CONCEPTION D'UN OUTIL DE GÉNÉRATION PROCÉDURALE D'ENVIRONNEMENTS URBAINS DESTINÉ À LA PRODUCTION DE JEU VIDÉO

par

Thibault PRIN

MÉMOIRE PRÉSENTÉ À L'ÉCOLE DES ARTS NUMÉRIQUES, DE L'ANIMATION ET DU DESIGN COMME EXIGENCE PARTIELLE À L'OBTENTION DE
LA MAÎTRISE EN ART

MONTRÉAL, LE JEUDI 29 MARS 2018

ÉCOLE DES ARTS NUMÉRIQUES, DE L'ANIMATION ET DU DESIGN
UNIVERSITÉ DU QUÉBEC À CHICOUTIMI

PRÉSENTATION DU JURY

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE :

M. Jocelyn Benoit, directeur de thèse
École des arts Numériques, de l'Animation et du Design, UQAC

M. Pierre Tousignant, président du jury
École des arts Numériques, de l'Animation et du Design, UQAC

M. Luc St-Onge, examinateur externe
UBISOFT MONTRÉAL

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE MARDI 17 AVRIL 2018

À L'ÉCOLE DES ARTS NUMÉRIQUES, DE L'ANIMATION ET DU DESIGN
UNIVERSITÉ DU QUÉBEC À CHICOUTIMI

REMERCIEMENTS

Merci aux membres du jury d'avoir accepté et pris le temps d'évaluer ce travail.

Merci à Jocelyn Benoit, mon directeur de recherche, qui a cru en mon sujet et qui m'a guidé tout au long de cette maîtrise, durant le développement du projet mais surtout durant la phase de rédaction de ce mémoire, accompagnée de craintes, d'incertitudes et d'hésitations. Merci pour le temps qu'il m'a consacré, les conseils qu'il m'a apportés et la patience qu'il a eue lors de la rédaction et de la correction de ce mémoire.

Merci aux amis qui m'ont soutenu et qui étaient là pour m'encourager. Merci à Mylène et Oli de m'avoir aidé à me concentrer sur ma maîtrise et de ne pas m'avoir laissé (trop) me disperser à travers certaines activités de type vidéoludique. Merci à Sophie et Myriam, mes amis diplômés de maîtrise qui m'ont montré la voie et qui m'ont convaincu que j'allais y arriver à mon tour. Merci également à Vincent Nesme, collègue et ami, qui a participé activement à l'élaboration de ce projet. En plus des fonctionnalités indispensables qu'il a apportées, je souhaite le remercier pour son soutien, sa bonne humeur et sa compagnie dont il a fait preuve durant ces mois de collaborations.

Merci à mes parents et à ma famille, eux qui m'ont soutenu quels que soient mes choix. C'est grâce à eux que j'ai pu concrétiser cette expérience à Montréal. Merci pour tout ce que vous avez toujours fait pour moi. Merci à Laurent qui a passé du temps à lire et relire ce document lors de la correction.

Merci à Alizée, toi qui m'accompagnes depuis plus de six ans, d'être là à chaque instant, de croire en moi dans les moments d'incertitudes comme dans les moments de joie, et dont la présence à mes côtés fait de moi un homme meilleur.

CONCEPTION D'UN OUTIL DE GÉNÉRATION PROCÉDURALE D'ENVIRONNEMENTS URBAINS DESTINÉ À LA PRODUCTION DE JEU VIDÉO

Thibault PRIN

RÉSUMÉ

Au fil des dernières décennies, l'évolution de la capacité de calculs et de la qualité des outils a permis d'augmenter la qualité visuelle des jeux vidéo. En conséquence, l'effort et le coût nécessaires pour créer ces jeux ont explosé. Il y a donc un besoin concernant la création d'outils afin de permettre aux artistes d'automatiser certaines tâches non-créatives, notamment à ce qui a trait à la génération d'environnement. Ce mémoire a pour objectif de décrire les différentes étapes de la création d'un outil permettant de générer de manière semi-automatique un environnement urbain. Une des contraintes de cette recherche est que cet outil doit pouvoir être utilisé dans un cadre de production de jeu vidéo. Pour cela, l'outil doit répondre à un certain nombre de critères, comme pouvoir laisser l'artiste décider de l'aspect visuel des bâtiments ainsi que du plan du réseau routier pour répondre aux décisions prises par la direction artistique. Ce mémoire fait dans un premier temps un état des lieux des travaux existants dans le domaine de la génération procédurale en milieu de jeu, et plus précisément sur les méthodes de génération procédurale d'environnements urbains. Dans une deuxième partie, ce document présente les différentes étapes de création nécessaires pour obtenir un outil de génération procédurale d'environnements urbains. Enfin, les résultats obtenus avec l'outil en question sont présentés, ouvrant la porte à une discussion sur d'éventuels compléments et travaux futurs. Ce mémoire est accompagné d'une création, un outil de génération procédurale permettant à l'artiste de générer une zone urbaine peuplée de bâtiments paramétrables à partir de modules qu'il aura préalablement créé.

Mots-clés : génération procédurale, outil, ville, bâtiments, jeu vidéo

CONCEPTION OF A PROCEDURAL URBAN ENVIRONMENTS GENERATION TOOL DESTINED TO VIDEO GAME PRODUCTION

Thibault PRIN

ABSTRACT

During the last decades, the evolution of computation capacity and the quality of tools allows to improve the visual quality of video games. Consequently, effort and cost needed to create such games exploded. There is therefore a need for the creation of tools to allow artists to automate certain non-creative tasks, including those related to environment generation. This master's thesis aims to describe the different stages of the development of a tool for semi-automatically generating an urban environment. One of the constraints of this research is that this tool must be operational in a context of video game production. To do so, the tool must meet a certain number of criteria, such as being able to let the artist determine the buildings' visual aspect, or the road network scheme chosen by the artistic direction. Firstly, this master's thesis analyzes literature reviews about PCG, and precisely urban environment generations. Secondly, this document presents the different steps needed to obtain an urban environment procedural generation tool. Then the results obtained with this tool are shown, leading to a discussion and an opening on potential future works on the subject. This master's thesis is accompanied by a creation, a procedural generation tool allowing the artist to generate an urban area populated with configurable buildings made from modules built by the artist.

Keywords: procedural generation, tool, city, buildings, video game

TABLE DES MATIÈRES

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

2D	Deux dimensions
3D	Trois dimensions
IPS	Image par seconde
LOD	*Level of details*
PC	*Personal computer*
PCG	*Procedural content generation*

INTRODUCTION

De nos jours, les jeux vidéo contiennent des environnements virtuels complexes que le joueur doit traverser lors de sa progression dans l'histoire. Telle qu'explicitée par Keith (2010), l'évolution des technologies dans l'industrie des jeux vidéo a suivi la loi de Moore (1965) et, parallèlement, la quantité de travail nécessaire pour réaliser un jeu a augmenté en suivant la même courbe exponentielle. La valeur « homme-année » pour faire un jeu de niveau « AAA »[1] est passée de moins de cinq en 1980 à plus de 120 en 2005. En contrepartie, les revenus ont seulement augmenté de plus ou moins 10% par année (Keith, 2010). Par conséquent, la production d'un jeu vidéo est de plus en plus risquée et la profitabilité est loin d'être assurée en raison de l'augmentation des coûts de production. Une grande partie de ces coûts est associée à la création de l'environnement visuel (Keith, 2010, p.129).

En effet, la création d'un environnement virtuel demande beaucoup de temps et d'étapes. Le travail de l'artiste d'environnement commence lorsque le directeur artistique a défini l'aspect visuel et narratif du niveau de jeu. En suivant les contraintes et les directives de la direction artistique, les artistes commencent la modélisation en trois dimensions (3D) des éléments de contenu qui constitueront l'environnement. Selon l'importance de l'élément, un certain niveau de détails doit être obtenu. D'abord placés sans texture dans l'engin de jeu[2], ces éléments de contenu permettent d'effectuer une première phase d'éclairage. Toujours de manière itérative, une fois l'ambiance lumineuse acceptée, les artistes passent aux textures et aux matériaux qui vont habiller les modèles 3D (Totten, 2014). Tout ce travail est effectué manuellement par les artistes et demande beaucoup de temps. L'artiste doit pouvoir modifier, tester et améliorer très

---

[1] L'expression « AAA » ou « triple-A » désigne un jeu vidéo à gros budget et mondialement connu. http://ww.gameart.eu/publi/dossier/lexique/aaa.html

[2] Un engin de jeu est le logiciel dans lequel le jeu est construit. Les différents éléments comme les personnages, les décors et les animations sont importé dans ce logiciel. Ces éléments sont ensuite positionnés dans un espace 3D et leurs différentes interactions sont définies à l'aide d'un langage de programmation. Enfin, l'engin de jeu simule l'éclairage, la physiques et d'autres propriétés pour créer le jeu en temps réel.

rapidement le résultat. Ce travail itératif est nécessaire pour peaufiner les éléments constituant l'environnement afin qu'ils atteignent une qualité visuelle suffisante. La création d'un environnement est donc divisée en deux parties principales : la création des éléments et leurs placements. Le placement des différents éléments est un travail délicat et nécessaire, notamment pour éviter une répétition visible entre les différents éléments qui viendrait réduire l'authenticité et la qualité du monde virtuel (Oravakangas, 2015), mais ne constitue pas un travail gratifiant pour l'artiste et pourrait être automatisée, ce qui permettrait de gagner du temps. L'artiste aurait donc plus de temps pour se concentrer sur la partie créative de son travail.

Dans le domaine des jeux vidéo, la présence d'environnements urbains est très fréquente. Qu'il s'agisse d'un jeu de course automobile ou d'un jeu de tir à la première personne, l'environnement permet d'établir un contexte historique et visuel au jeu. C'est en partie à travers l'environnement que le joueur s'immerge dans le jeu. Devant créer un engagement chez le joueur, « [la génération d'] *environnements urbains est particulièrement difficile étant donné leur haute complexité. Le grand nombre d'éléments individuels tels que les maisons, les rues, les automobiles et autres obstacles et accessoires similaires posent un défi considérable pour les designers de niveau* » (Kruse, Sosa et Connor, 2016) . La création de tous ces composants ainsi que leur positionnement dans l'espace 3D du jeu est extrêmement chronophage et très coûteuse financièrement.

De cette situation ressort plusieurs questionnements. Comment permettre à l'artiste de se concentrer uniquement sur l'aspect créatif de son travail? L'ordinateur peut-il assister l'artiste dans sa tâche de création? La problématique de recherche à l'origine de ce mémoire est donc la suivante : comment automatiser certains aspects de la création d'environnements urbains effectuée par l'artiste pour qu'il puisse se concentrer sur l'aspect créatif de son travail, tout en lui offrant la liberté de création nécessaire pour répondre à un cadre de production de jeu vidéo?

Afin de répondre à cette problématique, ce mémoire propose un système de génération semi-automatique d'environnements urbains. L'objectif principal de cet outil est de créer des environnements urbains assez vastes dans un temps limité. Cet outil permet la création de villes composées de quartiers résidentiels. Il crée les rues et place les bâtiments dans l'environnement 3D du moteur de jeu. Cette génération semi-automatique d'environnements urbains suit les plans et les directives fournis par l'artiste. L'outil permet également d'avoir le contrôle sur les bâtiments générés, leurs aspects visuels et leurs positions. L'artiste peut ainsi remplacer ou modifier les textures et regénérer ou supprimer des façades. L'éclairage des rues et le placement des accessoires doivent cependant être traités séparément. Les objectifs secondaires de ce travail sont de répondre aux contraintes d'un cadre de production, c'est-à-dire de permettre à l'artiste de travailler avec l'outil de façon itérative, de lui donner le contrôle nécessaire sur le résultat final de la génération, tout en restant le plus simple et le plus intuitif possible, en automatisant ce qui est rébarbatif et non créatif pour l'artiste.

Ce projet de maîtrise a bien sûr des limites. Les villes générées par l'outil ne sont constituées que de rues et de bâtiments de type résidentiel. Il n'y a donc pas d'éléments comme des lampadaires, des barrières ou des poteaux, des bancs, des panneaux et autres. Même si les bâtiments peuvent répondre à différentes règles de style de générations, ils se basent sur un schéma de bâtiment rectangulaire, avec une façade principale et des murs latéraux. L'outil ne peut donc pas générer des bâtiments aux formes complexes, tels que des bâtiments en deux parties distinctes superposées.

Dans le premier chapitre, ce mémoire contient une revue de littérature qui présente l'état actuel des avancements des outils de générations procédurales de contenu utilisés pour faciliter la création de jeux vidéo. La littérature exposée dans ce mémoire met l'accent sur les avancements à ce jour et détaille les objectifs du présent travail. Dans le chapitre 2, la procédure à suivre pour utiliser l'outil est présentée et les différents paramètres du système sont explicités. Par la suite, ce mémoire met l'accent sur l'analyse des résultats et des inconvénients du système. Finalement, la conclusion résume les éléments importants du mémoire.

CHAPITRE 1

ÉTAT DE L'ART

Dans le monde du jeu vidéo, la génération procédurale de contenu, communément appelée PCG, est une approche utilisée dans le but de créer une grande quantité d'éléments de contenu, de manière automatisée, en se basant sur un ensemble de règles ou de grammaires.

Ce chapitre présente les différentes utilisations et applications des approches de PCG, comme la génération de niveaux, de comportements, d'armes, de terrains ou de villes, de végétations ou encore d'intensités de jeu. Ces différentes approches sont divisées selon deux grandes catégories : les générateurs « pendant le jeu » et les générateurs « pendant la production » (Togelius, Shaker et Nelson, 2016). Un générateur « pendant le jeu » génère du contenu lorsque le jeu est lancé. Par exemple, à chaque fois que le joueur commence une partie, un générateur procédural de niveau va créer un nouveau terrain, unique, construit procéduralement, amenant une expérience nouvelle à chaque tentative de jeu. De son côté, un générateur « pendant la production » génère du contenu dans le cadre de la production du jeu. À partir de quelques paramètres, un générateur de sons va par exemple permettre de générer des sons et des musiques qui sont incorporés dans un jeu, sans pour autant prendre autant de temps que l'écriture traditionnelle d'une musique.

1.1 Générateurs de contenu « pendant le jeu »

Tel qu'introduit précédemment, les générateurs procéduraux de contenu « pendant le jeu » comprennent tous les types de générateurs qui produisent du contenu lorsqu'une partie de jeu est lancée. Ils sont utilisés pour de nombreuses raisons, telles que résoudre des problèmes liés aux limitations techniques, améliorer la « *rejouabilité* » du jeu ou encore proposer des mondes

quasi infinis à explorer pour le joueur (Smith, Othenin-Girard, Whitehead et Wardrip-Fruin, 2012).

En ce qui concerne la génération de niveau, il faut remonter au début des années 1980 pour trouver les premiers exemples. À cette époque, l'approche procédurale était utilisée pour compenser les capacités limitées des ordinateurs personnels. Dans l'exemple du jeu Elite[3], l'algorithme générait de manière procédurale huit galaxies contenant chacune 256 planètes avec pour chacune des propriétés particulières en stockant ces données dans un nombre « racine »[4] (Togelius, Kastbjerg, Schedl et Yannakakis, 2011) . Cette méthode permettait ainsi d'avoir un environnement de jeu très étendu compte tenu des performances limitées des ordinateurs personnels (PC) à l'époque. Tel que vu sur la Figure 1.1, chaque point blanc constitue une planète que le joueur peut explorer.

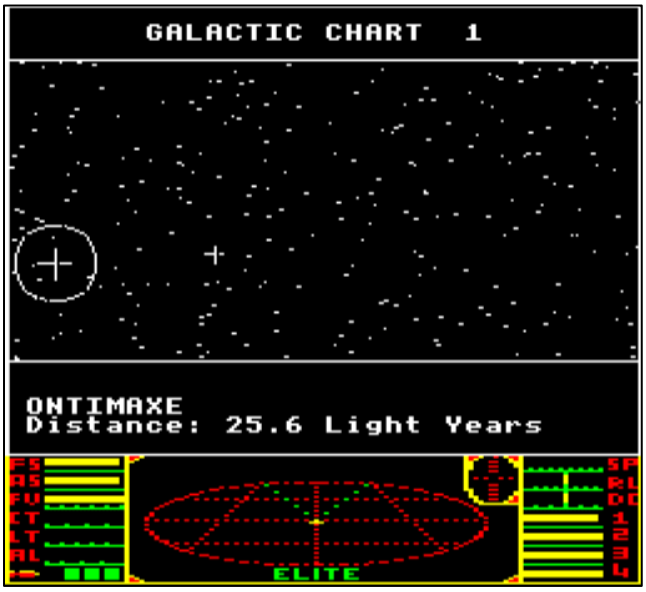


Figure 1.1 Écran du jeu Elite sur BBC Micro.
Tirée de www.gamasutra.com, 2009

[3] Elite, 1984, David Braben and Ian Bell, BBC Mirco.
[4] Le nombre « racine » est un paramètre pris en compte par le générateur procédural qui permet d'obtenir un résultat identique à chaque fois que cette valeur est utilisée.

Bell et Braben, les créateurs du jeu Elite, auraient pu y intégrer 2,82 x 10^{14} galaxies, mais Acornsoft, l'éditeur, a limité le nombre de galaxies à huit avec 256 planètes chacune. « *The final design was purposely limited by Acornsoft […] in order to hide the limitations of the game's algorithms while still creating an impressive universe.* »[5]

Toujours dans le domaine de la création procédurale de niveaux, *Rogue*[6] ainsi que ses dérivés appelés *Roguelike*, tels que *NetHack*[7] ou *Dwarf Fortress*[8], offrent une expérience de jeu intéressante car les niveaux sont regénérés à chaque fois qu'une partie est lancée. Même si ce type de jeu date du début des années 80, des *Roguelikes* sont créés encore aujourd'hui, comme *Spelunky*[9]. L'aspect procédural dans ce type de jeu permet de créer un nouvel environnement à chaque lancement de partie, c'est-à-dire de générer les couloirs et les salles de donjon, mais aussi d'y placer les montres, les pièges et les trésors cachés. Générer les niveaux en temps réel en se basant sur un algorithme permet à ces jeux de pousser le joueur à maitriser les règles du jeu et son système plutôt que sa géométrie et son environnement[10]. Cette approche procédurale ne répond pas aux contraintes de cette maîtrise pour plusieurs raisons. Le fait d'être dans la catégorie des PCG « pendant le jeu » indique que ces méthodes de générations procédurales créent du contenu lorsque le jeu est lancé, ce qui empêche l'artiste de pouvoir raffiner la qualité visuelle des environnements créés dans le cadre de la production du jeu vidéo. La seconde raison pour laquelle ce type de générateurs ne répond pas aux contraintes de cette maîtrise est visuelle. En effet, comme le jeu original Rogue, la plupart des *Roguelike* sont visuellement générés dans un format ASCII[11], c'est-à-dire que les différents éléments composants le jeu sont

---

[5] Masters of their Universe, F. Spufford. https://www.theguardian.com/books/2003/oct/18/features.weekend
[6] 1980, Michael Toy, Glenn Wichman, Ken Arnold
[7] 1987, création collective sur internet, http://www.nethack.org/
[8] 2006, Tarn Adams, http://www.bay12games.com/dwarves/
[9] 2008, Derek Yu, http://www.spelunkyworld.com/original.html
[10] http://www.gamasutra.com/view/news/262869/7_uses_of_procedural_generation_that_all_developers_should_study.php
[11] *American Standard Code for Information Interchange*, norme informatique de codage de caractères datant des années 1960.

représentés par différents caractères de l'alphabet. Le joueur est souvent représenté par le symbole « @ » et les monstres par des lettres. D'autres jeux utilisent l'approche procédurale avec le même objectif de générer des niveaux à l'infini, mais n'utilisent pas le format ASCII comme représentation visuelle. Ils sont donc présentés séparément dans ce chapitre. L'utilisation de la génération procédurale dans ce cas est très enrichissante pour l'expérience de jeu vécue par l'utilisateur, mais elle intervient seulement lorsqu'une partie de jeu est lancée. Ce type d'utilisation ne répond donc pas aux attentes de cette recherche qui souhaite utiliser les PCG lors de la phase de production de jeu.

Comme les jeux *Roguelike,* certains jeux dits « *SideScrollers* » (vue en deux dimensions[12] (2D) de profil avec l'environnement qui défile devant la caméra) utilisent une approche procédurale pour générer du contenu afin d'offrir de nouvelles expériences. Le jeu mobile *Tiny Wings*[13] en est un exemple. Dans ce jeu, la forme et l'aspect visuel des montagnes et des vallées que le personnage dévale sont générés de manière procédurale. Cependant, tout comme les jeux 2D *RogueLlike*, les jeux 2D *SideScrollers* ne répondent pas aux attentes de qualité visuelle d'un jeu « AAA ».

Les jeux 3D utilisant des méthodes de génération procédurale de contenu sont nombreux. De plus, ces jeux utilisent ces approches de génération de manières variées. Certains jeux vont utiliser la génération procédurale pour créer des mondes immenses, comme *MineCraft*[14] et *No Man's Sky*[15] qui reprennent le principe du « nombre racine » du jeu *Elite*. Même si une infinité de mondes ou de planètes peut être créée, le résultat sera toujours le même à partir du moment où le même nombre racine est utilisé pour initier la génération. On dit que cette approche procédurale est « déterministique », c'est-à-dire qu'elle permet de générer plusieurs fois le

[12] C'est-à-dire sans la notion de perspective, sans profondeur, une vue de dessus ou de profil par exemple.
[13] 2011, Andreas Illiger, http://www.andreasilliger.com/
[14] 2009, Markus Persson, https://minecraft.net/
[15] 2016, Hello Games, https://www.nomanssky.com/

même contenu en prenant en compte le même point de départ et les mêmes paramètres (Togelius *et al*., 2016). D'autres jeux, comme la série des *Diablo*[16], reprennent le principe de génération des jeux *Roguelike* et utilisent une approche procédurale pour créer des environnements de donjon, en plaçant de manière pseudo-aléatoire[17] les différents espaces, couloirs, monstres et trésors. Togelius *et al.* (2016, p. 9) disent que cette utilisation est stochastique. « *Recreating the same content [with stochastic PCG] is usually not possible* ». Une autre utilisation de la génération procédurale dans un jeu vidéo en 3D est la génération dite « adaptative » (Dormans, 2012). Ici, l'approche procédurale est utilisée pour modifier la difficulté du jeu en fonction des compétences du joueur et de ses habilités. Un exemple de cette utilisation est le jeu *Left 4 Dead*[18] qui ajuste la tension du jeu en se basant sur l'intensité émotionnelle du joueur dans le but de le tenir engagé.

L'emploi de méthodes procédurales peut aussi permettre de créer une intelligence artificielle dynamique ou évolutive (*Crusader Kings II*[19], *Borderlands 2*[20], *Shadow of Mordor*[21]). Dans ces jeux, l'intelligence artificielle est générée entièrement ou partiellement de manière procédurale, ce qui leur permet de rendre l'intrigue du jeu plus intense ou de générer des ennemis uniques qui porteront les marques des blessures d'anciens combats lorsque le joueur les rencontrera à nouveau.

Comme on peut le voir dans ces exemples, l'utilisation du PCG apporte beaucoup et permet de créer une multitude de systèmes de jeu (de « *gameplay* ») et d'interactions avec le joueur. Malheureusement, l'utilisation de ce type d'approches procédurales ne répond pas à la problématique de ce mémoire car ces approches ne sont pas utilisées dans le cadre de la

---

[16] 1996, Blizzard, http://eu.blizzard.com/fr-fr/games/d2/
[17] Le terme "aléatoire" peut être un faux-ami. Bien-souvent, une valeur aléatoire est prise en compte dans l'algorithme de génération, mais ce n'est pas toujours le cas.
[18] 2008, Valve, http://www.l4d.com/
[19] https://www.gdcvault.com/play/1020882/Emergent-Stories-in-Crusader-Kings
[20] http://www.gearboxsoftware.com/2013/09/inside-the-box-the-borderlands-2-loot-system/
[21]https://www.destructoid.com/shadow-of-mordor-s-nemesis-system-signals-the-true-beginning-of-this-generation-282160.phtml

production de ces jeux; elles ne servent pas aux artistes pour créer et générer du contenu lors de la phase de développement du jeu.

1.2 Générateurs de contenu « pendant la production »

Bien que la génération procédurale puisse être utilisée pour créer du contenu pendant l'exécution du jeu, plusieurs jeux vidéo l'utilisent plutôt pendant la période de production. Cette section présente ces différentes utilisations. Ici, ces méthodes de génération servent dans le processus de création, permettant aux artistes et aux designers de créer du contenu pour le jeu en développement. Khaled, Nelson et Barr (2013) proposent quatre métaphores pour les PCG qui permettent de bien séparer et bien présenter les différentes applications et utilisations des systèmes de PCG en production. Ces quatre métaphores servent de lignes directrices pour les sections suivantes. Ces auteurs divisent les PCG et leurs utilisations comme suit, outils, matériaux, designer et expert.

> Tools can be understood as devices or instruments manipulated for the purpose of achieving specific game design goals, that enhance and extend a designer's abilities. [...] Materials are dynamic, reconfigurable, procedurally generated substances that can be deployed and molded by the game designer. [...] Designers can be understood as PCG algorithms tasked with solving game design problems and conducting design tasks with little or no intervention from a designer. [...] We propose experts as uses of PCG related to monitoring, analyzing, interpreting, and assessing data resulting from gameplay. (Khaled *et al.*, 2013)

1.2.1 Approches procédurales en tant que matériau

Cette section présente l'utilisation de PCG en tant que matériau. Pour décrire ce type d'approche procédurale, Khaled *et al.* (2013) parlent d'une matière, d'une substance dynamique et reconfigurable que le designer peut utiliser, mouler et former afin d'obtenir un résultat satisfaisant généré procéduralement. Dans cette catégorie se retrouvent entre autres la

génération de végétation, la génération d'océans et de surfaces aquatiques, la génération de roches mais aussi la génération de démolition et de fractures. Toutes ces matières, ces éléments paramétrables et configurables permettent au designer de niveau de « peupler » son environnement.

Dans le cas de génération de plantes et d'arbres, la génération procédurale se fait à l'aide d'une succession de règles que l'on appelle grammaire : « *A (formal) grammar is a set of production rules for rewriting strings, i.e. turning one string into another. Each rule is of the form (symbol(s)) → (other symbol(s)).* » (Togelius *et al.*, 2016). Les « *L-systems* » sont une sorte de grammaire dont la particularité est la réécriture parallèle. L'interprétation graphique de ce mode de génération procédurale pourrait être comparée au déplacement d'une tortue dans un « graphique de tortue »[22]. Dans la Figure 1.2, la règle est simple : remplacer un I en Y. En répétant l'opération pour chaque nouvelle branche dessinée, on obtient au bout d'un certain nombre d'itérations une forme complexe d'arbre.

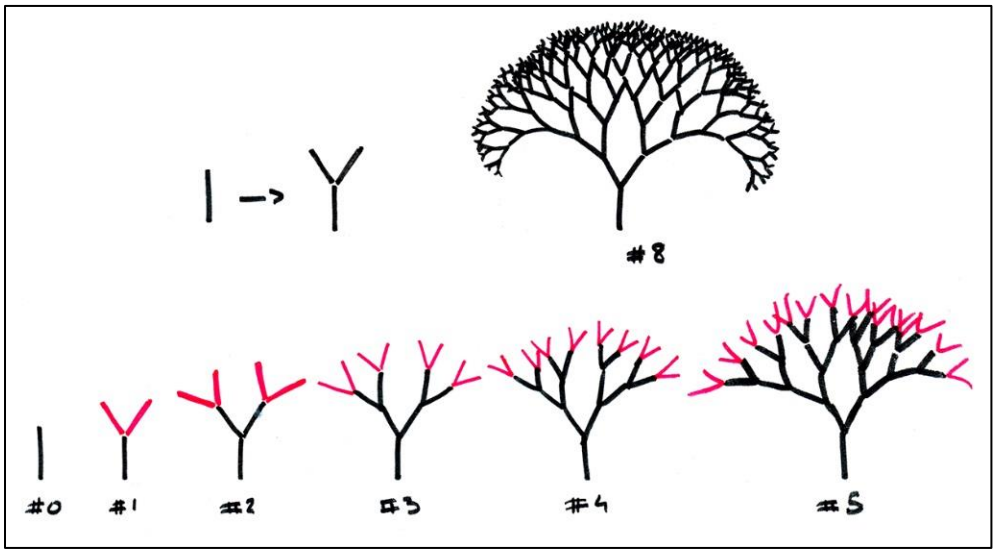


Figure 1.2 Représentation graphique d'une génération en « L-system »

[22] Ce terme vient du résultat que ferait une tortue si elle se déplaçait le long d'une feuille en tenant un stylo tout en suivant des instructions. https://www.ictlounge.com/html/control_applications_turtle_graphics.htm

En ajoutant plus de paramètres et de variables à ce principe de base, la grammaire permet de générer de la végétation terrestre (Longay, Runions, Boundon et Pruninkiewicz, 2012; Pirk, Niese, Deussen et Neubert, 2012; Shek, Lacewell, Selle, Teece et Thompson, 2010) ainsi que de la végétation sous-marine (Abela, Liapis et Yannakakis, 2015) .

L'utilisation de la génération procédurale est également présente dans le domaine de la simulation d'océans et de surfaces aquatiques (Gonzalez-Ochoa, Holder et Cook, 2012; Larsson, Zalzala et Duda, 2006; Tessendorf, 2002) . Que ce soit pour une production de film ou pour une affiche publicitaire, l'eau générée par ordinateur est devenue monnaie courante, affirme Tessendorf (2002). Dans ce type de génération, des algorithmes génèrent la géométrie des vagues, que ce soit une petite houle ou des vagues déferlantes. La réflexion et la réfraction de la lumière sont également prises en compte lors de la génération, ainsi que la filtration colorimétrique de la lumière lorsqu'elle entre et sort de l'eau.

Un autre élément généré procéduralement qui entre dans la catégorie des « matériaux » est la génération de fractures et de formations rocheuses. Différentes approches existent pour ce qui est de la génération procédurale de terrains et de formations rocheuses. Certains travaux utilisent des algorithmes paramétrables par le designer (Kamal et Uddin, 2007; Ong, Saunders, Keyser et Legget, 2005; Raffe, Zambetta et Li, 2011). Généralement, ces algorithmes influencent une « *heightmap* »[23] qui définit les différentes élévations d'une topologie. D'autres travaux utilisent des approches moins courantes, comme le travail de Génevaux, Gurin, Peytavie et Benes (2013) , qui se base sur le phénomène d'hydrologie pour former et sculpter le terrain, ou encore le travail de Becher, Kron, Reina et Ertl (2017) qui utilise des courbes primitives dans un espace 3D pour générer une volumétrie. Dans le domaine des fractures

[23] Une « heightmap » est une texture 2D en niveaux de gris qui permet de contrôler l'élévation de la topologie d'un terrain virtuel. Selon l'interprétation des logiciels, plus un pixel est blanc, plus la zone équivalente sur le maillage 3D associé sera basse, et plus le pixel est noir, plus l'élévation de la zone correspondante sera haute.

procédurales, un objet cible est divisé en sections (les différents morceaux résultants de la fracture). Pour effectuer cette division, le travail de Lee et Pavlov (2008) donne le contrôle à l'artiste pour qu'il puisse suivre les motifs de fracture établis par la direction artistique. Il peint alors directement sur l'objet 3D les « régions » qui correspondent aux débris. L'objet est ensuite décomposé en une multitude de nouveaux maillages qui représentent chaque morceau de la fracture. Garg et Maxwell (2010) proposent également à l'utilisateur de laisser l'algorithme sectionner l'objet à l'aide du diagramme de Voronoï[24].

Les travaux présentés ci-dessus sont de bons exemples de ce que Khaled *et al.* (2013) définissent comme matériaux. En effet, ces générations procédurales prennent la forme de contenus paramétrables permettant de « peupler » d'une manière ou d'une autre l'environnement virtuel du jeu. Néanmoins, ces catégories de matériaux procéduraux ne répondent pas à la problématique de ce mémoire car aucune d'entre elles ne permet de générer un environnement urbain. Une catégorie de « matériau » permet de se rapprocher de cet objectif : les générateurs procéduraux de bâtiments.

Dans le domaine de la génération procédurale de bâtiments, deux approches se distinguent : la génération par grammaire et celle par ensemble de règles et modules. Dans la première catégorie (Barret, Vance et Youngblood, 2011; Jesus, Coelho et Sousa, 2015; Larive et Gaildrat, 2006; Lipp, Wonka et Wimmer, 2008; Müller, Wonka, Haegler, Ulmer et Gool, 2006; Schwarz et Müller, 2015), la grammaire permet de décomposer une façade de bâtiment en une série de formes simples mises bout à bout. Chaque élément (porte, fenêtre, mur et angle) correspond à une lettre et un ensemble de règles dispose ces éléments de manière ordonnée pour former une façade cohérente. La Figure 1.3 est une représentation graphique du « découpage » d'une façade en éléments primaires (murs, coins, fenêtres et porte).

[24] https://originedesmotifs.wordpress.com/le-diagramme-de-voronoi/

L'utilisation de grammaires et d'ensembles de règles pour générer des bâtiments, en plus des multiples paramètres pris en compte dans la génération, permet d'obtenir une très grande variété de résultats. Le problème de ces travaux qui utilisent une grammaire de génération réside dans le fait qu'ils génèrent les façades de bâtiments à partir d'actions primaires (extrusion, déplacement, rotation, mise à l'échelle) définies par des règles de grammaire (Müller *et al.*, 2006) . Même si un certain niveau de détail peut être atteint, à force de complexifier et de détailler la génération du bâtiment, les résultats obtenus par ces travaux ne répondent pas aux attentes visuelles de l'industrie du jeu vidéo car l'artiste n'a pas un contrôle assez précis sur l'aspect esthétique des bâtiments.

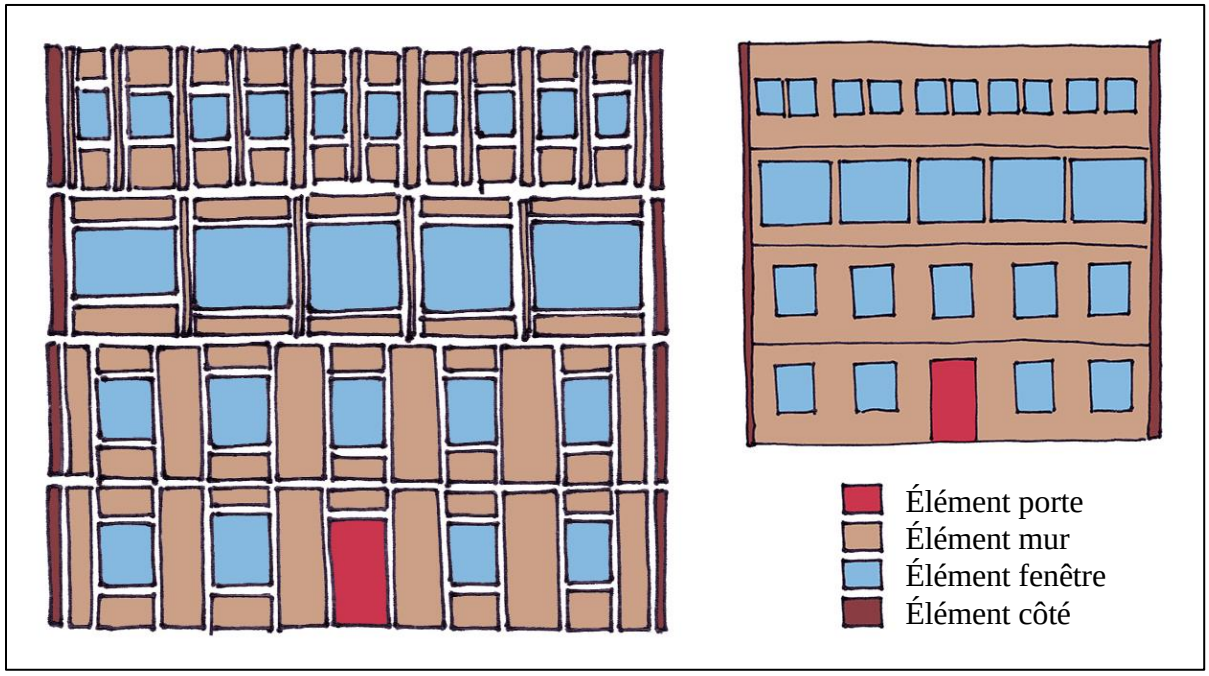


Figure 1.3 Arrangement des différents modules constituant la façade

Le deuxième groupe d'approches permettant de générer procéduralement des bâtiments ajoute une librairie de modules dans l'algorithme de génération. L'utilisation de modules permet à l'artiste qui crée ceux-ci d'apporter l'aspect esthétique et la qualité visuelle demandés par la direction artistique du jeu (Golding, 2010). Cet auteur décrit les objectifs de cette génération comme suit : « *(1) Good looking buildings with high visual density, (2) Easily change shape*

and size for gameplay, (3) Automatically générate LODs » (Golding, 2010). Le fonctionnement des approches qui utilisent une librairie de modules est généralement le suivant :

1) Le designer créé une description « haut-niveau » du bâtiment.
2) Les artistes constituent une librairie de maillages de façades rectangulaires et modulaires.
3) L'artiste crée un ensemble de règles qui décrit comment les modules de façades sont utilisés.

Cette approche de génération est très intéressante dans le cadre de cette maîtrise car elle permet de répondre (partiellement) à un des objectifs fixés; celui de répondre aux attentes visuelles et à la direction artistique prise pour la production du jeu. L'utilisateur est cependant limité à la création d'un seul bâtiment à la fois et il doit les positionner un à un pour obtenir un environnement urbain complexe et diversifié. Il serait préférable de pouvoir automatiser cette dernière tâche.

1.2.2 Approches procédurales en tant qu'outil

Selon Khaled *et al*. (2013), la génération procédurale de contenu dans un cadre de production peut être un outil de design. « De la même manière que les outils de design standard, les outils de design PCG visent à améliorer la chaine de travail (« workflow ») du designer, mais les outils PCG le font en ajoutant une composante générative. » (Togelius *et al*., 2016, p. 11). Ces outils permettent de générer de nombreux types de contenu pour le jeu, que ce soit des environnements de type urbain, des éléments d'environnement comme des paysages, des grottes ou des vallées, mais également des niveaux de jeux qui suivent des règles déterminées, comme l'outil de design de niveau 2D *Tanagra* avec lequel « l'humain et l'ordinateur peuvent travailler ensemble pour produire un niveau » (Smith, Whitehead et Mateas, 2010) . Appelée « *mixed-initiative* » (Togelius *et al.*, 2016, p. 195), cette approche de génération procédurale requiert une contribution humaine pour être utilisée. L'apport humain dans le résultat de la

génération est supérieur à la simple pression d'un bouton « générer ». De tels outils permettent par exemple à l'utilisateur d'avoir le rôle de juge par rapport à l'esthétique du résultat obtenu. C'est le cas de l'outil de génération procédurale de végétation de Keim, Simmons, Teece, Reisweber et Drakeley (2016) . Cet outil permet de générer des variations entre les espèces de végétaux, mais également entre deux végétaux de la même espèce, tout en offrant à l'artiste le contrôle nécessaire afin de respecter les contraintes de la direction artistique définie en amont. Le générateur de pistes de ski du jeu SSX[25] fait également partie de ce type d'approche procédurale. L'outil développé pour la création de ce jeu génère les pistes de ski de manière procédurale. En générant dans un premier temps un environnement simple et basique, les designers peuvent tester et corriger très rapidement la jouabilité du jeu. Dans un second temps, l'outil génère des environnements plus complexes en suivant les corrections établies. Cet aspect itératif de la génération a permis de créer des environnements de jeux complets, sans pour autant arrêter le développement du jeu ni les tests de jouabilité (Howard et Lemus, 2012).

En termes de génération de ville, plusieurs travaux procèdent de la manière suivante : (1) création d'une trame des axes routiers principaux, (2) subdivisions des parcelles obtenues et (3) génération des bâtiments peuplant la zone urbaine. Ces travaux utilisent différentes méthodes pour réaliser ces étapes. Certains génèrent le réseau routier à partir de données topographiques géolocalisées (Kelly et McCabe, 2007; Müller, 2006; Tang , 2009). Le travail de Kelly et McCabe (2007) permet de créer de larges zones urbaines en utilisant un système de grammaires de type « *L-system* » pour subdiviser le réseau routier principal obtenu à partir des données topographiques. L'utilisateur a ensuite la possibilité d'intervenir dans la génération des rues en apportant des « points de tension » ou directement à l'aide d'un « pinceau » en traçant des axes manuellement. Une fois que la grille routière répond aux attentes, l'outil génère des bâtiments en volume sous forme de parallélépipèdes permettant d'obtenir une ville complète. Le travail de Whelan, Kelly et McCabe (2008) quant à lui permet à l'utilisateur de créer directement les principaux axes du réseau routier à l'aide de l'interface

[25] EA Sports, 2012, https://www.ea.com/games/ssx/ssx-2012

de l'outil. « L'interface graphique permet la manipulation directe d'éléments structurels tels que des intersections de routes et permet également à l'utilisateur de modifier les différents paramètres qui contrôlent la génération procédurale de la route et des bâtiments » (Whelan *et al.*, 2008). Une autre solution existante se concentre uniquement sur la génération pseudo-infinie de villes et utilise un repère quadrillé pour générer un nombre donné de bâtiments (Greuter, Parker, Stewart et Leach, 2003). Tous ces travaux de génération procédurale de villes s'approchent des objectifs de cette maîtrise car ils permettent en effet de générer une ville et laissent l'utilisateur créer le réseau routier comme il le souhaite. Cela permet de répondre aux attentes des directeurs artistiques dans le cadre d'une production de jeu vidéo. Ils ne répondent cependant pas entièrement à la problématique de cette maîtrise car même si le résultat généré a une envergure satisfaisante et pourrait constituer l'environnement complet attendu, les modules utilisés pour représenter les bâtiments ne répondent pas à la qualité visuelle espérée. Même pour les outils les plus avancés tels que les travaux de Müller *et al*. (2006) et de Lipp *et al*. (2008) , les bâtiments générés n'ont pas une assez grande variété et le joueur est susceptible de rapidement remarquer la répétition des bâtiments, brisant ainsi son immersion.

1.2.3 Approches procédurales en tant que designer

Dans leur métaphore, Khaled *et al.* (2013) regroupent dans la catégorie « designer » tous les générateurs qui résolvent des tâches de design, de manière autonome, telles que déterminer « une mécanique de jeu convenable, ajouter de l'intrigue, adapter la difficulté d'un niveau de jeu ou encore prendre des décisions à propos de l'esthétique du jeu » (Khaled *et al.*, 2013, p.1513). Le système développé par Fernandez-Vara et Thomson (2012) nommé « *puzzle-dice system* », permettant de générer des systèmes de puzzles en suivant les directives données par le designer, en est un exemple. Par définition, ce type d'approche dans laquelle l'artiste joue un rôle mineur, voire inexistant, ne permet pas de répondre à la problématique de ce mémoire.

1.2.4 Approches procédurales en tant qu'expert

Khaled *et al.* (2013) séparent la catégorie « expert » en deux groupes principaux : « *player expert* » et « *domain expert* ». Le premier groupe identifie des approches qui analysent tous les types de données liées au joueur, pour adapter la difficulté d'un niveau par rapport aux compétences du joueur (Satheesh, Narasimhan et Senthil, 2009) ou personnaliser une piste de course au style de conduite du joueur (Togelius, Nardi et Lucas, 2007) par exemple. Toujours selon Khaled *et al.* (2013), le second groupe d'approches fournit une analyse et une interprétation spécifique à un domaine. Ces analyses sont ensuite intégrées dans le processus de conception du jeu. Autant les approches de la première catégorie que celles de la seconde ne répondent pas à la problématique de ce mémoire car elles assistent les développeurs du jeu avec leurs expertises, mais elles ne créent pas concrètement de contenu.

Tel qu'explicité dans la présente section, les différentes approches de génération procédurale de contenu dans le cadre de production de jeu peuvent être séparées en quatre catégories (en tant que matériel, outil, designer ou expert). L'approche de génération procédurale de contenu introduite dans ce mémoire peut être catégorisée dans « outils ». En effet, ce projet doit pouvoir assister l'utilisateur dans sa tâche de création d'environnement urbain. L'approche proposée peut également faire partie de la métaphore du « matériel » puisqu'elle permet à l'artiste de générer procéduralement des bâtiments paramétrables, configurables, dans le but de peupler l'environnement.

1.3 Objectifs

L'objectif principal de cette maîtrise est de concevoir un outil de génération procédurale de ville, destiné aux artistes d'environnements, utilisable durant la phase de production de jeu vidéo. Il est conçu pour l'artiste, dans le but d'optimiser son travail et de lui donner plus de temps pour se consacrer à la partie créative de son travail. En effet, cet outil effectue toutes les parties rébarbatives du travail de l'artiste d'environnement, tout en lui laissant le contrôle sur

l'aspect artistique. Il peut donc répondre aux attentes et contraintes des designers de niveau et concevoir l'environnement en fonction de celles-ci. Pour répondre à l'objectif de cette maîtrise et pour que cet outil soit utilisé en entreprise dans le cadre de production de jeu, il doit répondre aux critères suivants :

- L'outil doit être en mesure de générer des bâtiments paramétrables répondant à une qualité visuelle attendue par la direction artistique.
- Il doit permettre à l'artiste de générer un réseau routier plus complexe qu'un simple quadrillage. Les bâtiments générés doivent se positionner de manière automatique par rapport à ce réseau routier.
- L'outil doit donner un maximum de contrôle à l'artiste. En effet, pour répondre aux attentes et aux contraintes du designer de niveaux, l'artiste doit pouvoir accéder à un maximum de contrôles, que ce soit en amont ou en aval de la génération. Il doit pouvoir indiquer la position des rues, définir le style de bâtiments qui les constitueront, configurer des informations comme le niveau de variation dans l'utilisation des styles de bâtiments, etc. Une fois la génération faite, l'artiste doit pouvoir avoir un maximum de contrôle sur les bâtiments générés, il doit pouvoir les sélectionner facilement, les regénérer, les modifier manuellement ou les supprimer.
- Il doit être relativement simple. Il faut utiliser des paramètres empiriques qui soient intuitifs pour l'artiste. Dans une production de jeux vidéo, et particulièrement dans le cadre d'une production de jeu « AAA », le travail d'un artiste coûte cher et ce dernier a très peu de temps pour effectuer ce travail. Si l'interface n'est pas intuitive et que l'artiste doit passer trop de temps à chercher ses repères, à comprendre l'outil et son interface, l'artiste ne l'utilisera pas.

1.3.1 Positionnement de l'outil dans un contexte de production de jeu vidéo

Permettant à l'artiste d'environnement d'obtenir rapidement une zone urbaine, l'outil permet tout autant de faire des prototypes rapidement que de créer itérativement l'environnement final

durant la production du jeu. Dans l'approche de prototypage, l'outil permet d'offrir rapidement aux autres équipes de production (mécanique de jeu, gestion du multijoueur ou encore expérience utilisateur) un environnement dans lequel elles vont pouvoir travailler. L'outil peut également être utilisé pour obtenir un résultat final convenable dans certains types de jeux, comme des jeux de courses automobiles ou encore de skateboard, dans lesquels l'accent est mis sur le déplacement rapide dans la rue. Les façades défilent vite, le joueur est concentré sur son personnage ou sa trajectoire et a peu de temps pour observer en détail l'environnement. À l'opposé, un jeu de type monde ouvert dans lequel le joueur doit explorer l'environnement demanderait à l'outil d'offrir un résultat beaucoup plus poussé, ou demanderait une somme de travail manuel à postériori du résultat offert par l'outil.

Le chapitre suivant présente la méthodologie utilisée dans le cadre de cette recherche pour créer un outil qui répond à ces attentes.

CHAPITRE 2

GÉNÉRATION PROCÉDURALE D'ENVIRONNEMENTS URBAINS

Tel qu'explicité précédemment, l'objectif de ce travail de maîtrise est de créer un outil de génération semi-automatique d'environnements urbains (tel que celui de la Figure 2.1) qui est simple d'utilisation, intuitif, complet et qui donne suffisamment de contrôle à l'artiste sur l'allure du résultat final. Les sous-sections de ce chapitre détaillent les étapes à franchir pour générer un environnement urbain à l'aide de l'outil proposé.

- Création des modules et de leurs matériaux (section 2.2)
- Génération des rues et des intersections (section 2.3)
- Génération des bâtiments (sections 2.4 et 2.5)
- Définition des ensembles de règles de génération de bâtiments (section 2.6)
- Ajustements et modifications du résultat obtenu (section 2.7)

Finalement, la section 2.7 présente les résultats obtenus à l'aide de l'outil de génération procédurale d'environnement urbain proposé. Une série de rendus est présentée, explicitant différents aspects des générations obtenues. Cette section offre également une analyse sur les avantages et les limites de l'outil.

2.1 Présentation générale de l'outil

L'approche de génération semi-automatique d'environnements urbains proposée dans ce mémoire se divise en plusieurs étapes. Tout d'abord, l'artiste crée et fournit un ou plusieurs ensembles de modules de bâtiments, comme des modules de portes, de fenêtres et de murs, ainsi que les textures habillant ces éléments. L'artiste spécifie ensuite un ou plusieurs ensembles de règles, via le gestionnaire de règles, qui permettent à l'outil de générer les bâtiments selon les paramètres établis par l'artiste. À l'aide du gestionnaire d'intersections, celui-ci définit ensuite le plan des rues qui constitueront le réseau routier. L'artiste utilise enfin

l'outil de génération pour créer l'ensemble des bâtiments de la ville. Les bâtiments se positionnent de part et d'autre des rues créées précédemment par l'artiste, en suivant les intersections et les angles formés par les rues. Une fois la génération effectuée, l'artiste a la possibilité de venir modifier l'aspect visuel des bâtiments en agissant sur leurs paramètres, comme la hauteur, la position de la porte, la présence de vitrine ou d'habitation en rez-de-chaussée ou encore le matériau utilisé pour le bâtiment.

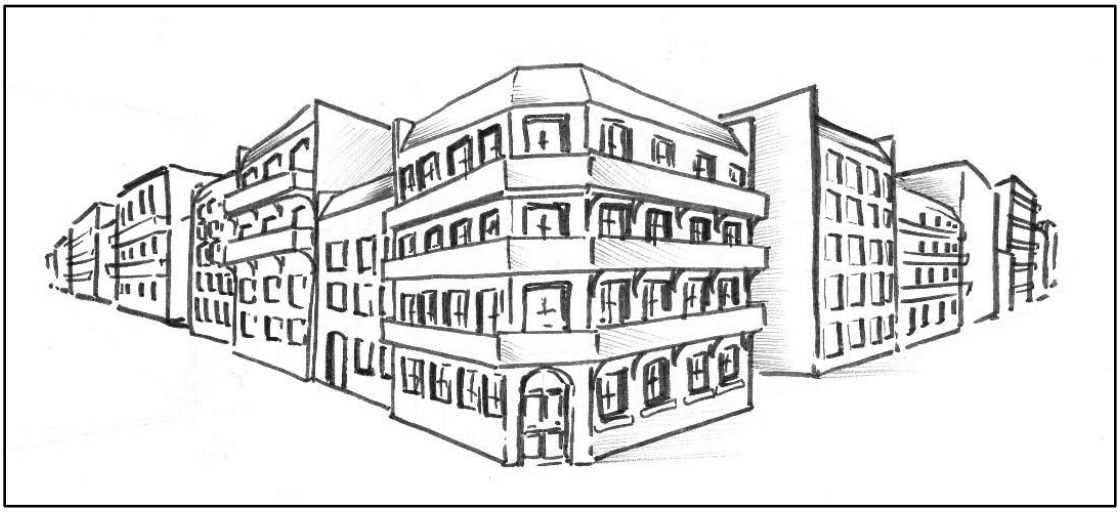


Figure 2.1 Schéma d'une zone urbaine pouvant être générée par l'outil

L'interface de cet outil est simplement constituée des différents paramètres publics disponibles dans l'engin *Unreal Engine 4*. Chaque élément de l'outil, que ce soit le gestionnaire de règles, le générateur de rues ou le générateur de bâtiments, contient ses propres paramètres dans son panneau « détails ». C'est à travers l'interface de l'engin de jeu, et plus précisément grâce au panneau « détails » de chaque élément, que l'utilisateur peut obtenir et modifier les résultats.

Bien que la génération de l'environnement urbain soit automatique, l'artiste doit tout de même fournir au système les pièces modulaires de base qui composent les différents bâtiments. La section suivante présente le processus de création de ces modules que doit suivre l'artiste d'environnement.

2.2 Création des modules et de leurs matériaux

L'approche de génération semi-automatique proposée ne s'occupe pas de la création des différents modèles 3D, mais elle s'affaire plutôt à agencer les différents modules fournis par l'artiste de façon à créer des bâtiments. Pour que l'outil fonctionne correctement, il faut donc fournir un ensemble minimal de modules : un module de porte, un module de fenêtre et un module de mur latéral. Évidemment, plus l'artiste crée de modules, plus les bâtiments sont variés. Dans le cadre de cette maîtrise, un ensemble de 20 modules (voir l'annexe 1) a été réalisé. La création de ces modules a permis de raffiner l'outil – en adaptant le code et les fonctionnalités en fonction des contraintes rencontrées – et d'établir une série de règles que les modules doivent respecter pour être utilisés dans l'outil de génération procédurale de ville.

L'échelle et l'orientation des modules de façades constituent les deux premières contraintes. Chacun de ces modules doit avoir une largeur de deux mètres, selon l'échelle de mesure du logiciel de modélisation 3D. Cette dimension imposée a été définie de façon empirique pour ce prototype, à partir de plusieurs expérimentations. Un module de façade doit être aligné sur l'axe des Y et faire face à l'axe des X de valeur positive (voir la Figure 2.2). Le troisième point important à prendre en compte est la position du point de pivot (ou point d'origine). En effet, le point de pivot d'un objet 3D détermine sa position dans un espace 3D et l'outil utilise les valeurs de position des points de pivot pour placer correctement les modules de façades côte à côte. Le point de pivot doit être dans le coin inférieur gauche du module lorsqu'il est vu de face. L'artiste peut choisir la hauteur du module de façade car l'outil s'adapte selon celle-ci. La profondeur des fenêtres et l'avancée du balcon ne doivent pas excéder deux mètres selon l'axe des X. Ces contraintes laissent amplement de liberté à l'artiste pour créer des modules cohérents.

En ce qui concerne les modules des murs latéraux (voir la Figure 2.2), ils doivent être dirigés de manière perpendiculaire aux modules de façades, orientés vers l'axe des X de valeur

négative et faire face à l'axe des Y de valeur négative. Leur hauteur peut être de dimensions variées, mais leur largeur doit faire trois mètres pour répondre aux contraintes de l'outil.

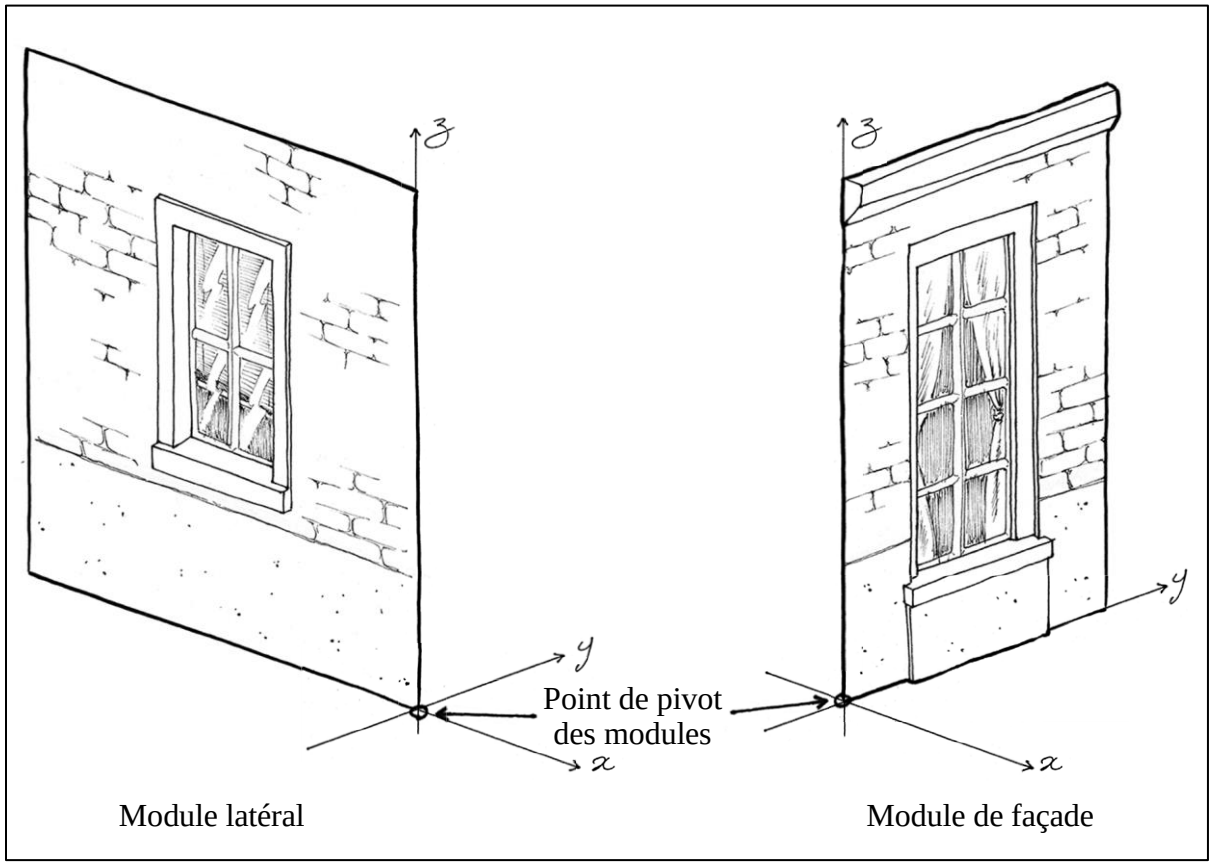


Figure 2.2 Orientation des différents modules et position de leur point de pivot

Concernant l'aspect esthétique des modules de bâtiments, l'outil a été créé pour que l'artiste ait la plus grande liberté possible. Du moment qu'il respecte les contraintes énumérées ci-dessus, l'artiste peut choisir l'époque et le style architectural qui lui conviennent. L'utilisateur peut donc générer des bâtiments de styles totalement différents, comme par exemple des bâtiments modernes new yorkais, des bâtiments européens de type haussmannien ou encore des bâtiments répondants à une architecture futuriste. Actuellement, l'outil accepte sept types de modules. En plus des modules de porte, de fenêtres, de murs, de mur arrière et de toit, l'outil peut aussi accepter des vitrines et des « fenêtre rez-de-chaussée ». Ces deux derniers éléments

viennent compléter le premier étage du bâtiment avec la porte. Un paramètre permet à l'artiste de choisir entre ces deux options.

Tandis que les « fenêtres rez-de-chaussée » doivent répondre aux mêmes contraintes que les fenêtres normales, les vitrines offrent une liberté supplémentaire. En effet, la largeur des vitrines peut être un multiple de deux mètres. Un ensemble de six modules de vitrines a été modélisé ayant comme largeur deux mètres, quatre mètres, six mètres et dix mètres. En fonction de la largeur totale du bâtiment, l'outil sélectionnera de manière aléatoire les vitrines et adaptera la sélection pour que la longueur additionnée des modules sélectionnés soit égale à la longueur totale de la façade (voir Figure 2.3).

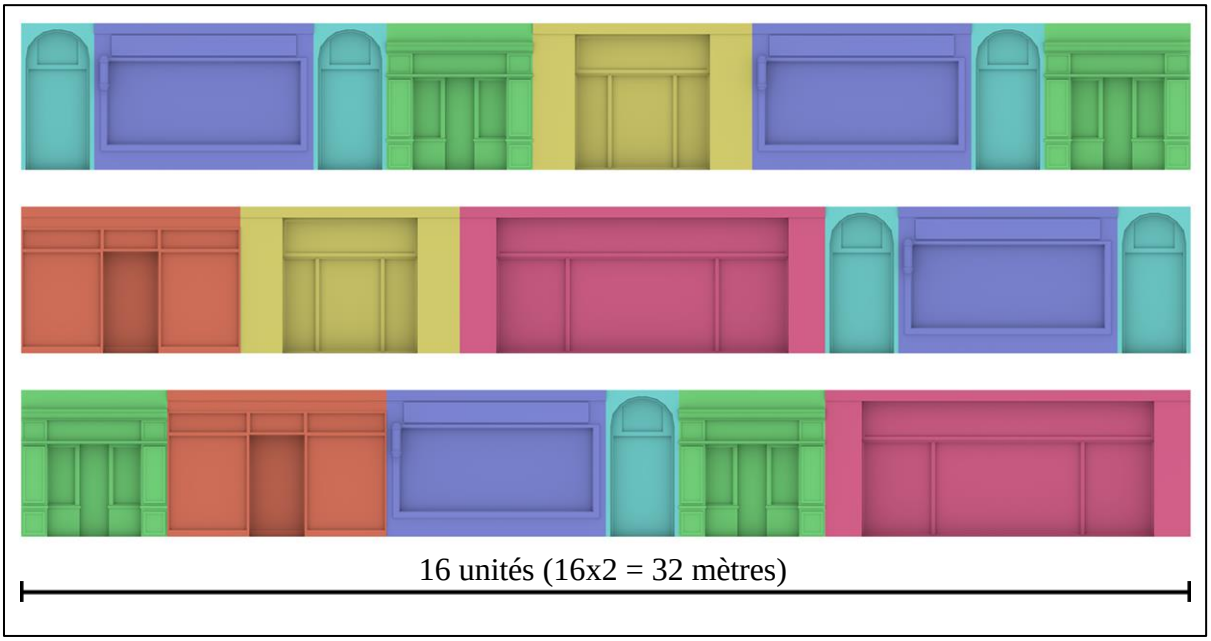


Figure 2.3 Exemples d'agencements de "vitrines" sur une longueur de 16 unités

2.2.1 Matériaux paramétrables

Lors de la création d'environnements virtuels, les matériaux jouent un rôle important. En effet, les matériaux, à l'aide des textures et des différents réglages qu'ils offrent, permettent de

donner un aspect réel à un ensemble de polygones virtuels. Ce sont les matériaux qui définissent la teinte, la réflexion, la transparence ou encore la luminance de la surface modélisée. La Figure 2.4 est une série d'exemples d'aspects que l'on peut obtenir à partir du même maillage polygonal en changeant de matériaux.

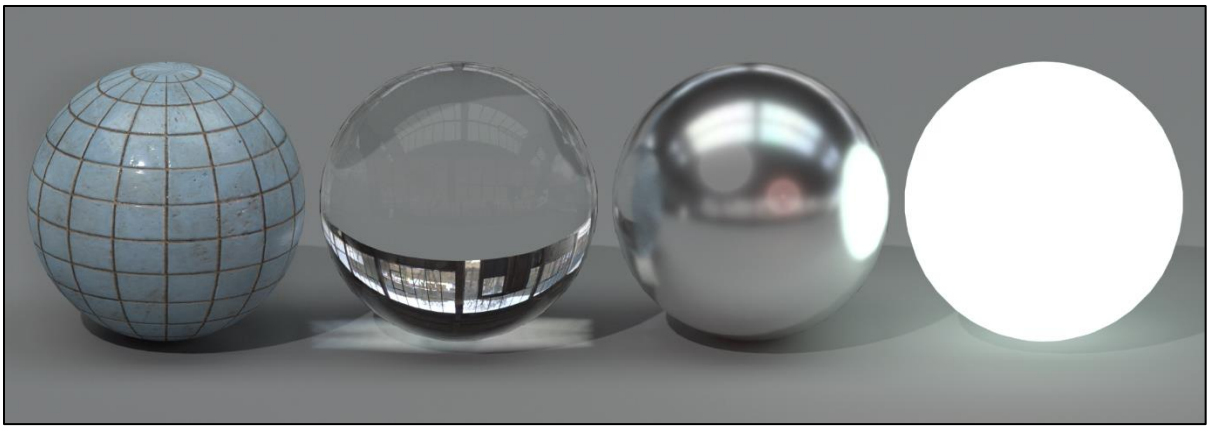


Figure 2.4 Différents exemples de matériaux

Pour utiliser cet outil, l'artiste doit fournir, en plus des modules de façade, deux librairies de textures liées aux modules. La première librairie constitue l'ensemble de textures représentatives des éléments « spécifiques » tels que les portes, les cadres de fenêtres, les devantures de vitrines ou les armatures de balcons. Ces textures sont liées à un maillage particulier, au contraire de la seconde librairie qui constitue une banque de textures « interchangeables ». Ces dernières représentent le matériau appliqué sur les murs et leur interchangeabilité permet d'appliquer différents matériaux aux sections murales des modules. On peut voir dans la Figure 2.5 que différents matériaux ont été appliqués au même ensemble de modules, ce qui permet de diversifier le résultat à partir d'un nombre limité de modules créés.

Toujours dans l'objectif de maximiser la diversité des bâtiments à partir d'un nombre limité de modules créés, l'ajout des paramètres « teinte » et « échelle » aux matériaux permet d'obtenir des résultats différents à partir d'un même ensemble de modules et de textures.

Le paramètre « teinte » permet d'ajuster la teinte de la texture du matériau (voir Figure 2.6) tandis que le paramètre « échelle » influence le facteur de mise à l'échelle de la texture. Une fois la génération effectuée, l'artiste peut sélectionner un bâtiment et modifier sa teinte s'il n'est pas satisfait de la génération initiale.

Figure 2.5 Différents matériaux appliqués sur le même ensemble de modules

Ces matériaux paramétrables permettent à l'artiste de travailler de manière itérative et intuitive. En effet, à partir d'un nombre minimal de textures, l'artiste est capable de générer une ville et peut travailler sur une première ébauche de la ville. Il peut ensuite compléter sa librairie de textures interchangeables pour obtenir de plus en plus de diversité et de meilleurs résultats dans la génération.

Figure 2.6 Exemples de variation du paramètre « teinte »

2.2.2 Optimisation des blocs

Lors de la création d'un jeu vidéo, la fluidité visuelle est primordiale. Le nombre d'images par seconde (IPS) que l'engin de jeu doit générer ne doit pas descendre en dessous de 30 ou 60 par exemple, selon les jeux. Pour ne pas arriver en dessous de cette limite, les capacités graphiques sont très importantes et doivent être prises en compte dans le processus de création. Un des facteurs importants qui influence les performances est le nombre de polygones des modèles 3D; il doit être le plus bas possible. Dans le cas de l'outil de génération procédurale de ville, les blocs créés par l'artiste sont dupliqués plusieurs fois. En effet, environ 50 modules servent à constituer un bâtiment, la ville étant composée d'une centaine de bâtiments au minimum, la minimisation du nombre de polygones de chaque module est d'autant plus importante puisque chacun de ceux-ci sera dupliqué plusieurs fois dans l'environnement.

Une des solutions utilisées par l'outil permettant de réduire les coûts de performance est l'utilisation de *levels of details* (LOD). Il s'agit d'une méthode qui remplace un modèle 3D, à mesure que la caméra s'éloigne de celui-ci, par une version similaire comprenant moins de détails, donc moins de polygones. Telle que représentée dans la Figure 2.7, cette méthode

permet de réduire drastiquement le nombre de polygones présents dans la scène et de réduire le temps de calcul lié aux polygones.

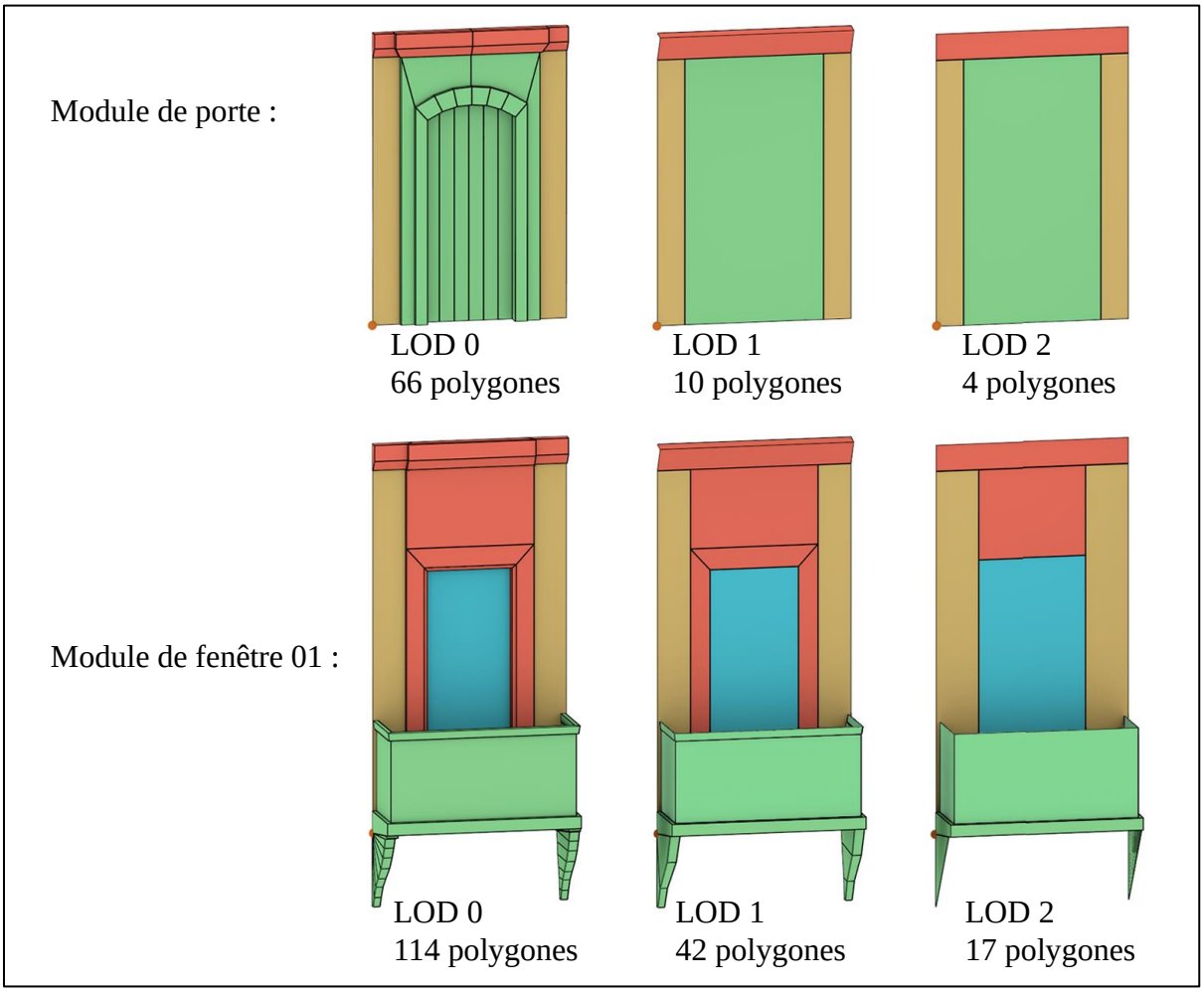


Figure 2.7 Exemples de modules avec plusieurs niveaux de détails

Une deuxième solution utilisée pour optimiser les performances consiste à réutiliser les mêmes modèles 3D en leur appliquant différents matériaux. L'artiste peut donc créer plusieurs variations d'un modèle en optimisant l'espace mémoire nécessaire pour conserver l'information liée à la géométrie (voir Figure 2.5). Avec le même souci d'optimisation de l'espace mémoire, l'artiste peut également réutiliser le même matériel sur plusieurs modèles 3D différents. La Figure 2.8 représente cette situation alors que plusieurs modèles 3D utilisent

les mêmes matériaux génériques (partagés entre les modules), bien que certains modèles utilisent également des matériaux uniques.

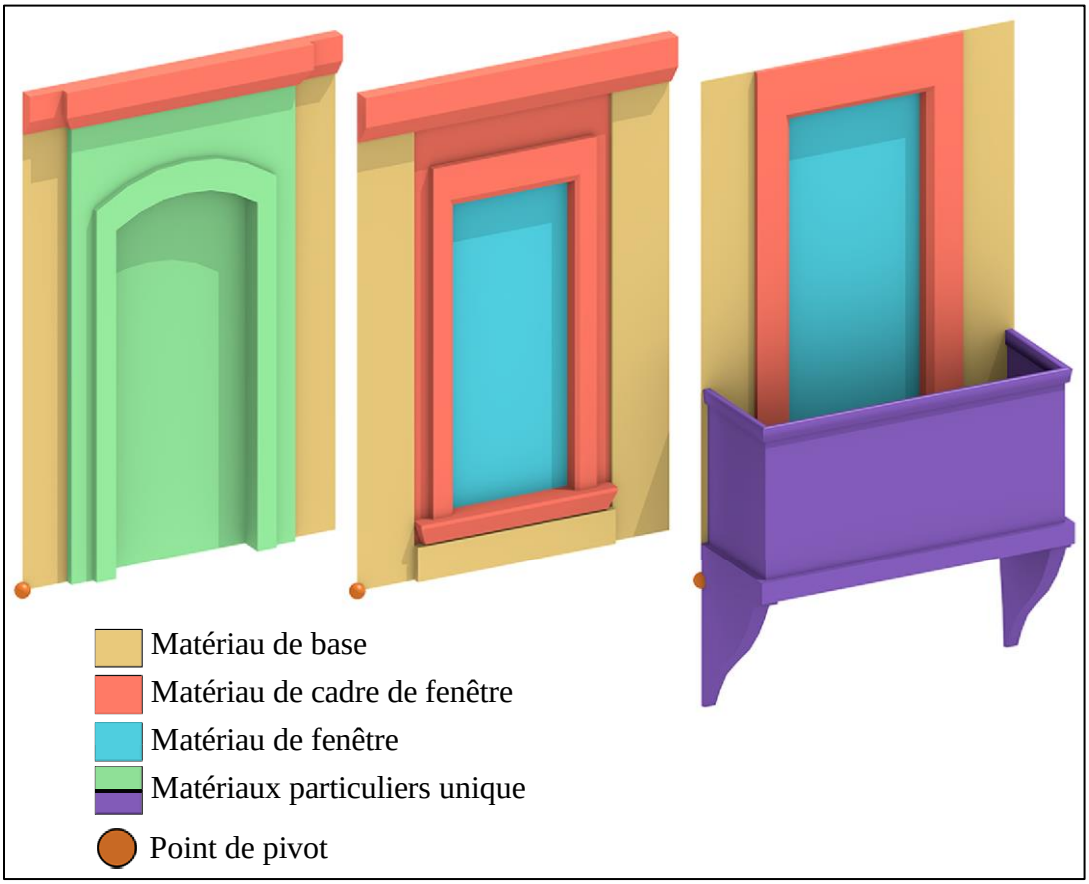


Figure 2.8 Découpage des modules en fonction des matériaux partagés ou uniques

Une fois que l'utilisateur a créé et importé les modules et les textures associées, il peut passer à l'étape suivante du processus de création : la génération du réseau routier grâce au gestionnaire d'intersections.

2.3 Génération des intersections et des rues

Un des objectifs principaux de ce travail de recherche est de laisser le plus de liberté possible à l'utilisateur en termes de création et de génération. Comme il a été dit dans le chapitre portant

sur l'état de l'art, plusieurs générateurs de ville utilisent un plan quadrillé pour représenter la ville. Si l'utilisation d'un plan quadrillé est suffisante pour générer une ville qui sert d'arrière-plan, cette approche n'est pas adéquate lorsque le joueur navigue à l'intérieur de l'environnement. En effet, le joueur remarque rapidement les schémas répétitifs de la génération, ce qui diminue le réalisme de l'environnement virtuel. Pour pallier ce problème, une fonctionnalité spécifique a été développée dans l'outil : le générateur d'intersection qui permet à l'utilisateur de générer des points symbolisant les intersections des rues. L'artiste positionne manuellement les points d'intersection dans l'espace 3D et décide lesquels sont reliés entre eux. Chaque paire de points reliés constitue une rue et grâce à ce principe, l'artiste peut déterminer le nombre de rues convergentes vers un même point d'intersection. La Figure 2.9 présente un exemple de plan de ville complexe qui peut être généré par l'outil proposé.

Les bâtiments sont générés entre les rues formant des « blocs » de bâtiments. La partie délicate de cette fonction est de calculer les points de départ et d'arrivée réels des rangées de bâtiments par rapport à une rue, sachant que la rue a une certaine largeur et que l'angle formé à une intersection entre deux rues influence la position de ces points. En effet, les coordonnées définies par l'utilisateur via le gestionnaire d'intersections indiquent le départ et l'arrivée des rues, en leur centre, mais les bâtiments seront placés à côté des rues, au niveau du trottoir (visible dans la section c de la Figure 2.9).

Dans la section b) de la Figure 2.9, les traits noirs en pointillés représentent les rues formées par les points d'intersection positionnés par l'utilisateur tandis que les traits noirs de part et d'autre des pointillés représentent l'épaisseur réelle de la rue. L'épaisseur des rues est définie par l'utilisateur et elle influence la position des points réels de générations des rangées de bâtiments. Ces points correspondent aux points de départ et d'arrivée de chaque côté d'un bloc de bâtiment (visible dans la section c de la Figure 2.9). On peut voir dans la section d) de cette même figure les valeurs nécessaires pour effectuer le calcul d'un point réel de position de bâtiment.

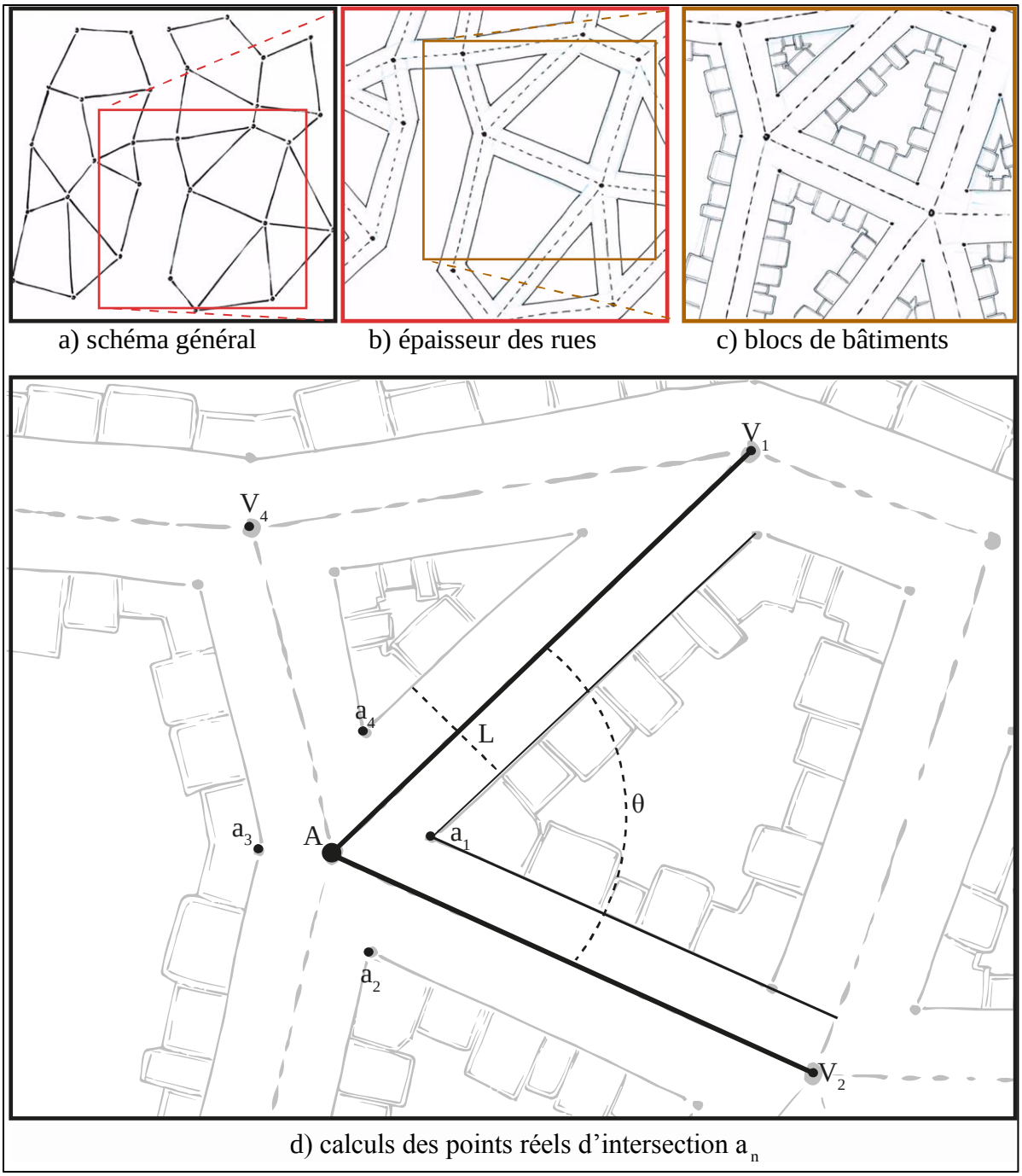


Figure 2.9 Création du réseau routier et calcul des points réels d'intersection

En considérant le point d'intersection « A » placé par l'utilisateur, on peut définir a_n points réels où n est le nombre de rues qui commencent ou se terminent au point A. Ces points réels

correspondent aux points d'intersection des paires de rangées de bâtiments se rejoignant au point A. Si quatre rues convergent au point A, quatre bâtiments angulaires sont disposés aux alentours de celui-ci, entre chaque rue. La partie délicate de ce générateur d'intersections est de calculer ces points réels de départ et d'arrivée des rangées de bâtiments par rapport à une rue, sachant que la largeur des rues et les angles formés aux intersections influencent la position de ces points. La section d) de la Figure 2.9 permet d'aider à la compréhension du calcul ci-dessous.

À partir des informations connues suivantes :

$A=[A_x , A_y]$	Point d'intersection
n	Nombre de rues
$V_{1 \text{ à } n}$	Points d'intersections des rues qui convergent au point A
L	Largeur des rues

Les points réels de départ et d'arrivée des rangées de bâtiments a_n sont définis ainsi :

$$a_n = A + p_n \, \hat{f}_n + L \, \hat{f}_n^{\perp} \tag{3.1}$$

avec

$e_n = \overrightarrow{AV_n}$

$f_n = \overrightarrow{AV_{n+1}}$

$\hat{e}_n$ = Vecteur unitaire de e_n

$\hat{f}_n$ = Vecteur unitaire de f_n

$$p_n = \frac{L}{\tan\left(\frac{\theta_n}{2}\right)} \tag{3.2}$$

où

$$\theta_n = \arccos\left(\frac{e_n \cdot f_n}{||e_n|| \, ||f_n||}\right) \tag{3.3}$$

Lorsque l'utilisateur a créé le réseau routier désiré, que toutes les rues sont connectées entre elles et que le calcul des points réels de position de bâtiments est effectué, le gestionnaire

d'intersection conserve ces données pour que l'outil de génération puisse effectuer la prochaine étape, soit la génération et le placement des bâtiments.

2.4 Outils de génération de bâtiments

La génération de bâtiments est au cœur de l'outil de génération d'environnements urbains. Ce sont les bâtiments qui peuplent les rues et la ville. Grâce aux informations obtenues dans les calculs précédents et aux paramètres fournis par l'utilisateur, l'outil peut générer les bâtiments. Cette section présente et explique le fonctionnement du générateur de bâtiment. Après avoir généré l'ensemble de la ville, l'utilisateur peut ensuite apporter des correctifs ou continuer à habiller l'environnement en créant des bâtiments uniques ou des rangées de bâtiments (voir section 2.5). Il est actuellement possible de générer deux types de bâtiments différents : les bâtiments angulaires et les mitoyens. Les bâtiments mitoyens n'ont qu'une seule façade sur rue tandis que les bâtiments angulaires ont deux façades sur rue et sont situés dans un angle, à l'intersection de deux rues. La section 2.4.2 décrit en détail le fonctionnement et l'utilisation de ces deux types de bâtiments : angulaire et mitoyen.

2.4.1 Paramètres de génération

Afin de fournir à l'utilisateur un maximum de contrôle sur l'apparence esthétique finale de la génération, le système proposé offre à l'artiste la possibilité de spécifier différents paramètres. En plus de la hauteur et de la largeur du bâtiment, l'utilisateur peut définir sa profondeur, la position de la porte par rapport au bâtiment et la valeur de l'angle formé par les deux façades pour les bâtiments angulaires. La liste complète des paramètres utilisés lors de la génération est présentée dans le tableau 2.1. Si certains paramètres sont définis par une valeur précise (booléenne, numériques, etc.), d'autres sont plutôt choisis aléatoirement à l'intérieur d'un intervalle de valeur (bornes inférieure et supérieure). Ces intervalles de valeur permettent d'obtenir plusieurs variations de génération à partir d'un même ensemble de paramètres (voir Figure 2.10).

L'utilisateur peut définir des ensembles de règles qui précisent l'allure finale des bâtiments. Présenté plus en détail dans la section 2.6, l'ensemble de règles regroupe la totalité des règles et paramètres nécessaires à la génération d'un style de bâtiment. L'artiste peut ainsi définir plusieurs styles de générations, assigner des valeurs différentes aux paramètres de chaque style et de ce fait générer une plus grande variété de bâtiments.

Tableau 2.1 Liste des paramètres nécessaires pour l'outil de génération

Sous-section	Nom	Description	Type de paramètre
Interface	Hauteur	Nombre d'étages	Intervalle
	Largeur	Nombre de modules	Intervalle
	Profondeur	Nombre de modules	Intervalle
	Position Porte	Emplacement de la porte par rapport au bâtiment	Nombre
	Mur gauche	Présence ou non d'un mur gauche	Booléen
	Mur droit	Présence ou non d'un mur droit	Booléen
	Angle	Angle formé par les deux façades	Nombre (degré)
	ID	Identifiant de l'ensemble de règles à utiliser pour la génération	Nombre
Composants	Porte	Module de porte	Module
	Fenêtres	Modules de fenêtres	Liste de modules
	Murs latéraux	Modules de murs latéraux	Liste de modules
	Vitrines	Modules de vitrines	Liste de modules
	Poutre	Module de poutre (situé aux extrémités gauche et droite des façades)	Module
	Toit	Module de toit	Module
	Matériau	Matériau « de base »	Matériau
	Teinte	Teinte assignée à la texture	Couleur RVB
	Échelle	Échelle de la texture	Nombre

a) Largeur : 3	b) Largeur : 6	c) Largeur : 3	d) Largeur : 5
Hauteur : 3	Hauteur : 5	Hauteur : 5	Hauteur : 4
Profondeur : 3	Profondeur : 1	Profondeur : 2	Profondeur : 3

Figure 2.10 Différentes générations suivant le même ensemble de règles

2.4.2 Bâtiments angulaires et mitoyens

Dans l'optique de générer une ville, le système proposé fait la distinction entre deux types de bâtiments : mitoyen et angulaire. Un bâtiment mitoyen est constitué d'une seule façade sur rue et de deux murs latéraux tandis qu'un bâtiment angulaire possède deux façades sur rue. Ce dernier est placé à l'intersection de deux rues et ses façades forment le même angle que celui formé par les rues en question (voir Figure 2.11). Puisque l'utilisateur de l'outil peut former le schéma de rues qu'il souhaite, les angles formés par l'intersection des rues ne sont pas limités à des valeurs telles que 90° ou 180°. Ils peuvent avoir n'importe quelle valeur comprise entre 0° et 360°.

Pour des soucis d'esthétisme, le système place un biseau entre les deux façades principales d'un bâtiment angulaire, ce qui permet aussi de limiter les incohérences lorsque l'angle entre les deux façades est très faible. Un exemple populaire de bâtiment angulaire avec un angle

faible est celui du *Flatiron building* dans la ville de New York. La Figure 2.12 présente un schéma de la vue de dessus du bâtiment. Les deux façades du bâtiment forment un angle de 23° et il devient difficile pour le générateur de bâtiment de créer des modules cohérents dans la pointe du bâtiment (identifiée en rouge dans la Figure 2.12). Le biseau permet de « casser » cet angle et d'apporter une notion d'épaisseur minimale ajoutant du réalisme au bâtiment.

a) Bâtiment angulaire b) Bâtiment mitoyen

Figure 2.11 Exemple de bâtiments angulaire et mitoyen

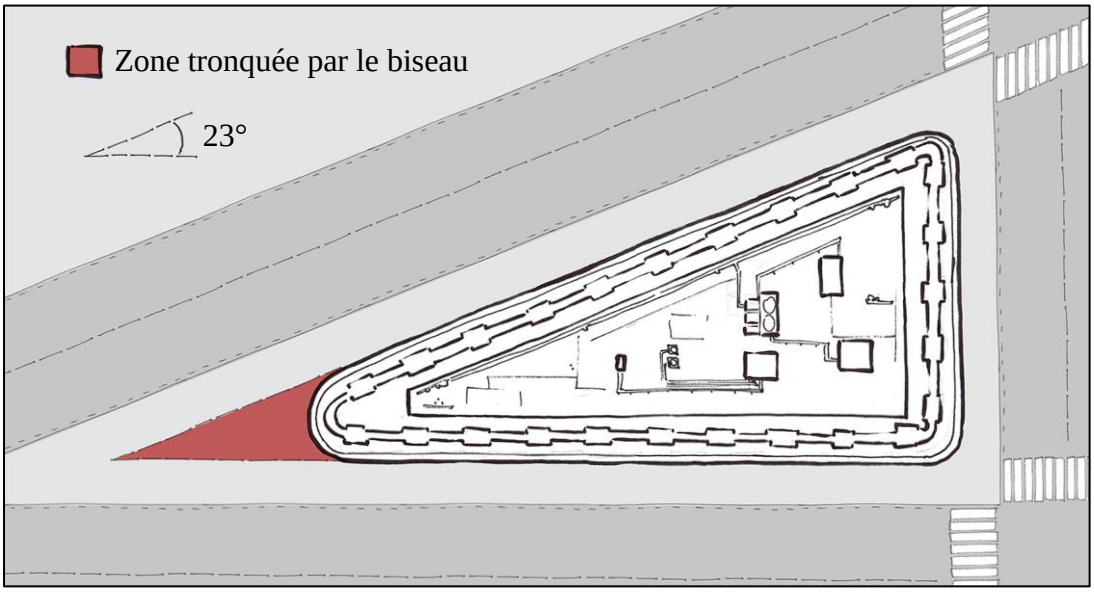


Figure 2.12 Schéma du *Flatiron Building*

2.5 Génération indépendante

Pour permettre à l'artiste de générer l'environnement urbain désiré, l'outil de génération de ville n'est pas la seule solution proposée. Cet outil est accompagné de deux systèmes de génération indépendante : la génération unique et la génération en rangée. L'utilisateur a la possibilité, en amont de la génération ou lorsque celle-ci a été effectuée, de générer ces bâtiments de manière autonome pour répondre à des attentes particulières. Si les plans de ville établis par la direction artistique exigent qu'un bâtiment soit isolé, entouré par un stationnement ou un espace vert par exemple, l'utilisateur a la possibilité de générer un bâtiment grâce à l'option de génération unique. Disposant des mêmes paramètres que les bâtiments générés par l'outil de génération globale, ce bâtiment peut être modifié et ajusté pour qu'il réponde parfaitement aux directives artistiques. Dans un autre exemple, si l'utilisateur a besoin de générer une série de bâtiments, sans que ceux-ci fassent partie d'un bloc fermé de bâtiments, entouré de rues et délimité par leurs intersections, la génération indépendante en rangée lui permet d'obtenir ce résultat. Dans ce cas, l'utilisateur dispose d'un microgénérateur (visible à gauche des bâtiments dans la section de droite de la Figure 2.13) qui place un nouveau bâtiment à côté du précédent autant de fois qu'il le souhaite.

Figure 2.13 Génération unique et génération en rangée

2.6 Création d'un ensemble de règles

Introduit précédemment, l'ensemble de règles est un des éléments clés pour rendre possible la génération de bâtiments. Son utilisation permet à l'artiste de définir la valeur des paramètres que l'outil utilise pour la génération. Un gestionnaire d'ensemble de règles a été mis en place pour permettre à l'utilisateur d'enregistrer plusieurs ensembles de règles à une même place, de manière à pouvoir choisir lequel assigner au bâtiment lors de sa génération. Grâce à ce gestionnaire, l'artiste peut facilement apporter des modifications aux ensembles de règles, les dupliquer pour en faire des variantes et en créer des nouveaux, se constituant ainsi une librairie d'ensemble de règles. Au fil de l'utilisation de l'outil, l'artiste a donc la possibilité de réutiliser un ensemble de règles établi pour une seconde génération, réduisant ainsi le temps nécessaire pour générer une ville, ou constituer un autre ensemble de règles permettant de diversifier l'aspect esthétique de la ville générée.

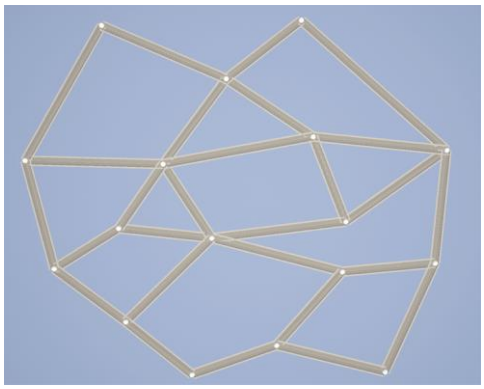
Lorsque l'artiste a terminé de configurer au minimum un ensemble de règles (et que toutes les étapes présentées précédemment ont été complétées), l'outil est prêt à générer la ville. L'utilisateur n'a plus qu'à cliquer sur le bouton d'action dans le gestionnaire de ville.

2.7 Résultats

Pour permettre à l'artiste de se concentrer sur la partie créative de son travail en automatisant les parties rébarbatives, un des critères principaux de ce travail de maîtrise est d'effectuer les différentes tâches dans un temps minimum. Plusieurs facteurs sont à prendre en compte pour déterminer le temps nécessaire pour obtenir le résultat final de la génération. Tout d'abord, le facteur le plus conséquent, le temps nécessaire à l'artiste pour réaliser l'ensemble de modules et de la banque de matériaux interchangeables qui les accompagne. Bien sûr, ce temps varie selon les artistes et est différent pour chaque module réalisé. Le deuxième facteur de temps à prendre en compte est le temps que l'utilisateur prend pour l'établissement du plan de ville dans l'outil. Vient ensuite le temps accordé à la configuration d'un ou plusieurs ensembles de

règles. Ces deux dernières périodes durent quelques minutes en général. Le facteur de temps suivant est le temps utilisé par l'outil pour générer l'ensemble des bâtiments et les positionner de manière cohérente autour des rues. Ce temps varie selon le nombre de bâtiments, influencé par le nombre et la longueur totale des rues. Le dernier facteur de temps à prendre en compte pour déterminer la durée nécessaire d'une génération d'environnement urbain est le temps passé par l'utilisateur pour modifier, ajuster et personnaliser le résultat de la génération. En effet, lorsque la génération est terminée, l'artiste a la possibilité de modifier l'aspect visuel des bâtiments en accédant à leurs paramètres (voir la liste complète des paramètres dans le tableau 2.1). Grâce à l'interface de l'engin de jeu *Unreal Engine 4*, l'artiste peut appliquer des modifications à plusieurs bâtiments de manière simultanée. En effet, si l'artiste sélectionne plusieurs éléments de la scène 3D dans l'engin de jeu, les paramètres qu'ils ont en commun apparaissent dans l'onglet « détails ». Si l'artiste sélectionne plusieurs bâtiments, les paramètres tels que les dimensions, le matériau ou la présence de vitrine sont accessibles. Il peut donc appliquer des modifications à une large sélection de bâtiments. Cet aspect est un point important car un travail itératif permet d'offrir un gain de temps considérable. L'ajustement rapide des paramètres permet à l'artiste et aux designers de développer le jeu avec un environnement urbain rapide à générer, à modifier et à corriger.

Les figures 2.14, 2.15 et 2.16 présentent respectivement le plan de la ville établi par l'artiste, une vue générale de la génération, des exemples visuels de résultats et des informations des trois environnements urbains créés à l'aide de l'outil proposé. Les temps nécessaires à l'élaboration des différentes étapes de la génération y sont également détaillés. Le tableau 2.2 regroupe ces différents temps et permet de mieux les comparer. Bien sûr, les temps varient en fonction des performances de l'ordinateur utilisé. Les résultats présentés ont été obtenus sur un ordinateur basé sur Windows 7 64-bits, ayant 32 gigabits de mémoire vive (RAM), un processeur Intel® Core™ i7-6700K à 4,00GHz ainsi qu'une carte graphique NVidia GeForce GTX 1070.



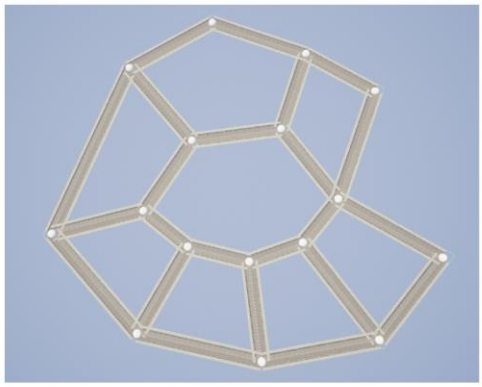
a) réseau routier généré

b) vue générale de la génération

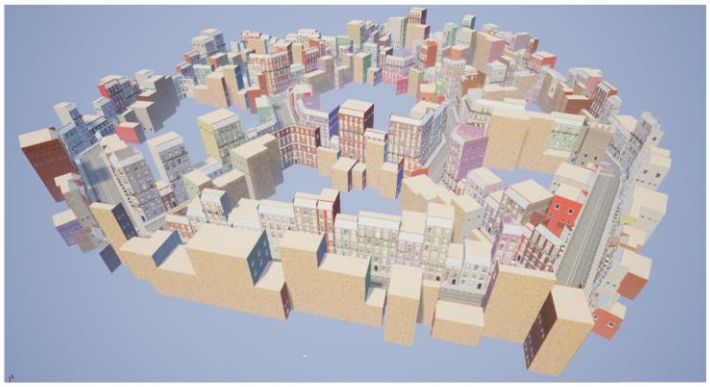
c) Sélection de rendus de la ville 01

Informations	**Durée**	(HH : MM : SS)
Ville 01	Création des modules :	~ 30 : 00 : 00
362 bâtiments	Création du réseau routier :	00 : 04 : 46
26 rues	Création de(s) ensemble(s) de règles :	00 : 01 : 41
	génération des bâtiments :	00 : 00 : 16
	Corrections et ajustements :	00 : 12 : 29
	Temps total :	~ 30 : 21 : 32

Figure 2.14 Résultats de la génération de la ville 01



a) réseau routier généré

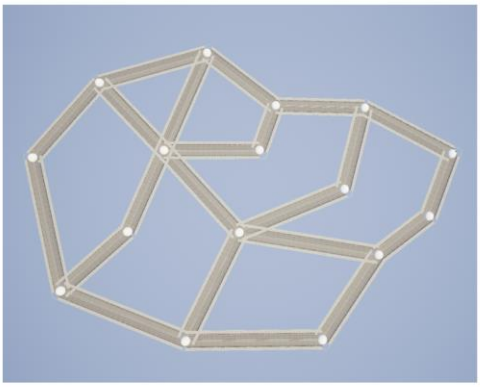


b) vue générale de la génération

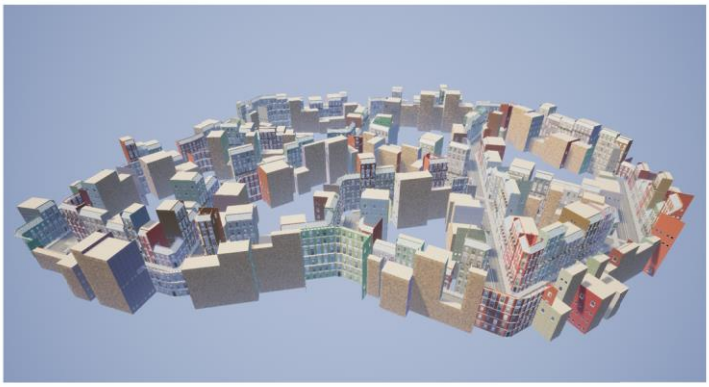
c) Sélection de rendus de la ville 02

Informations	**Durée**	(HH : MM : SS)
Ville 02	Création des modules :	00 : 00 : 00
290 bâtiments	Création du réseau routier :	00 : 03 : 52
23 rues	Création de(s) ensemble(s) de règles :	00 : 01 : 25
	génération des bâtiments :	00 : 00 : 05
	Corrections et ajustements :	00 : 10 : 37
	Temps total :	00 : 15 : 59

Figure 2.15 Résultats de la génération de la ville 02



a) réseau routier généré



b) vue générale de la génération

c) Sélection de rendus de la ville 03

Informations	**Durée**	(HH : MM : SS)
Ville 03	Création des modules :	00 : 00 : 00
251 bâtiments	Création du réseau routier :	00 : 04 : 05
22 rues	Création de(s) ensemble(s) de règles :	00 : 00 : 00
	génération des bâtiments :	00 : 00 : 17
	Corrections et ajustements :	00 : 11 : 08
	Temps total :	00 : 15 : 30

Figure 2.16 Résultats de la génération de la ville 03

Tableau 2.2 Détails des temps nécessaires pour les différentes étapes de la génération

	Modules (hh:mm:ss)	Rues (mm:ss)	Règles (mm:ss)	Génération (mm:ss)	Modifications (mm:ss)	Total (hh:mm:ss)
Ville 01	~30 :00 :00	04 :46	01 :41	00 :16	12 :29	~30:21:32
Ville 02	00 :00 :00	03 :52	01 :25	00 :05	10 :37	00:15:59
Ville 03	00 :00 :00	04 :05	00 :00	00 :17	14 :08	00:15:30

Au fur et à mesure que l'utilisateur génère des villes, il a la possibilité de réutiliser des éléments, comme les modules 3D ou les ensembles de règles définis. On peut donc observer une différence de temps énorme entre les villes 01 et 02, du fait de conserver le même ensemble de modules.

2.7.1 Temps de développement et d'entretien de l'outil

Présentés précédemment, les temps nécessaires pour obtenir des résultats avec l'outil de génération procédurale proposé dans ce mémoire sont très courts. Il en est de même avec le temps demandé lors du passage de l'outil d'une version du logiciel Unreal Engine à une autre. Dans le cadre de cette maîtrise, l'outil a d'abord été développé dans la version 4.13.2 et a ensuite été exporté vers la version 4.17.2. La transition d'une version à l'autre n'a pas demandé d'étapes particulières, si ce n'est la première ouverture du logiciel dans sa nouvelle version, durant laquelle celui-ci recompile l'entièreté des shaders et les différents scripts. Passé cette étape de chargement, l'outil est prêt à être utilisé de la même manière que dans sa version précédente, sans que l'utilisateur n'observe une quelconque différence.

En ce qui concerne le temps pris pour développer l'outil lui-même, cela s'est étalé sur une période d'un an et demi, durant laquelle plusieurs itérations de l'outil ont été faites, certaines approches ayant été envisagées mais n'ayant pas abouties. Durant cette période, l'outil a été

développé simultanément avec la rédaction de ce mémoire ainsi que la création des ensembles de modules et de textures.

2.7.2 Avantages

Le système de génération semi-automatique d'environnements urbains présenté dans ce mémoire offre plusieurs avantages :

- Le déroulement de la génération proposé par l'outil permet à l'artiste d'obtenir un résultat de génération rapidement, notamment grâce à l'automatisation des étapes non créatives et à la possibilité de réutiliser certains composants tels que les ensembles de règles ou les modules.
- L'artiste a la possibilité d'ajuster le résultat obtenu de manière itérative grâce au principe de fonctionnement de l'outil. Il peut également améliorer ou compléter ce résultat à l'aide des autres outils fournis (génération unique, génération en rangée et modifications de plusieurs bâtiments en simultané).
- Contrairement à des travaux comme celui de Greuter *et al*. (2003) qui ne permet pas d'avoir des schémas de rues irréguliers, le générateur d'intersections offre à l'artiste une liberté d'expression qui lui permet de concevoir des environnements basés sur des schémas de rues irréguliers.
- L'utilisation de modules externes créés par un artiste permet d'obtenir des bâtiments dont la qualité visuelle et l'aspect esthétique répondent à la direction artistique prise lors du développement du jeu. Même si quelques contraintes sont présentes, leur faible nombre permet à l'outil de s'adapter à plusieurs styles architecturaux.
- La gestion des matériaux paramétrables par l'outil permet elle aussi d'augmenter le nombre de variations possibles dans la génération de bâtiments, notamment par la variation de la teinte ou de l'échelle des textures. De plus, un modèle-type de matériau paramétrable permet à l'artiste de créer une très grande multitude de matériaux et de les intégrer à l'outil.

- Le gestionnaire d'ensemble de règles permet à l'utilisateur de définir en amont de la génération une librairie d'ensemble de règles que les bâtiments vont devoir suivre lors de leur création. Une simple modification à un ensemble de règles et tous les bâtiments qui suivaient cettedite règle sont affectés par cet ajustement. Ce point permet lui aussi d'obtenir un travail itératif, en permettant à l'artiste d'effectuer une modification rapide à une grande partie de la génération.

2.7.3 Limitations de l'outil

Pour ce qui est des inconvénients liés à cet outil, ils sont liés aux contraintes imposées à l'artiste en amont de la génération ainsi qu'aux types de bâtiments pouvant être générés. En effet, l'outil ne permet de générer que deux variantes de bâtiments, les mitoyens et les bâtiments angulaires. À travers cet outil, l'artiste n'a donc pas la possibilité de générer de manière automatique des bâtiments aux formes complexes, tels qu'un bâtiment sur pilotis ou un bâtiment en deux parties distinctes superposées. À ce stade du développement, l'outil ne permet pas à l'artiste de choisir la largeur des modules de façade. L'outil n'offre pas non plus à l'artiste la possibilité de choisir différents modules de fenêtres pour un même étage. Présentement, un module unique est assigné à toute la largeur de l'étage. L'artiste peut changer de module et en assigner un autre, mais il ne peut pas appliquer plusieurs modules à un même étage, pour créer une colonne centrale démarquée par exemple. Concernant la génération du réseau routier, l'outil proposé ne permet pas à l'utilisateur de générer un réseau ayant des rues en pente pour créer un plan de ville sur un flanc de montagne par exemple. De plus, l'outil fonctionne correctement seulement si les rues forment des blocs fermés. Une rue se terminant en impasse empêche le gestionnaire d'intersection d'identifier les ensembles de bâtiments, ce qui perturbe le bon fonctionnement de l'outil. Toujours concernant les intersections, l'outil ne gère pas les angles que forment les modules de rues au niveau du trottoir. On remarque une superposition des modules de rues aux angles aigus ainsi qu'un vide entre deux trottoirs aux angles obtus. L'utilisation d'un module de rue avec un squelette d'animation permettrait de positionner chaque coin de rue aux points

réels d'intersection calculés dans la section 2.3, ce qui résoudrait ce problème de superposition, mais l'intégration de cette fonctionnalité n'a pas été un succès.

Malgré l'utilisation de LOD, de modules réutilisés et de matériaux paramétrables, les performances graphiques de la scène 3D sont vraiment impactées par la génération de la zone urbaine. En effet, le nombre d'images par seconde affichées par l'engin de jeu est réduit drastiquement dès que le nombre de bâtiments présent dans la scène augmente. Cela est dû à l'utilisation d'un grand nombre de modules pour chaque bâtiment. Pour une zone urbaine conséquente, ce nombre peut atteindre des milliers. De plus, le moteur de rendu de l'engin calcule un « *drawcall*[26] » pour chaque matériau présent sur un objet. C'est principalement ce nombre élevé de *drawcalls* qui consomme les performances graphiques et qui empêche un affichage fluide de la scène. Une solution proposée par l'engin *Unreal Engine 4* est de fusionner entre eux les différents modules constituant le bâtiment. Grâce à cette action, le nombre de *drawcalls* est réduit considérablement et permet d'obtenir une meilleure fluidité. Le problème avec cette solution est que l'utilisateur doit générer manuellement cette fusion, pour chaque bâtiment. C'est aux antipodes de l'objectif de cette maîtrise d'automatiser le travail de l'artiste.

Voici quelques améliorations qu'il serait intéressant d'apporter au système :

1. Le réseau routier pourrait être généré à partir d'une image 2D fournie par l'artiste d'environnement. Cette méthode permettrait d'apporter plusieurs avantages. La création de ladite texture, extérieure à l'engin de jeu, ne demande pas à l'artiste de comprendre le fonctionnement du gestionnaire d'intersections. Dans un contexte réel de production de jeu, l'artiste maîtrise probablement l'utilisation d'un logiciel d'édition d'image. Cette fonctionnalité permettrait également d'utiliser des données géographiques réelles. En plus de générer le réseau routier, cette image 2D pourrait stocker des données (par un code de couleurs) telles que la largeur des rues ou la

[26] Un « *drawcall* » est le calcul graphique que fait le moteur de jeu pour afficher un objet à l'écran.

concentration démographique (permettant de générer différents types de bâtiments en fonction de cette information).

2. La création procédurale de rues « secondaires » permettrait à l'artiste d'environnement de ne créer que les rues principales telles que les rues définies par le designer de niveau et d'automatiser la génération des rues périphériques.
3. Concernant les matériaux appliqués aux bâtiments, l'ajout de paramètres tels qu'une variable de détérioration ou l'ajout de graffitis offrirait une diversité de génération supplémentaire à l'outil.
4. Offrir à l'artiste un paramètre de largeur de rue lui permettant de définir séparément la largeur de chaque rue. Cela permettrait de différencier les ruelles et rues piétonnes des artères principales.

Bien que les résultats puissent permettre à eux seuls de confirmer la validité de certains objectifs visés, tels que l'obtention d'une qualité visuelle suffisante, ou encore la rapidité d'exécution de l'outil, d'autres objectifs n'ont pas été validé scientifiquement. C'est par exemple le cas pour l'aspect itératif de l'outil. Pour pouvoir affirmer que l'outil permet d'offrir un travail plus itératif, il faudrait choisir un échantillon de spécialiste de l'industrie du jeu vidéo et leur faire essayer l'outil. À travers cette méthodologie, ces utilisateurs experts pourraient à la suite de leur essai remplir un questionnaire dans lequel ils donneraient leurs impressions sur l'outil, sur la rapidité et la facilité avec lesquelles ils auront utilisé l'outil ainsi que leur avis sur l'aspect itératif de l'outil proposé. Les résultats obtenus grâce à ce questionnaire permettraient d'établir scientifiquement la validité de l'objectif itératif. D'autres objectifs pourraient être validés ou infirmés scientifiquement en suivant une méthodologie similaire, tels que la complexité supposée du réseau routier ou encore l'utilisation intuitive de l'outil par l'artiste.

CONCLUSION

Le coût et le temps de production d'un jeu vidéo ont beaucoup augmenté depuis plusieurs années. De quelques programmeurs pouvant créer un jeu entier en quelques mois, la création d'un jeu peut demander plusieurs centaines de personnes pendant plusieurs années. Toujours plus grands et immersifs, les environnements virtuels sont un des éléments les plus importants dans un jeu vidéo. Leur création dans le cadre d'une production de jeu vidéo est coûteuse et chronophage. Bien qu'il y ait une grande partie créative dans la conception d'un environnement, d'autres parties telles que le placement et la duplication d'éléments ne le sont pas. Ces parties sont même rébarbatives et pourraient être automatisées.

Les villes et autres environnements urbains sont fréquemment présents dans les jeux vidéo de nos jours. Tels que vus dans la revue de littérature, plusieurs travaux actuels permettent de créer procéduralement des environnements urbains, mais ils possèdent des limitations majeures réduisant leur utilisation dans un contexte réel de production. D'un côté, certains outils sont capables de générer un environnement vaste accompagné d'un réseau routier complexe, mais qui est peuplé de bâtiments trop simplifiés dans leur forme, souvent sans matériaux ou avec une répétition trop apparente entre eux. D'un autre côté, des générateurs permettent aux utilisateurs de générer des bâtiments complexes et variés, répondant aux attentes de la direction artistique prise pour le jeu, mais n'offrent pas à l'artiste la possibilité de générer une zone urbaine globale. L'artiste doit générer les bâtiments un à un à l'aide de l'outil, puis les positionner dans l'espace 3D manuellement afin d'obtenir l'environnement urbain désiré.

Les limitations apportées par les travaux actuels ont permis de définir les objectifs de ce mémoire de la façon suivante :

- Une génération de bâtiments paramétrables répondant à une qualité visuelle attendue par la direction artistique.

- La création d'un réseau routier complexe.
- Le positionnement automatique des bâtiments générés par rapport au réseau routier créé en amont.
- L'aspect itératif de la création et de la génération qui permet à l'artiste de travailler de manière optimale dans un cadre de production de jeu. L'utilisateur doit pouvoir modifier, ajuster ou supprimer facilement et rapidement un élément généré, sans avoir à retravailler sur le reste du contenu généré.
- Prenant place dans un contexte de production de jeu vidéo, cet outil doit pouvoir être compris rapidement et utilisé de manière intuitive par les artistes d'environnement.

Présenté dans le chapitre 2, l'outil de génération procédurale d'environnement urbain proposé dans ce mémoire est séparé en plusieurs parties. Dans un premier temps, l'artiste doit créer un ensemble de modules dans le logiciel de création 3D de son choix. Ces modules constitueront les bâtiments. Il doit également créer les textures accompagnant ces modules avant de les importer dans l'engin de jeu. Dans un deuxième temps, il définit un ou plusieurs ensembles de règles. Ces ensembles définissent les paramètres que vont suivre les bâtiments lors de leur génération. L'artiste génère également le réseau routier de manière semi-automatique grâce à l'outil et à l'interface de l'engin de jeu. Lorsque l'artiste est satisfait du réseau routier généré, que les modules et les textures sont importés et que les matériaux sont configurés, l'artiste lance la génération. En quelques secondes, la zone urbaine définie est peuplée de bâtiments. Si l'artiste le souhaite, il peut dans un dernier temps ajouter des bâtiments grâce aux générateurs indépendants (unique ou en rangée) et/ou modifier un ou plusieurs bâtiments à l'aide de l'outil. Il peut également interférer avec le matériau assigné, ajuster l'échelle ou la teinte de la texture, mais aussi le remplacer entièrement par un autre matériau de son choix. Ces réglages se font de manière instantanée, ce qui permet à l'outil d'offrir un travail itératif et intuitif à l'artiste.

En conclusion, le système qui a été développé dans ce mémoire fournit un nouvel outil de création semi-automatique d'environnement urbain. Il permet de générer un environnement vaste avec un réseau routier complexe, tout en proposant des bâtiments variés, répondant aux

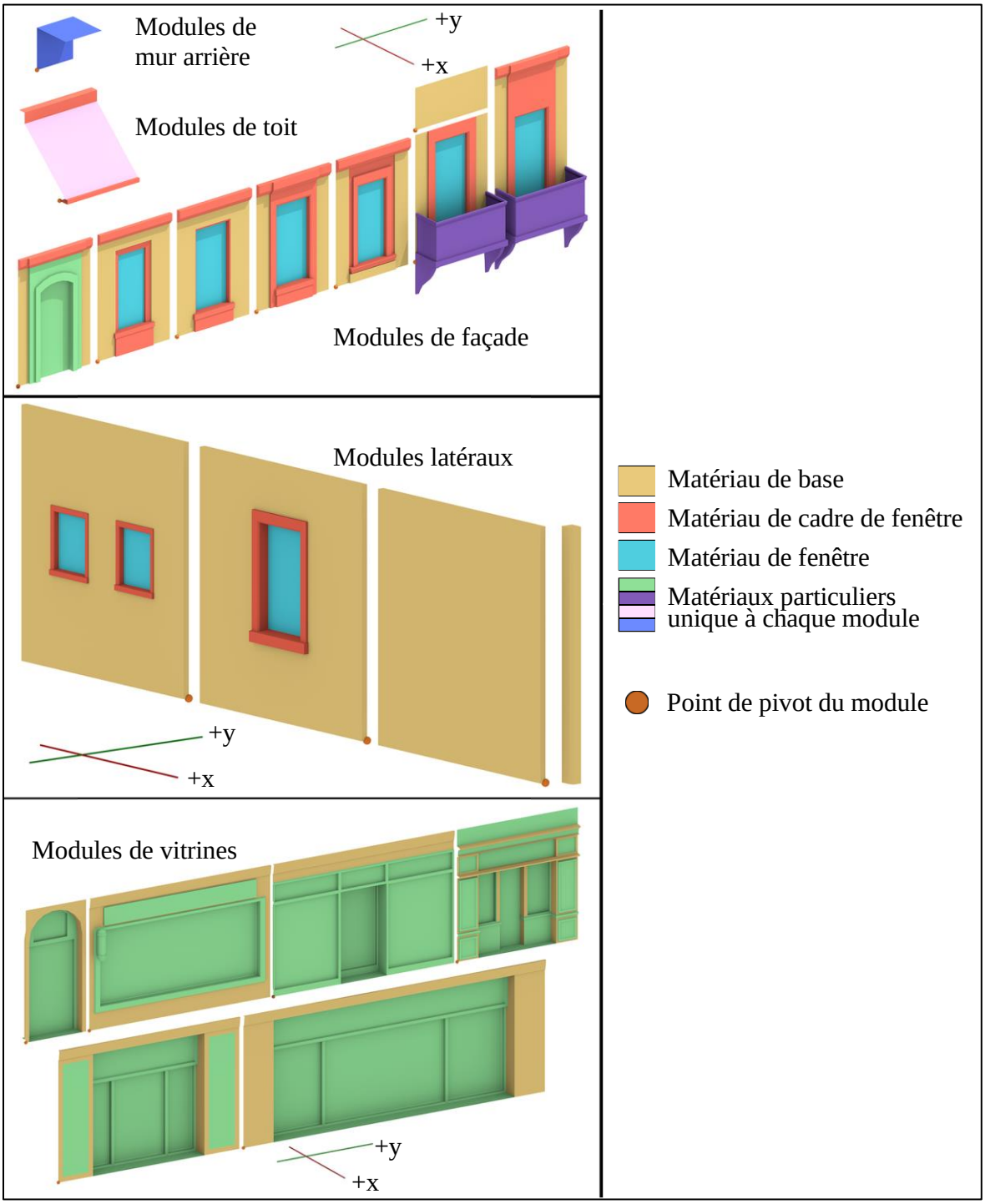
attentes de la direction artistique. L'outil est intuitif, offre un travail itératif et génère un environnement de qualité en l'espace de quelques secondes.

Dans de futures recherches, il serait intéressant de se pencher sur un outil encore plus intuitif, offrant à l'utilisateur une plus grande liberté d'expression tout en offrant un meilleur contrôle sur le résultat obtenu par la génération. Cet outil pourrait offrir des fonctionnalités telles que :

- La génération d'ensemble de règles à partir d'images fournies par l'artiste.
- La possibilité pour l'artiste de spécifier des types de quartiers (résidentiel, commercial industriel, etc.) qui viendrait influencer le type de bâtiments générés.
- L'utilisation d'un menu dynamique (tel un clic droit ou une roue dynamique) permettant d'accéder directement aux différents paramètres de l'élément sélectionné.
- L'ajout de variables personnalisées permettant d'adapter l'outil à une exigence particulière du cadre de production. Ces variables pourraient être une valeur de détérioration, une texture utilisée pour les barrières de balcons ou encore le nombre de portes que contient un bâtiment.

ANNEXE 1

Ensemble des modules créés



LISTE DE RÉFÉRENCES BIBLIOGRAPHIQUES

Abela, R., Liapis, A., & Yannakakis, G. N. (2015). A Constructive Approach for the Generation of Underwater Environments. Dans *FDG workshop on Procedural Content Generation in Games*.

Barret, L., Vance, C., & Youngblood, G. M. (2011). *Lessons in user interface design in the procedural city generation for games tool & Ürban PAD* présentée à Proceedings of the 2nd International Workshop on Procedural Content Generation in Games, Bordeaux, France.

Becher, M., Krone, M., Reina, G., & Ertl, T. (2017). *Feature-based volumetric terrain generation* présentée à Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, San Francisco, California.

Dormans, J. (2012). *Generating Emergent Physics for Action-Adventure Games* présentée à Proceedings of the The third workshop on Procedural Content Generation in Games, Raleigh, NC, USA.

Fernandez-Vara, C., & Thomson, A. (2012). *Procedural Generation of Narrative Puzzles in Adventure Games: The Puzzle-Dice System* présentée à Proceedings of the The third workshop on Procedural Content Generation in Games, Raleigh, NC, USA.

Garg, A., & Maxwell, K. (2010). *Seamless fracture in a production pipeline* présentée à ACM SIGGRAPH 2010 Talks, Los Angeles, California.

Génevaux, J.-D., Gurin, E., Peytavie, A., & Benes, B. (2013). Terrain generation using procedural models based on hydrology. *ACM Trans. Graph., 32*(4), 1-13.

Golding, J. (2010). Building Blocks: Artist Driven Procedural Buildings. *GDC Vault*. Repéré à http://gdcvault.com/play/1012655/Building-Blocks-Artist-Driven-Procedural

Gonzalez-Ochoa, C., Holder, D., & Cook, E. (2012). *From a calm puddle to a stormy ocean: rendering water in Uncharted* présentée à ACM SIGGRAPH 2012 Talks, Los Angeles, California.

Greuter, S., Parker, J., Stewart, N., & Leach, G. (2003). *Real-time procedural generation of `pseudo infinite' cities* présentée à Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia, Melbourne, Australia.

Howard, C., & Lemus, C. (2012). *Asking the impossible on SSX: creating 300 tracks on a ten track budget* présentée à ACM SIGGRAPH 2012 Talks, Los Angeles, California.

Jesus, D., Coelho, A., & Sousa, A. A. (2015). *Towards interactive procedural modelling of buildings* présentée à Proceedings of the 31st Spring Conference on Computer Graphics, Smolenice, Slovakia.

Kamal, K. R., & Uddin, Y. S. (2007). *Parametrically controlled terrain generation* présentée à Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia, Perth, Australia.

Keim, H., Simmons, M., Teece, D., Reisweber, J., & Drakeley, S. (2016). *Art-directable procedural vegetation in disney's zootopia* présentée à ACM SIGGRAPH 2016 Talks, Anaheim, California.

Keith, C. (2010). *Agile game development with Scrum*. Upper Saddle River, NJ: Addison-Wesley.

Kelly, G., & McCabe, H. (2007). *Interactive city generation methods* présentée à ACM SIGGRAPH 2007 posters, San Diego, California.

Khaled, R., Nelson, M. J., & Barr, P. (2013). *Design metaphors for procedural content generation in games* présentée à Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Paris, France.

Kruse, J., Sosa, R., & Connor, A. M. (2016). *Procedural urban environments for FPS games* présentée à Proceedings of the Australasian Computer Science Week Multiconference, Canberra, Australia.

Larive, M., & Gaildrat, V. (2006). *Wall grammar for building generation* présentée à Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia, Kuala Lumpur, Malaysia.

Larsson, M., Zalzala, J., & Duda, G. (2006). *A procedural ocean toolkit* présentée à ACM SIGGRAPH 2006 Sketches, Boston, Massachusetts.

Lee, L., & Pavlov, N. (2008). *Procedural fracturing and debris generation for Kung-Fu Panda* présentée à ACM SIGGRAPH 2008 talks, Los Angeles, California.

Lipp, M., Wonka, P., & Wimmer, M. (2008). *Interactive visual editing of grammars for procedural architecture* présentée à ACM SIGGRAPH 2008 papers, Los Angeles, California.

Longay, S., Runions, A., Boudon, F., & Prusinkiewicz, P. (2012). *TreeSketch: interactive procedural modeling of trees on a tablet* présentée à Proceedings of the International Symposium on Sketch-Based Interfaces and Modeling, Annecy, France.

Moore, G. E. (1965). Cramming More Components Onto Integrated Circuits Dans *Electronics* (Vol. 38). New York: McGraw-Hill.

Müller, P. (2006). *Procedural modeling of cities* présentée à ACM SIGGRAPH 2006 Courses, Boston, Massachusetts.

Müller, P., Wonka, P., Haegler, S., Ulmer, A., & Gool, L. V. (2006). *Procedural modeling of buildings* présentée à ACM SIGGRAPH 2006 Papers, Boston, Massachusetts.

Ong, T. J., Saunders, R., Keyser, J., & Leggett, J. J. (2005). *Terrain generation using genetic algorithms* présentée à Proceedings of the 7th annual conference on Genetic and evolutionary computation, Washington DC, USA.

Oravakangas, L. (2015). *Game Environment Cration : Efficient and Optimized Working Methods* (University of Applied Sciences - Kajaanin Ammattikorkeakoulu).

Pirk, r., Niese, T., Deussen, O., & Neubert, B. (2012). Capturing and animating the morphogenesis of polygonal tree models. *ACM Trans. Graph., 31*(6), 1-10.

Raffe, W. L., Zambetta, F., & Li, X. (2011). *Evolving patch-based terrains for use in video games* présentée à Proceedings of the 13th annual conference on Genetic and evolutionary computation, Dublin, Ireland.

Satheesh, S., Narasimhan, H., & Senthil, P. (2009). *Evolving player-specific content for level based arcade games* présentée à Proceedings of the 4th International Conference on Foundations of Digital Games, Orlando, Florida.

Schwarz, M., & Müller, P. (2015). Advanced procedural modeling of architecture. *ACM Trans. Graph., 34*(4), 1-12.

Shek, A., Lacewell, D., Selle, A., Teece, D., & Thompson, T. (2010). *Art-directing Disney's Tangled procedural trees* présentée à ACM SIGGRAPH 2010 Talks, Los Angeles, California.

Smith, G., Othenin-Girard, A., Whitehead, J., & Wardrip-Fruin, N. (2012). *PCG-based game design: creating Endless Web* présentée à Proceedings of the International Conference on the Foundations of Digital Games, Raleigh, North Carolina. Repéré à http://dl.acm.org/citation.cfm?doid=2282338.2282375

Smith, G., Whitehead, J., & Mateas, M. (2010). *Tanagra: a mixed-initiative level design tool* présentée à Proceedings of the Fifth International Conference on the Foundations of Digital Games, Monterey, California.

Tang, M. (2009). *City generator: GIS driven genetic evolution in urban simulation* présentée à SIGGRAPH '09: Posters, New Orleans, Louisiana.

Tessendorf, j. (2002). *Simulating Ocean Water* présentée à SIGGRAPH 2002.

Togelius, J., Kastbjerg, E., Schedl, D., & Yannakakis, G. N. (2011). *What is procedural content generation?: Mario on the borderline* présentée à Proceedings of the 2nd International Workshop on Procedural Content Generation in Games, Bordeaux, France.

Togelius, J., Nardi, R. D., & Lucas, S. M. (2007). Towards automatic personalised content creation for racing games. Dans *Proceedings of IEEE Symposium on Computational Intelligence and Games (CIG)* (pp. 252-259).

Togelius, J., Shaker, N., & Nelson, M. J. (2016). *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer.

Totten, C. W. (2014). *An architectural approach to level design*.

Whelan, G., Kelly, G., & McCabe, H. (2008). *Roll your own city* présentée à Proceedings of the 3rd international conference on Digital Interactive Media in Entertainment and Arts, Athens, Greece.

https://www.gamasutra.com/view/feature/3983/the_history_of_elite_space_the_.php, consulté le 04 aout 2017.