre travail, et que si des groupes des années suivantes désiraient avoir un modèle 3d du bâtiment, qu'il puisse profiter de notre travail.

Chacun participa alors au dépôt une fois les premiers modèles ajoutés. Nous pouvions alors travailler sur Unity.

Une question dont nous n'avions pas encore la connaissance s'est posée : comment travailler tous ensemble sur la même scène ? La réponse fut trouvée avec l'utilisation des Prefabs. Ceux-ci permettent de stocker un GameObject complet avec ses composants et ses propriétés. Ils agissent comme des templates à partir desquelles nous pouvons créer de nouvelles instances d'objets dans une scène. La consigne fut alors de n'être qu'une seule personne à la fois à travailler sur un même Prefab.

Lors du développement d'une nouvelle fonctionnalité, une nouvelle branche était créée. Une fois la fonctionnalité opérationnelle, une fusion (merge) avec la branche principale était alors effectuée, et ce, de façon plus ou moins facile. En effet malgré l'apport que nous a apporté l'utilisation de Git, certain problème de prise en charge du logiciel au sein d'un projet Unity nous aura fait perdre plusieurs heures de travaux.

Conclusion

En conclusion, ce projet nous a permis à tous de mettre en pratique les différents éléments que nous avons vu en cours. Nous avons ainsi pu réaliser les différents objectifs que nous nous étions fixés au début de notre projet, et ce malgré quelques erreurs de notre part. En effet, nous estimons que nous nous sommes un peu trop attardé sur la partie modélisation et que nous y avons perdu un peu de temps. Néanmoins, l'immersion dans notre jeu d'horreur reste complète que ce soit au niveau de l'environnement de Polytech ou encore au niveau de l'ambiance du jeu.

Notre jeu est donc actuellement complet sur les parties les plus importantes. Il reste néanmoins certaines choses à régler. Par exemple, nous n'avons pas finalisé le cas où la victoire est acquise. C'est d'ailleurs un cas particulièrement difficile à obtenir.

Pour conclure ce rapport, nous pouvons tous dire que nous avons beaucoup apprécié ce projet. Nous avons pu mettre en place un gros travail autant en terme de modélisation qu'en terme de développement du jeu en lui-même, et ce dans un intervalle de temps relativement court.

Annexe A : Guide de modélisation

Guide : modéliser un étage du DI sous Blender

Sommaire

[Modélisation sous Blender](#)

[Préparation](#)

[Utiliser le système métrique](#)

[Afficher des infos utiles](#)

[Travailler avec le plan en fond](#)

[Modélisation](#)

[Règles générales](#)

[Comment faire ?](#)

[Tutoriaux](#)

[Raccourcis utiles](#)

[Importation dans Unity](#)

Modélisation sous Blender

Préparation

Utiliser le système métrique

Dans le panneau Scene, onglet Scene, dans la partie Units, sélectionner le système métrique. Par défaut le panneau Scène est dans la partie droite de l'interface. On laisse Scale à 1 car, comme on le verra plus tard, Unity et Blender utilisent la même unité de mesure.



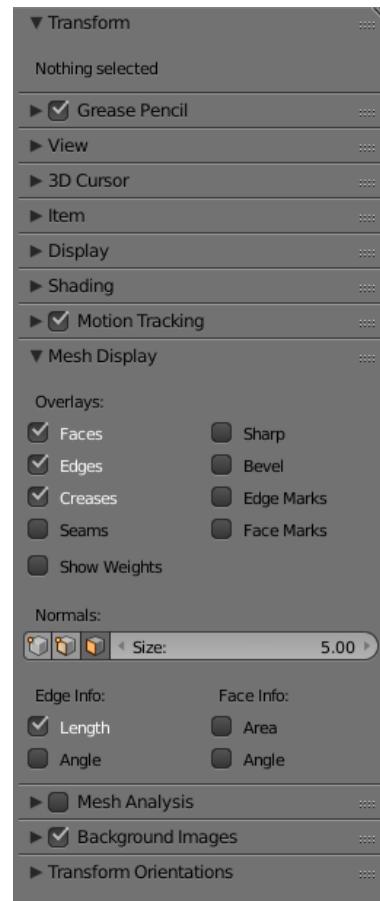
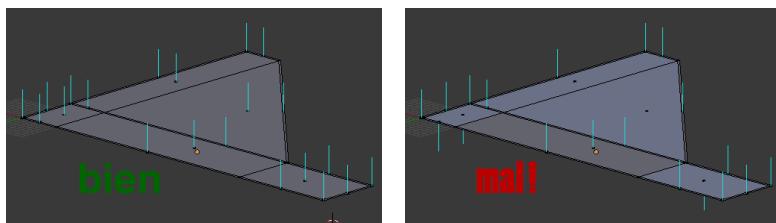
Guide : modéliser un étage du DI sous Blender

Afficher des infos utiles

Je vous conseille aussi d'afficher deux choses importantes : Les cotes, et les normales. Vous trouverez les cases à cocher dans le panneau des propriétés (raccourci : N).

Le premier (**Edge info : Length**) permet d'afficher la longueur des arêtes lorsqu'elles sont sélectionnées.

Le second (**Normals** : ) permet d'afficher les normales des faces. C'est très important ! Si vos faces ne sont pas dans le bon sens, elles ne seront pas rendues à cause du back face culling.

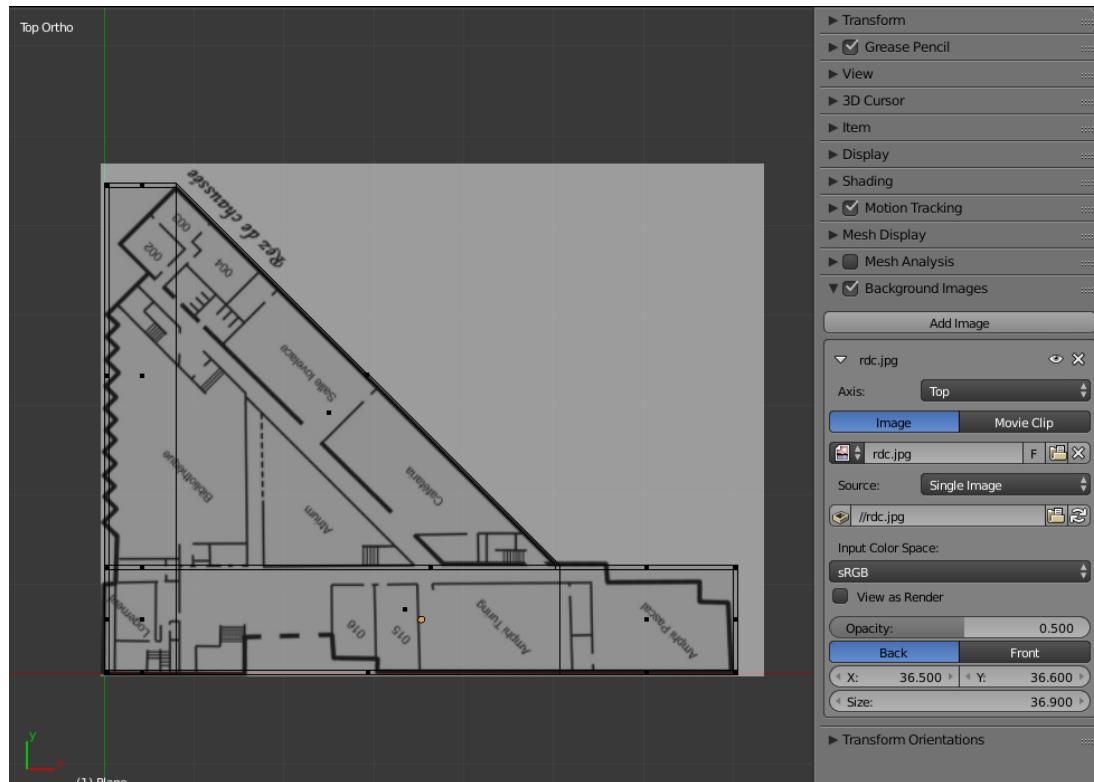


Guide : modéliser un étage du DI sous Blender

Travailler avec le plan en fond

On peut mettre le plan du bâtiment en image de fond de Blender afin de modéliser par dessus. Cela permet d'avoir une base de travail. Attention, cela ne dispense **pas** de respecter rigoureusement les cotes ! Au contraire, les plans d'évacuation ne sont pas très précis...

On va mettre en arrière-plan le plan du rez-de-chaussée. Ouvrir le panneau dont je ne connais pas le nom mais dont je me sers tout le temps (raccourci : N). Ouvrir la section Background Images et sélectionner le fichier du plan droit. Ensuite, paramétrer l'image de fond comme suit :



Le plan ne sera visible **que** lorsque vous utiliserez une vue (avec le pavé numérique, et vous pouvez paramétriser pour quelles vues il s'affiche dans Axis) **ET** que vous êtes **en mode orthogonal** (raccourci : 5 sur le pavé numérique).

Guide : modéliser un étage du DI sous Blender

Modélisation

Règles générales

Faire un maillage propre ! C'est très important si on veut s'y retrouver et faire facilement des modifications. Pour ça, la règle principale est de **ne jamais se faire croiser les arêtes**.



Aussi, je pense que c'est une bonne idée de séparer les différentes parties du bâtiment. Par exemple : sol, murs extérieurs, murs intérieurs, détails ajoutés (tableaux, barrières...). Ca permet de les séparer en calques pour y voir plus clair, et aussi, parfois, de minimiser le nombre de vertice du mesh.

Voir tutoriaux

Comment faire ?

Faire le maillage en 2D :

Il faut subdiviser le plan de départ en de nombreuses sections. Cela doit permettre non seulement de dessiner les murs, mais aussi d'ajouter plus tard les portes et autres détails.

Extruder les murs

Le plus simple ! Vous pouvez le faire en une seule fois puis couper horizontalement les murs pour ajouter les vertice/arêtes nécessaires à la création des portes et fenêtres, ou bien le faire en plusieurs fois pour directement créer ces vertices/arêtes.

Ajouter les détails (encadrements de porte, fenêtres, renfoncements de murs...)

Tutoriaux

Créer le maillage et les murs :

<https://www.youtube.com/watch?v=FjVCQPOon7g>

Conseils généraux et en particulier, à la fin, sur la façon de séparer les éléments du mesh :

<http://playuptools.blogspot.fr/2011/08/part-iii-general-advice-on-modeling.html>

Guide : modéliser un étage du DI sous Blender

Raccourcis utiles

Ce sont ceux que j'utilise le plus.

Z : passer en vue filaire/vue solide

1, 3, 7 : vues orthogonales. J'utilise surtout 7 puisque je crée le plan 2D vue de haut puis que j'extrude les murs.

Shift + clic droit : ajouter un soustraire un élément à la sélection

B : cadre de sélection. Attention, les éléments ainsi sélectionnés ne remplacent pas la sélection courante mais s'y ajoutent.

F : crée une arête (si deux points sont sélectionnés) ou une face (si deux arêtes ou plus sont sélectionnées). On peut ne sélectionner que deux arêtes opposées pour créer un quadrilatère. Cependant si la face a plus de 4 côtés je vous conseille de sélectionner toutes les arêtes par précaution.

W, puis **Merge** : pour fusionner deux vertex ou plus en un seul, utile pour corriger les erreurs de maillage ou créer un triangle à partir d'un quad

W, puis **Subdivide** : utile pour ajouter un point à une arête.

Ctrl+R : couper une ou plusieurs faces de façon droite. Utile pour rajouter des points bien alignés.

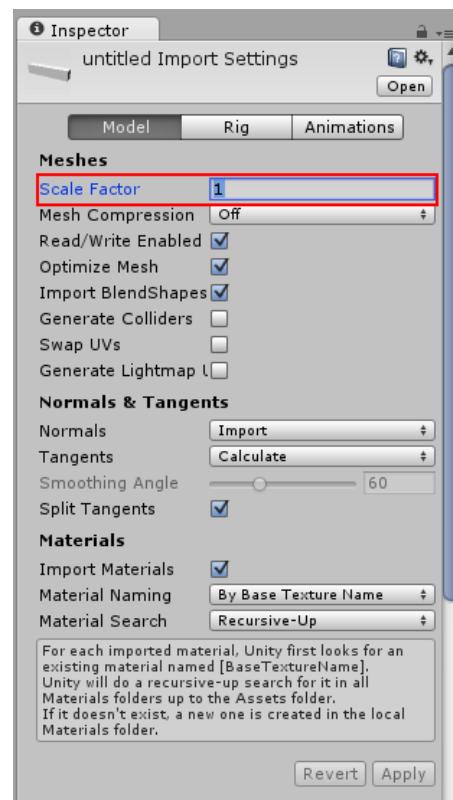
E : extruder (une face pour créer du volume, une arête pour créer une face)

Importation dans Unity

Dans Blender, exporter le fichier au format .fbx. Il faut l'enregistrer dans le dossier Assets du projet Unity pour qu'Unity le détecte automatiquement dans son explorateur.

Dans Unity, sélectionner le .fbx. L'inspecteur s'ouvre dans le volet droit. Dans l'inspecteur, le paramètre Scale Factor est à 0.01 par défaut. Il faut le changer à 1. Enregistrer les changements.

Il n'y a plus qu'à faire glisser le mesh sur la scène pour l'instancier avec les bonnes dimensions.



Annexe B Guide d'utilisation de Git

Guide : GitHub et Unity

Pour pouvoir mettre en commun notre travail, tout en nous permettant de travailler à plusieurs sur le même fichier, il est nécessaire d'utiliser un logiciel de gestion de version. Nous utiliserons dans ce cadre le logiciel Git, et l'interface Web Github. Il vous faudra ainsi tout d'abord pour inscrire sur GitHub via le lien suivant : <https://github.com/>

Installer Git :

Se référer à la page suivante :

<http://openclassrooms.com/courses/gerer-son-code-avec-git-et-github/installer-git>

Cela installer Git et GitBash, nécessaire pour utiliser git dans un environnement Bash

Installer le Client GitHub :

C'est ici : <https://windows.github.com/>

Une fois installé, allez dans les options, et régler le shell par défaut sur GitBash.

Forker et Cloner le dépôt :

Pour pouvoir travailler sur le projet, il faudra tout d'abord Forker le projet principal disponible dans ma liste de dépôt disponible ici :

<https://github.com/DadEap>

Pour cela, il faut cliquer sur le lien fork en haut à droite.

Le projet sera alors dans votre liste de projet. Vous pourrez alors cloner le dépôt sur votre disque en utilisant le client GitHub (la croix en haut à droite).

Il y a en tout deux branches. La branche master est réservée garder un projet clean dans le pire des cas. Nous travaillerons donc sur la branche **MainBranch**.

Une fois le projet cloné et vérification que tout s'est bien passé, c'est bon ! Il ne vous reste plus qu'à faire vos modifications. Quand vous voulez sauvegarder votre avancement, il vous suffit de commiter via le client GitHub en faisant attention à bien mettre un nom/description cohérent pour pouvoir s'y retrouver.

Pull/ Push

Il faut changer le repository d'origine. Pour cela, lancer votre invite de commande git, et tapper :

git remote set-url origin <https://github.com/DadEap/PolySlender>



Le bouton synchroniser en haut à droite est l'un des plus importants. C'est grâce à lui que vous pourrez pull les changements du projet de ma version, et de pusher sur votre répertoire.

Transmettre votre travail à la source

Une fois votre fonctionnalité terminée, et prête à être intégrée au projet principal, il vous faut faire une requête de Pull. Il faut pour cela cliquer dans la zone verte entourée en rouge.

Résumé ici : <https://help.github.com/articles/creating-a-pull-request/>

Slender in Polytech Tours ! — Edit

This screenshot shows a GitHub repository summary for the 'PolySlender' project. At the top, there are statistics: 4 commits, 2 branches, 0 releases, and 1 contributor. Below this, a green button labeled 'Pull Request' is circled in red. The main area displays the 'MainBranch' with one commit ahead of 'DadEap:MainBranch'. The commit list includes 'Assets', 'NewDossier', 'ProjectSettings', and 'README.md', all authored by 'DadEap' 40 minutes ago. There are also 'Pull Request' and 'Compare' buttons at the top of the commit list.

commit	author	time
Assets	DadEap	40 minutes ago
NewDossier	DadEap	40 minutes ago
ProjectSettings	DadEap	2 hours ago
README.md	DadEap	40 minutes ago

La fenêtre récapitulative du pull s'ouvre, vérifier que tous ce que vous voulez Pull est présent.

 TestDude37 / PolySlender
forked from DadEap/PolySlender

[Unwatch](#) 1 [Star](#) 0 [Fork](#) 1

base fork: DadEap/PolySlender base: MainBranch ... head fork: TestDude37/PolySlender compare: MainBranch

[Create pull request](#) Discuss and review the changes in this comparison with others.

- 1 commit 4 files changed 0 commit comments 1 contributor

Commits on Feb 16, 2015

DadEap commit 0f9f94d

Showing 4 changed files with 2 additions and 0 deletions.

Unified Split

BIN Assets/1erEtage_7.blend View

Binary file not shown

BIN Assets/1erEtage_7.blend View

Binary file not shown

Ensuite, il vous est demandé de nommer et de décrire votre Pull. Pour cette phase je vous demanderais d'être précis pour pouvoir s'y retrouver facilement.

 TestDude37 / PolySlender
forked from DadEap/PolySlender

[Unwatch](#) 1 [Star](#) 0 [Fork](#) 1

base fork: DadEap/PolySlender base: MainBranch ... head fork: TestDude37/PolySlender compare: MainBranch

Pull du script de déplacement SlenderMan

Write Preview Markdown supported Edit in fullscreen

Message décrivant le contenu du Pull

Attach images by dragging & dropping or selecting them.

 Able to merge.

These branches can be automatically merged.

[Create pull request](#)

- 1 commit 4 files changed 0 commit comments 1 contributor

Une fois la demande faite, il ne me restera plus qu'à l'accepter via mon profil :

commit #1

Open TestDude37 wants to merge 1 commit into `DadEap:master` from `TestDude37:master`

Conversation 0 Commits 1 Files changed 4 +2 -0

TestDude37 commented 35 minutes ago tu capte ?

commit 0f9f94d

This pull request can be automatically merged.
You can also merge branches on the [command line](#).

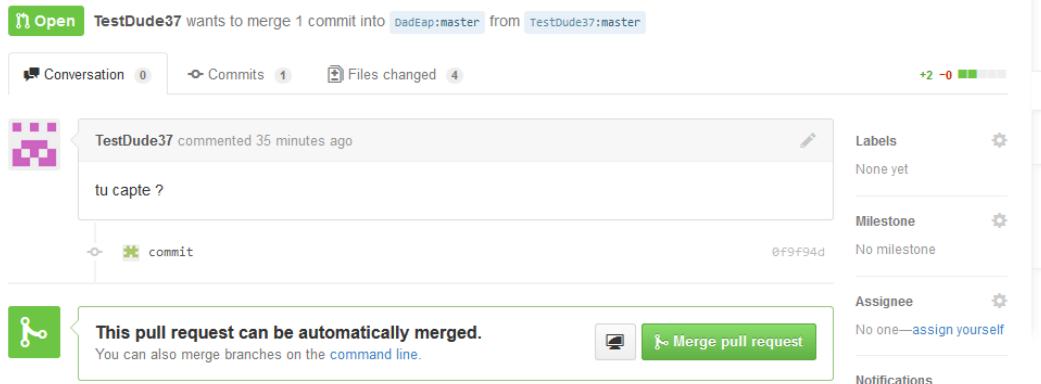
Merge pull request

Labels None yet

Milestone No milestone

Assignee No one—assign yourself

Notifications



Annexe C : Guide de pose des textures

Guide de texturing

[Intro, définitions](#)

[Textures \(avec image\)](#)

[Tout d'abord](#)

[Sur Blender](#)

[Dépliage d'UVs](#)

[Affectation de l'image](#)

[Sur Unity](#)

[Material uni, sans image \(vitres, surfaces brillantes...\)](#)

[Troubleshooting \(FAQ\)](#)

[UV mapping](#)

[Mes UVs ne s'affichent pas](#)

[Mes UVs font juste un point](#)

[Materials dans Unity](#)

[Mon Material dans Unity s'appelle None _xxx](#)

[Mon Material dans Unity s'appelle Material _xxx](#)

[Unity ne met pas les bonnes images sur les bonnes faces](#)

Intro

Ce guide a pour but de détailler la marche à suivre pour correctement importer les textures et Materials d'un objet de Blender vers Unity.

Material = en gros le render state de l'objet. Définit sa couleur, sa brillance, le shader utilisé pour effectuer le rendu, l'image de texture...

Texture = l'image utilisée dans le Material. Dans Blender, on peut affecter une texture dans effector de Material, c'est pour ça que je fais la différence entre les deux. (Unity ne peut pas affecter une texture sans Material : il créera un Material qui utilise la texture qu'on aura affectée avec Blender)

UVs : ça représente les texels, comme on a vu en cours. Ce sont des coordonnées dont on se sert pour appliquer une texture (2D) à un objet (3D) et pour dire à quel endroit chaque partie de l'image doit aller sur le modèle 3D.

Guide de texturing

Textures (avec image)

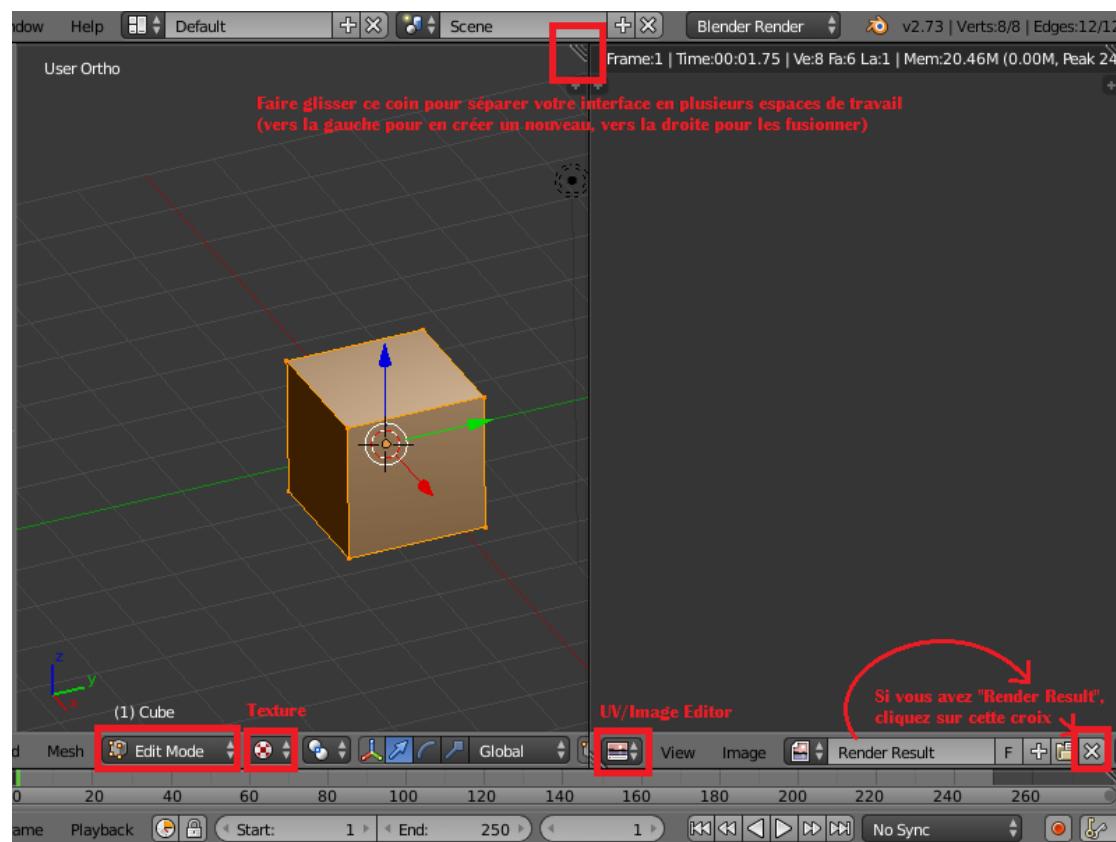
Tout d'abord

Récupérez l'image que vous souhaitez utiliser comme texture. Il y en a déjà un certain nombre dans le dossier Textures de ce dossier partagé. Vous devez placer cette image dans le projet Unity (dossier Assets ou sous-dossier, il faudrait qu'on se mette d'accord sur l'arborescence des fichiers)

Sur Blender

Dépliage d'UVs

Séparer votre espace de travail en deux. Dans votre espace de travail de gauche, mettez-vous en edit mode et affichez les textures. Dans votre espace de travail de droite, sélectionnez la vue UV/Image editor. Cliquez sur la croix à côté de Render result pour pouvoir visualiser les UVs.



Ensuite, on va déplier les UVs de cet objet. Dans l'interface de gauche, sélectionnez les faces que vous voulez texturer, et utilisez le raccourci **u** ("u" comme "unwrap") pour avoir les options de dépliage. Pour des objets comme les murs, portes, sols, ce qui marche le mieux

Guide de texturing

est généralement cube projection. Vous pouvez aussi utiliser project from view, attention alors à bien se placer en face de la face qu'on veut déplier. Un bon dépliage d'UVs a des bords droits (enfin ça dépend de la gueule de votre objet aussi) et respecte les proportions de l'objet. Parfois il est plus simple de le faire face par face que tout l'objet d'un coup, surtout s'il est complexe ou pas très géométrique. Ajustez les UVs à droite en regardant le résultat sur votre objet à gauche. Les raccourcis pour manipuler les UVs sont :

g : déplacer

s : agrandir/réduire

r : rotation

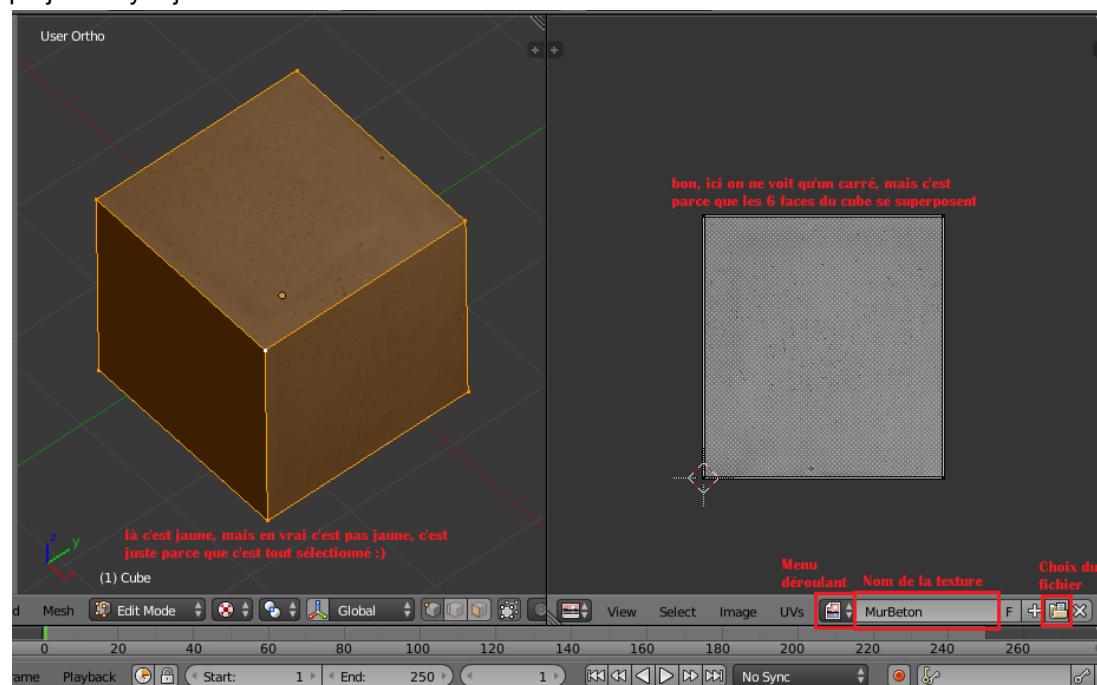
et, comme dans les autres modes :

b : sélectionner tous les vertex d'une zone

ctrl + : sélectionner tous les vertices adjacents à ceux déjà sélectionnés

Affectation de l'image

Si vous avez déjà utilisé cette image dans votre fichier Blender, sélectionnez celle-ci dans le menu déroulant. Sinon, cliquez sur open et sélectionnez l'image qui a été placée dans le projet Unity. Ajustez vos UVs.



Attention : le nom de la texture sur Unity sera le nom de la texture que vous utilisez. Par défaut, c'est le nom du fichier que vous avez sélectionné. Je vous demande de systématiquement renommer la en supprimant son extension. Cela permettra d'avoir dans Unity des textures nommées par exemple, SolGris, alors qu'en gardant l'extension elle se nommera SolGris.jpg, et c'est moche :)

Guide de texturing

Sur Unity

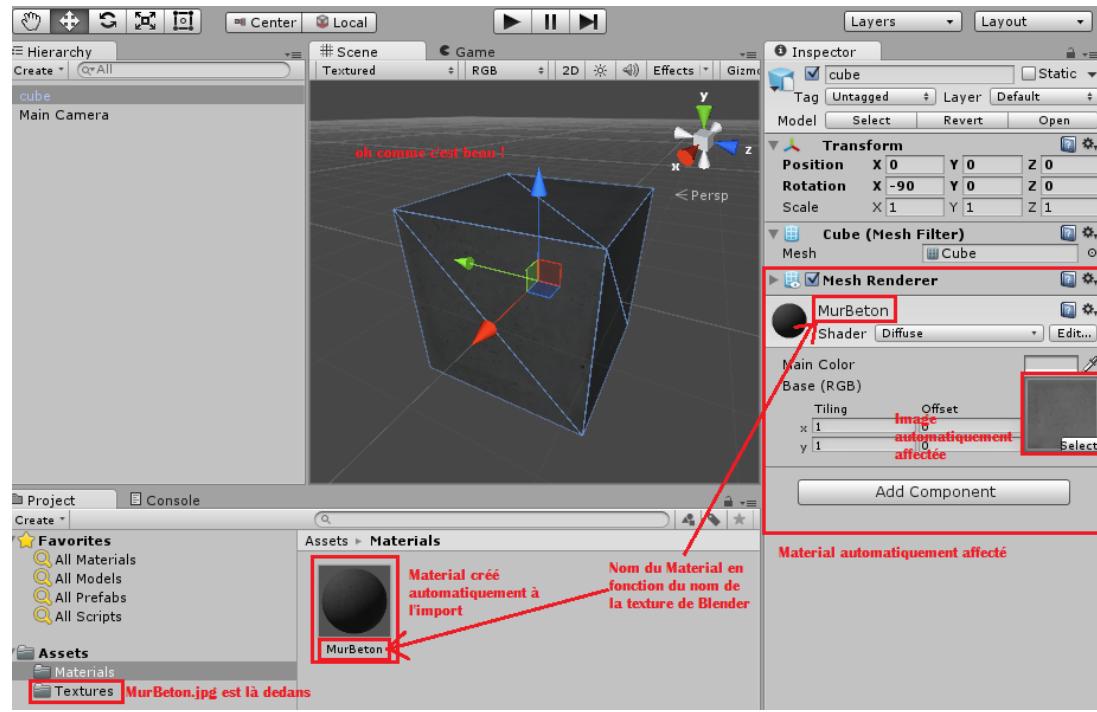
Unity utilise son propre système de Materials, qui est différent de celui de Blender.

Heureusement, il sait faire la conversion la plupart du temps.

Lorsque votre objet est importé dans Unity, Unity va chercher les textures dans son dossier

Assets, crée les Materials nécessaires et les affecte conformément à votre mappage d'UVs.

Chaque Material doit porter le nom de la texture qu'il applique.



Material uni, sans image (vitres, surfaces brillantes...)

En théorie : il suffit d'affecter le Material dans Blender, et quand on l'importera dans Unity, il créera le Material équivalent et celui-ci portera le même nom que le Material de Blender.

En pratique : ça fait de la merde, et ça m'énerve. J'ai besoin d'encore y travailler pour vous donner quelque chose de... "déterministe".

En attendant : j'affecte une texture bidon aux objets voulus, puis une fois dans Unity je modifie le Material correspondant. Cela m'a permis par exemple d'obtenir des vitres transparentes.

Guide de texturing

Troubleshooting (FAQ)

UV mapping

Mes UVs ne s'affichent pas

- Vérifiez que vous avez fermé le “render result” en cliquant sur la croix (X) à sa droite.
- Les UVs ne s'affichent qu'une fois qu'ils ont été dépliés, il faut donc d'abord choisir un mode de dépliage en faisant **u** en edit mode.
- Vérifiez que vous n'êtes pas trop zoomé, trop dézoomé, etc...

Mes UVs font juste un point

- Choisissez un autre mode de dépliage d'UVs (en général, unwrap... ça marche pas bien !). Je conseille cube projection, ou project from view (en étant bien en face de la face).
- Vérifiez que vous n'êtes pas trop dézoomé

Materials dans Unity

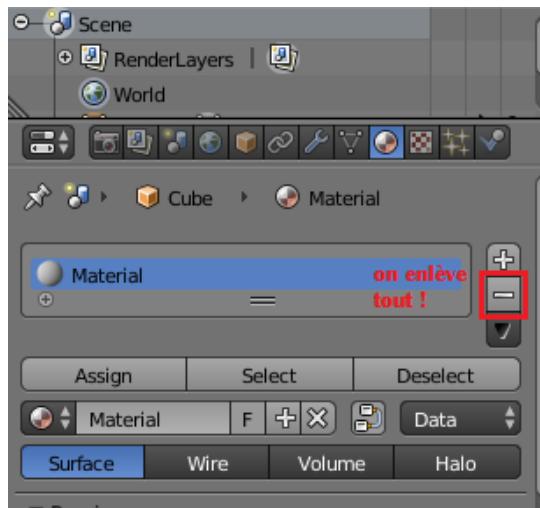
Mon Material dans Unity s'appelle None__xxx

- Unity nomme ainsi le Material s'il ne trouve pas l'image associée au moment de l'import. S'assurer que l'image est bien dans le projet, que c'est bien celle-ci qui est sélectionnée dans l'UV editor de Blender, supprimer ce Material, et réimporter l'objet.

Mon Material dans Unity s'appelle Material__xxx

En réalité Unity nomme les Materials de la façon suivante : NomMaterial si c'est un Material de Blender, NomMaterial__NomTexture si un Material ET une texture sont affectés à l'objet, et NomTexture s'il y a seulement une texture. Si votre Material (Unity) se nomme Material__xxx, c'est probablement que le Material (Blender) par défaut est appliqué à votre objet dans Blender. Pour le supprimer, le sélectionner en object mode et cliquer sur le - dans l'onglet des Materials dans Blender.

Guide de texturing



Unity ne met pas les bonnes images sur les bonnes faces

J'ai rencontré ce problème lorsque j'ai retiré de certaines faces la texture que j'avais appliquée. J'ai l'impression qu'Unity l'a mal pris et a décidé de remettre une texture au hasard sur ces faces... S'assurer que toutes les faces ont bien une texture d'affectée, donc. Si ça ne marche pas... je sais pas :/

Compte-rendu

Mise en œuvre collective

Résumé :

Mots clefs :

Abstract:

Keywords: **Enseignant**

Sébastien Aupetit

sebastien.aupetit@univ-tours.fr

Étudiants

David Boas

david.boas@etu.univ-tours.fr

Nicolas Fontaine

nicolas.fontaine@etu.univ-tours.fr

Guoxiang Liu

guoxiang.liu@etu.univ-tours.fr

Charlie Mathonnet

charlie.mathonnet@etu.univ-tours.fr

Bastien Meunier

bastien.meunier@etu.univ-tours.fr

Armand Renaudeau

armand.renaudeau@etu.univ-tours.fr

Li Wenlong

li.wenlong@etu.univ-tours.fr