Génération procédurale de niveaux de jeu alliant approche constructive et optimisation

Raphaël Bailly

HESAM UNIVERSITÉ

le cnam

ÉCOLE DOCTORALE Sciences des Métiers de l'Ingénieur

Laboratoire Cédric - Équipe ILJ

THÈSE

présentée par : **Raphaël BAILLY**
soutenue le : **8 décembre 2022**

pour obtenir le grade de : **Docteur d'HESAM Université**
préparée au : **Conservatoire National des Arts et Métiers**

Discipline : **Informatique**
Spécialité : **Informatique**

Génération procédurale de niveaux de jeu alliant approche constructive et optimisation

THÈSE dirigée par :
M. CUBAUD Pierre-Henri Professeur des universités, Cnam

et co-encadrée par :
M. LEVIEUX Guillaume Maître de conférences, Cnam

JURY

M. Ludovic DENOYER	Professeur des universités, La Forge, Ubisoft Bordeaux	Président
Mme Anne-Gwenn BOSSER	Maîtresse de conférences, Lab-STICC, École Nationale d'Ingénieurs de Brest	Rapporteuse
M. Ronan CHAMPAGNAT	Maître de conférences, L3i, La Rochelle Université	Rapporteur
M. Eric GALIN	Professeur des universités, LIRIS, Université Claude Bernard Lyon 1	Examinateur

THÈSE

Affidavit

Je soussigné, Raphaël Bailly, déclare par la présente que le travail présenté dans ce manuscrit est mon propre travail, réalisé sous la direction scientifique de M. Pierre-Henri Cubaud (directeur) et de M. Guillaume Levieux (co-encadrant), dans le respect des principes d'honnêteté, d'intégrité et de responsabilité inhérents à la mission de recherche. Les travaux de recherche et la rédaction de ce manuscrit ont été réalisés dans le respect de la charte nationale de déontologie des métiers de la recherche. Ce travail n'a pas été précédemment soumis en France ou à l'étranger dans une version identique ou similaire à un organisme examinateur.

Fait à Paris, le 12 octobre 2022

Raphaël Bailly

Affidavit

I, undersigned, Raphaël Bailly, hereby declare that the work presented in this manuscript is my own work, carried out under the scientific direction of M. Pierre-Henri Cubaud (thesis director) and of M. Guillaume Levieux (co-thesis supervisor), in accordance with the principles of honesty, integrity and responsibility inherent to the research mission. The research work and the writing of this manuscript have been carried out in compliance with the French charter for Research Integrity. This work has not been submitted previously either in France or abroad in the same or in a similar version to any other examination body.

Place Paris, date 12 October 2022

Raphaël Bailly

À ma famille, à mes amis

À Alvin, mon animal de compagnie,
par sa petite présence réconfortante

Remerciements

Après avoir passé un Master jeux et médias interactifs numériques au Cnam-Enjmin d'Angoulême et effectué un stage chez Ubisoft Paris à Montreuil, j'ai eu la chance et l'honneur d'être sollicité par mes professeurs respectifs, Guillaume Levieux et Axel Buendia pour un projet de recherche aboutissant à une thèse. L'initiative a été rendue possible grâce notamment à la collaboration avec Ubisoft Bordeaux et au financement de la région Nouvelle-Aquitaine.

Ce travail de longue haleine, qui a débuté en janvier 2019, a nécessité la collaboration de nombreuses personnes. Il est donc normal que j'exprime ma reconnaissance envers toutes celles qui m'ont soutenu tout au long de ces années de recherche et de rédaction. Je remercie tout d'abord Pierre-Henri Cubaud, directeur de thèse, ainsi que Guillaume Levieux, co-encadrant, pour leur soutien et leurs enseignements tout au long de cette thèse.

J'aimerais également exprimer ma gratitude envers toute l'équipe ILJ et les membres administratifs du laboratoire Cédric au CNAM de Paris. Je n'ai pas oublié Viviane Gal et nos nombreuses sessions de discussions sur les jeux. Je tiens également à remercier Evelyne Emerit, Nicole Chaouat, Annie-France Aucaucou et Hassan Labiah qui ont répondu avec calme et patience à mes nombreuses sollicitations d'ordre administratif. Leur disponibilité ainsi que leur gentillesse m'ont énormément aidé lors des inévitables démarches à accomplir pour le bon déroulement d'une thèse. De même, je tiens à remercier Samia Bouzefrane et Claire Ryckmans dans l'équipe du suivi doctorant au CNAM, pour leurs conseils.

Je profite de ces quelques lignes pour remercier l'équipe d'Ubisoft de Bordeaux, à savoir, Julien Risse, Simon Contraires, Rayane Benbouazza, parmi d'autres, pour leur prévenance et leurs conseils qui m'ont aidé à progresser dans mes recherches, dans le cadre de notre collaboration durant trois années.

En novembre 2021, j'ai eu la chance de participer à la conférence IFIP ICEC à Coimbra au Por-

tugal. J'ai ainsi eu l'opportunité de rencontrer des chercheurs de diverses nationalités, d'assister à de nombreuses conférences très enrichissantes et d'échanger de nouvelles idées. Je remercie également, plus particulièrement, les membres de l'organisation, dont Licínio Roque, et les membres du doctoral consortium, à savoir Liliane Machado, Artur Lugmayr et Helmut Hlavacs. Une pensée, également, pour Leo Thurler, qui comme moi, a participé à sa première conférence scientifique. Toutes ces personnes, par leur gentillesse et disponibilité, ont rendu mon séjour inoubliable.

J'ai, d'un autre côté, eu la chance de rencontrer Oskar Stålberg lors de l'IndieCade organisée à Paris en 2019. J'ai eu l'honneur de l'interviewer à propos de mes recherches sur les algorithmes de génération procédurale.

Au terme de ce parcours, je tiens également à remercier et exprimer ma gratitude envers tous mes amis pour leur soutien moral et qui m'ont accompagné durant ces longues années, que ce soit en vocal ou en physique.

Pour finir, un grand merci à mes parents et mon frère, pour leur patience et leur encouragement, ainsi qu'à mon parrain pour son humour sans faille. Une pensée pour mes grands-parents, et plus particulièrement, ma grand-mère qui n'aura pas eu la chance de voir l'aboutissement de mon travail.

Résumé

Le travail de recherche effectué dans ce manuscrit se positionne dans le domaine de la génération procédurale de contenu dans le jeu vidéo. Cette thèse s'intéresse plus spécifiquement aux questionnements liés à la diversité et à la qualité du contenu généré. En effet, dans le milieu du jeu vidéo, les développeurs sont amenés continuellement à offrir une palette diversifiée d'espaces de jeu de qualité aux joueurs. Ce défi perpétuel nous a interpellés et nous a conduit à l'étude de nombreuses méthodes et algorithmes, grâce à une revue de littérature de la génération procédurale de contenu. Finalement, nous nous sommes penchés sur la problématique suivante : « Comment aboutir à une méthode de génération proposant une forte diversité d'expériences de jeu, tout en conservant une certaine qualité structurelle dans ses résultats ? » Nous avons centré notamment notre étude sur la conception des niveaux de jeu et le placement d'objets. Nous nous sommes focalisés, plus particulièrement, sur la génération de niveaux 3D ouverts non-linéaires de type jeu de tir à la première personne, avec une infrastructure de campement dont le contenu est positionné sur une grille 2D.

Cette thèse propose une nouvelle méthode, nommée *Genetic-WFC*, dans le but d'offrir une diversité d'expériences de jeu avec des niveaux possédant une certaine qualité structurelle. Il s'agit d'un pipeline de génération procédurale qui combine une approche *Search-Based*, à savoir un algorithme génétique et une évaluation par simulation, avec une méthode constructive, le *Wave Function Collapse*, afin de générer des niveaux ciblant des expériences de jeu spécifiques. Le *Wave Function Collapse*, abrégé WFC, est un algorithme de propagation de contraintes locales d'adjacence. Dans notre approche, il extrait ces contraintes à partir d'exemples de niveau, et nous permet d'effectuer la recherche génétique sur des résultats ne présentant pas d'erreurs de placement d'objets. Il agit en tant qu'opérateur de réparation pour les individus de la population de l'algorithme génétique. Le pilotage du WFC, par l'algorithme de recherche, est rendu possible en influençant la probabilité de sélection de ses éléments. Nous employons une solution de réencodage des niveaux qui nous permet d'améliorer le processus

d'optimisation de notre algorithme évolutionnaire. Nous utilisons également un joueur synthétique pour évaluer l'expérience de jeu à l'aide de trois heuristiques de perception, à savoir, la nouveauté, la sécurité et la complexité, lors d'une simulation d'un parcours.

Diverses expérimentations sur notre approche ont été menées afin d'établir ses capacités et ses performances. Après s'être intéressé au temps de calcul du WFC pour la génération de niveaux, une seconde expérience s'est attachée, plus particulièrement, à comparer notre méthode à d'autres similaires en prenant en compte, notamment, le temps de calcul et la valeur de score des résultats obtenus. Nous y confrontons également les différences visuelles entre certains niveaux produits par ces diverses méthodes. Une dernière expérience repose, quant à elle, sur l'évaluation de la diversité des expériences de jeu que peut proposer notre algorithme de génération procédurale.

Nous terminons cette thèse en évoquant plusieurs pistes d'amélioration et de poursuite de recherche, qui peuvent encore être approfondies. Une expérience utilisateur avec des campements jouables pourrait représenter, par exemple, une prochaine étape dans l'étude de notre méthode.

Mots-clés : Génération Procédurale de Contenu, Jeu Vidéo, Level Design, Diversité, Algorithme Génétique, Wave Function Collapse, Opérateur de Réparation, Expérience de Jeu, Simulation, Intelligence Artificielle

Abstract

The research work in this thesis is positioned in the field of procedural content generation in video games. This study focuses more specifically on the questions related to the diversity and quality of the generated content. Indeed, in the video game industry, developers are constantly challenged to offer a diverse range of quality game environments to players. This perpetual quest led us to study numerous methods and algorithms, through a literature review of procedural content generation. Eventually, we addressed the following problem : « How to achieve a generation method that offers a high degree of diversity of game experiences, while maintaining a certain structural quality in its results ? » Our study is centred in particular on the level design and the placement of objects in game levels. We also targeted a generation of non-linear open 3D levels for a first-person shooter, with a settlement infrastructure whose contents are positioned on a 2D grid.

This thesis introduces a new method, named *Genetic-WFC*, with the aim of providing a diversity of game experiences with levels having a certain structural quality. It is a procedural generation pipeline that combines a *Search-Based* approach, consisting of a genetic algorithm and simulation-based evaluation, with a constructive method, the *Wave Function Collapse*, to generate levels targeting specific game experiences. The *Wave Function Collapse*, abbreviated WFC, is an algorithm for propagating local adjacency constraints. In our approach, it extracts these constraints from level examples, and allows us to perform genetic search on results that do not exhibit object placement errors. It serves, in fact, as a repair operator for the individuals in the population of the genetic algorithm. The driving of the WFC, by the search algorithm, is made possible by influencing the selection probability of its elements. We employ a level re-encoding solution that allows us to improve the optimisation process of our evolutionary algorithm. We also use a synthetic player to evaluate the game experience using three perception heuristics, namely, the novelty, the safety and the complexity, during a simulation of a walkthrough.

Various experiments on our method have been conducted in order to establish its capabilities and performance. After looking at the computation time of the WFC for the generation of levels, a second experiment focused on comparing our approach to other similar methods, taking into account, in particular, the computation time and the score value of the results obtained. We also look at the visual differences between certain levels produced by these various methods. A last experiment is based on the evaluation of the diversity of game experiences that our procedural generation algorithm can provide.

We conclude this thesis by mentioning several areas of improvement and further research, which can be pursued more thoroughly. For example, a user experience with fully playable settlements could be a next step in the study of our method.

Keywords : Procedural Content Generation, Video Games, Level Design, Diversity, Genetic Algorithm, Wave Function Collapse, Repair Operator, Game Experience, Simulation, Artificial Intelligence

Table des matières

Liste des tableaux

Table des figures

Chapitre 1

Introduction

Contenu

1.1 Contexte de la recherche

L'univers du jeu vidéo est le théâtre d'une concurrence décuplée d'année en année entre les nombreux studios et les développeurs, qu'ils soient indépendants ou non. En effet, les joueurs peuvent passer facilement d'un jeu à l'autre en fonction de leurs envies et du fait d'un très grand choix de jeux disponibles sur le marché [27]. Par conséquent, l'industrie du jeu doit sans cesse s'adapter et se transformer selon les préférences et la demande des joueurs. La Figure 1.1 page suivante, illustre la montée vertigineuse du nombre de jeux, publié par an, uniquement sur la plateforme Steam, l'une des plus populaires pour la diffusion de jeux numériques [28].

Parmi ces nombreux jeux sur le marché, il existe des jeux dits « Triple-A » [1], qui sont de plus en plus ambitieux, que ce soit en termes de coût ou de contenu. Certains de ces jeux incorporent également le principe appelé « jeu en tant que service », ou *Game As A Service* abrégé GAAS en anglais. Il s'agit, notamment, de rajouter du contenu régulièrement, sur plusieurs années par exemple, dans le but de fidéliser et de maintenir l'intérêt du joueur le plus longtemps possible [30]. Nous pouvons citer, par exemple, le jeu *Rainbow Six : Siege* [31]. Que ce soit pour des jeux d'envergure ou de plus petits projets, les développeurs cherchent à offrir des concepts uniques, comme le jeu Spore [32], par exemple, ou encore une forte rejouabilité.

Par ailleurs, le développement de jeux est devenu particulièrement exigeant en raison de l'évolution rapide de la technologie informatique, notamment des différentes plateformes et des moteurs de jeux [27]. De surcroît, ce processus de développement est une activité pluridisciplinaire qui requiert une synergie de talents créatifs et techniques afin de donner vie à un concept. En plus de la main-d'œuvre, viennent s'ajouter le manque de temps et le coût parfois élevé d'une production sur une longue durée. En effet, les équipes de développement créent habituellement le contenu du jeu à la main [Chapitre 4][4]. Ainsi, offrir une palette diversifiée d'espace de jeu de qualité reste un enjeu majeur pour tous les développeurs.

1. Le terme « Triple-A » est un terme de classification utilisé pour les jeux vidéo dotés de budgets de développement et de promotion très élevés [29].

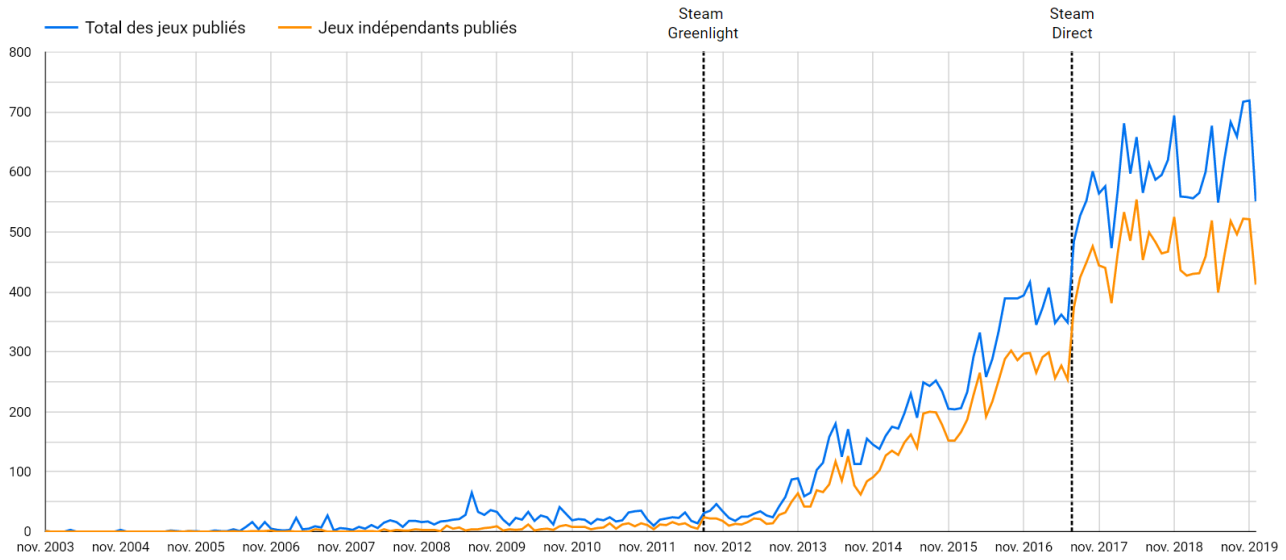


FIGURE 1.1 – Évolution du nombre des jeux publiés sur Steam au fil du temps, par mois, de 2003 à 2019 [1].

Dans cette optique, une solution consisterait à déléguer le travail de développement à la machine grâce à une technique particulière : il s'agit de la génération procédurale de contenu, *Procedural Content Generation* ou PCG en anglais. Cette thématique, répandue dans le domaine de la création de jeux vidéo, fait référence à la création automatique de contenu de jeu à l'aide d'algorithmes. Ces derniers peuvent être utilisés dans le but de créer de nouvelles façons de jouer, ou encore de proposer des méthodes différentes pour concevoir des jeux vidéo [33]. Ce concept peut, en effet, prêter main-forte aux développeurs. Le travail effectué lors du développement de jeu, ou la création de contenu numérique, peut être délégué à ces solutions de PCG. Ces dernières peuvent, entre autres, automatiser des tâches répétitives, améliorer la rejouabilité, ou assister directement l'utilisateur lors de la conception. Ces nombreuses applications expliquent pourquoi la PCG est au centre de nombreuses recherches. Ainsi, de multiples méthodes et algorithmes ont déjà été étudiés et développés. Plusieurs jeux vidéo en font d'ailleurs usage depuis plusieurs dizaines d'années et certains l'utilisent même comme principal élément de gameplay [2], par exemple : *No Man's Sky* [5] et *Minecraft* [36] où les terrains de jeu y sont entièrement procéduraux.

Néanmoins, utiliser de telles méthodes de PCG peut être risqué, car le contenu généré doit avant

2. Le gameplay définit la manière dont les joueurs peuvent interagir avec le jeu et comment ce dernier réagit à leurs actions [34, Page 661]. Il peut se décrire sous la forme de défis et d'actions que le joueur doit exécuter pour atteindre l'objectif correspondant à un défi spécifique [35].

tout rester attrayant pour le joueur, c'est-à-dire proposer de nouvelles expériences excitantes et différentes. La quantité de contenu dans un jeu ne rend pas celui-ci forcément plus intéressant. Il y a donc un compromis à trouver entre la quantité et la qualité de contenu généré. De ce fait, développer un générateur proposant des résultats diversifiés tout en préservant le niveau de qualité n'est pas une tâche aisée.

Notre travail de recherche se positionne dans ces questionnements liés à l'utilisation de la PCG. Il nous semble que la diversité et la qualité représentent, en effet, des critères intéressants à développer. C'est pourquoi, nous allons chercher à approfondir et à proposer de nouvelles pistes à explorer dans cette direction.

1.2 Plan de la thèse

Dans le chapitre suivant, nous commençons par l'introduction des différents concepts, méthodes et travaux liés à la génération procédurale de contenu dans le jeu vidéo. Nous y détaillons les principales familles de méthodes qui nous ont paru particulièrement intéressantes pour notre recherche. Cet état de l'art a pour but de guider notre raisonnement vers une piste à explorer pouvant mener vers une solution innovante dans le domaine de la PCG.

Dans le Chapitre 3, nous décrivons notre choix de problématique face à l'état de l'art et notre proposition de solution appropriée sous la forme d'une nouvelle méthode. Nous y exposons nos axes de recherches, nos limitations et notre positionnement face aux travaux déjà existants.

Dans cette continuité, le Chapitre 4 s'articule autour des difficultés envisagées et aborde nos hypothèses de développement concernant notre approche. Quelques questionnements sur la faisabilité sont également présentés.

Le Chapitre 5 décrit, quant à lui, le fonctionnement de notre pipeline de génération procédurale, proposé précédemment. Nous y développons sa mise en place, et plus spécifiquement ses trois parties le composant, c'est-à-dire notre implémentation du *Wave Function Collapse*, notre algorithme génétique employé et finalement notre évaluation par simulation et ses heuristiques.

Le Chapitre 6 est dédié à l'évaluation de la pertinence et des capacités de notre méthode, au travers de trois expérimentations et des résultats obtenus. La première expérience consiste à examiner le temps de calcul uniquement du WFC. La deuxième se concentre sur l'évaluation et la comparaison

de notre approche face à d'autres méthodes similaires. En dernier lieu, nous présentons une analyse de la diversité des expériences de jeu que peut proposer notre méthode au travers de l'exploration de l'espace de génération de niveau.

Le Chapitre 7 nous permet d'amener une discussion et de mettre en perspective l'ensemble des travaux réalisés dans cette thèse. Nous nous y interrogeons sur la pertinence de notre démarche face à notre problématique et au domaine de la recherche. Cette réflexion nous a amené à considérer les apports et les inconvénients que soulève notre méthode. Nous terminons par la proposition de nombreuses pistes à explorer ainsi que celles qui ont été déjà amorcées, avec, entre autres, l'élaboration d'une ébauche d'expérience utilisateur.

Le dernier chapitre est dédié à la conclusion générale. Nous y présentons une synthèse de nos travaux menés au sein de cette thèse.

L'Annexe A porte sur une étude des campements, présents, entre autres, dans les jeux d'Ubisoft. L'Annexe B, présente en détail des valeurs montrées dans l'expérience 2. L'Annexe C expose la totalité des mosaïques illustrant les résultats obtenus dans la troisième expérimentation.

1.3 Financement et collaboration

Le travail de cette thèse a bénéficié d'une aide financière de la région Nouvelle-Aquitaine, dans le cadre du projet « Kiwi » sous la convention de subvention n°18001957, et d'une contribution de la BPI pour le projet « United VR ».

Nous avons également eu l'occasion de collaborer le temps de cette recherche avec le studio Ubisoft Bordeaux, dans le contexte du projet « Kiwi ». En effet, celui-ci se penche également sur les problématiques liées à l'utilisation de la PCG dans la création des jeux vidéo. Les développeurs d'Ubisoft se sont intéressés plus particulièrement au sujet de la génération procédurale de campements [3], à forte variabilité perçue, dans la perspective de proposer du contenu pour leurs jeux à monde ouvert. Leur question portait ainsi principalement sur l'étude de la perception de variabilité chez le joueur dans l'attaque et/ou l'infiltration de campement dans un monde ouvert. Ces différentes réflexions sur le

3. Un fortin ou campement ennemi, est un lieu comportant divers objectifs à atteindre pour le joueur, tels que la libération de prisonniers ou la collecte d'informations. La méthode d'approche est très souvent laissée au choix du joueur, voir Annexe A page 213.

level design[4] des niveaux se trouvaient être complémentaires de notre sujet de recherche. Nous avons choisi effectivement, dans cette thèse, d'explorer la direction plus spécifique, centrée sur la diversité et la qualité, qui nous semble être une étape indispensable pour résoudre ces questionnements évoqués ci-dessus.

1.4 Publications

Article de journal

R. Bailly, G. Levieux, « Genetic-WFC : Extending Wave Function Collapse with Genetic Search », *IEEE Transactions on Games*, 2022.

Conférence

R. Bailly, G. Levieux, « Genetic WFC : Procedural Level Generation Maximizing Perceived Diversity », *IFIP ICEC - Doctoral Consortium*, Coimbra Portugal, 2021.

4. Le *level design* est le processus dans la création de jeu vidéo qui s'occupe de la réalisation des niveaux, c'est-à-dire la construction de l'espace de jeu à proprement parler [34, Chapitre 23].

Chapitre 2

État de l'art de la génération procédurale de contenu pour le jeu vidéo

Contenu

2.1 Introduction

Ce chapitre va présenter une étude bibliographique qui s'inscrit dans le contexte de cette thèse, à savoir l'utilisation de la génération procédurale de contenu dans le domaine du jeu vidéo. Nous avons choisi de nous intéresser de manière générale à son concept et à ses caractéristiques, ainsi qu'aux principales méthodes et travaux déjà existants. Nous pensons qu'offrir une vue d'ensemble est essentielle à la compréhension des avantages et des problématiques concernant la génération procédurale de contenu.

Dans la réalisation de cette revue de la littérature, nous proposons, dans un premier temps, d'introduire le concept général de la génération procédurale de contenu ainsi que ses diverses propriétés. Puis, nous développerons les diverses familles d'algorithmes, leurs approches et leurs applications, qui nous semblent pertinentes et qui sont utilisées dans le domaine de la recherche et dans les jeux vidéo.

2.2 Génération procédurale de contenu

2.2.1 Présentation du concept

La génération procédurale de contenu, ou *Procedural Content Generation* abrégé PCG en anglais, est une méthode permettant de générer du contenu à l'aide d'algorithmes avec une entrée limitée ou indirecte de la part d'un utilisateur, par opposition à la création manuelle du contenu par un être humain [33]. Elle est souvent utilisée dans le domaine du jeu vidéo, mais également pour la synthèse d'images et les effets spéciaux de films entre autres. Plus récemment, Georgios N. Yannakakis a présenté la PCG comme un champ de l'intelligence artificielle du jeu vidéo qui a vu son intérêt décuplé ces dernières années, non seulement dans les productions de jeux, mais également dans différents domaines de la recherche académique [4, Chapitre 4].

Il nous semble intéressant de noter que la PCG a joué un rôle dans le *game design*[1] avant les jeux vidéo numériques, tels que les jeux de rôle papier [38]. De plus, certains algorithmes, toujours utilisés de nos jours, n'étaient pas conçus au départ pour les jeux vidéos. À ce titre, nous pouvons citer les *L-System* utilisés dans le domaine de la biologie et l'étude des plantes [39], mais également le « bruit de Perlin » permettant la réalisation d'effets spéciaux dans le domaine du cinéma [40].

1. Le *game design* correspond à la conception théorique du jeu vidéo : style, gameplay, genre de jeu, etc [37]. Il s'agit du processus de création et de mise au point des règles et autres éléments constitutifs d'un jeu.

Des implémentations de générations procédurales peuvent être retracées dès les années 80, notamment dans le jeu de simulation de commerce spatial *Elite* [41] générant de nouveaux systèmes planétaires avec leur population et leur propre système économique et politique. Nous pouvons citer également le jeu de rôle *Rogue* [3] de type *dungeon crawler*[2] qui crée automatiquement ses niveaux et y place des objets. *Rogue*, voir capture d'écran Figure 2.1, est considéré comme le premier jeu « graphique » d'aventure et a été populaire plusieurs années, notamment grâce à ses niveaux générés à chaque partie par des algorithmes. De plus, il est considéré comme le pionnier du genre auquel il donne son nom, le *rogue-like*[3].

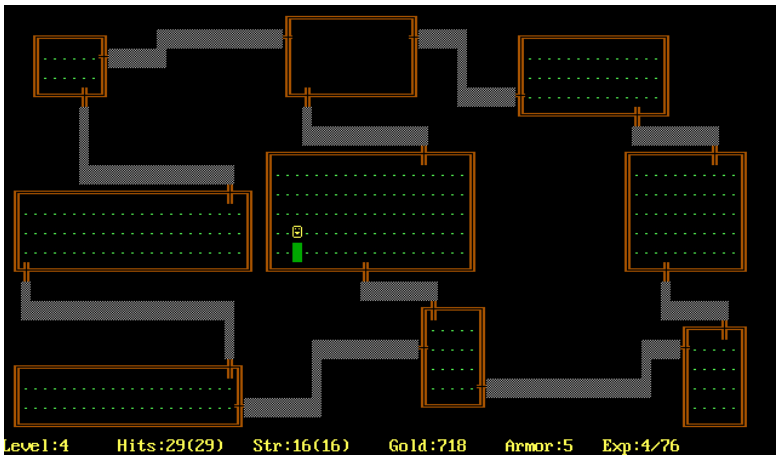


FIGURE 2.1 – Un niveau généré de manière procédurale [2] dans le jeu *Rogue* [3].

Nous allons maintenant nous pencher sur cette formulation spécifique qu'est la *Procedural Content Generation* dans le domaine du jeu vidéo. Les termes *Procedural* et *Generation* impliquent la présence d'une procédure ou d'un algorithme produisant un résultat considéré comme du contenu. Ce dernier se réfère aux différentes parties qui constituent un jeu, à savoir les niveaux, les terrains, les mécaniques de jeu, les textures, les scénarios, les quêtes, les objets, les musiques, les armes, les véhicules, les personnages, etc. [21]. En théorie, un jeu pourrait générer entièrement son contenu.

De manière générale, nous parlons de système PCG dès que nous incorporons une méthode de PCG en tant que l'une de ses composantes, par exemple un jeu s'adaptant aux actions du joueur ou un outil de création assisté par intelligence artificielle [21, Chapitre 1]. En effet, une méthode de PCG

2. Le *dungeon crawler* est un genre de jeu vidéo de rôle dont le gameplay privilégie l'exploration de donjons [42].
3. Le *rogue-like* est un genre de jeu caractérisé par la génération aléatoire de ses environnements et la mort permanente impliquant de recommencer le jeu à chaque échec [43].

est exécutée par l'ordinateur, avec ou sans intervention humaine, pour aboutir à un résultat. Dans le cas de l'implication d'un développeur ou d'un joueur, nous pouvons nous référer à une méthode dite *mixed-initiative* [4]. Celle-ci peut être, par exemple, un outil d'aide à la création de quêtes pour des jeux de donjon [5] [46].

Un système de PCG peut être utilisé pour générer du contenu inédit, modifier ou créer des combinaisons nouvelles de contenus déjà existants. L'algorithme peut tout à fait générer le contenu en une fois, ou le produire en continu [33]. De plus, le système va pouvoir assurer un résultat différent grâce à des modificateurs aléatoires et à des paramètres accessibles à un utilisateur, développeur ou joueur. Nous pouvons citer comme paramètre, par exemple, la notion de rythme qui a été explorée notamment pour la génération de niveau de plateformes [47]. Selon la complexité de l'algorithme, il sera possible de générer des attributs aléatoires pour des armes [48] jusqu'à des systèmes planétaires réalistes [49].

D'après la taxonomie de Georgios N. Yannakakis et Julian Togelius [4, Chapitre 4], il existe plusieurs propriétés, voir Figure 2.2 page suivante, dans lesquelles peuvent être classifiés : le contenu généré, l'algorithme ou la méthode en elle-même, et le rôle de la PCG dans le processus de conception du jeu. Ainsi, nous pouvons, tout d'abord, distinguer le contenu créé par l'algorithme comme nécessaire ou optionnel à la complétion d'un niveau. En ce qui concerne les méthodes, elles peuvent intégrer une notion d'aléatoire ou non, ce qui peut permettre une variation des résultats. D'un autre côté, ces méthodes peuvent également être contrôlées selon divers paramètres qui influenceront ou non l'espace de génération. Finalement, nous pouvons distinguer les méthodes dites « constructives » et celles dites par « essai-erreur », ou *generate-and-test* en anglais. Cette distinction sera détaillée dans la partie traitant des méthodes principales de PCG, à savoir la Section 2.3 page 39. La taxonomie propose également une classification des rôles de la PCG. Il peut s'agir de développer un système autonome qui n'est pas influencé par un être humain ou à l'opposé de proposer un outil d'aide à la création qui implique, dans ce cas, la présence d'un développeur. Quant à la dernière notion, elle sépare les générations de contenu qui sont dépendantes ou non du comportement du joueur. Cela peut permettre ou non la personnalisation et l'adaptation du contenu.

4. Bien que l'initiative mixte n'ait pas de définition concrète, nous considérons que l'humain et l'ordinateur contribuent de manière proactive à la résolution d'un même problème [44]. Dans le cadre de la création de contenu, ce dernier est produit par des cycles itératifs entre le concepteur humain et le système procédural [45].

5. Dans le jeu vidéo, un donjon représente un environnement labyrinthique dans lequel des aventuriers collectent des trésors, évitent ou tuent des monstres, sauvent des personnes, tombent dans des pièges [21, Chapitre 3].

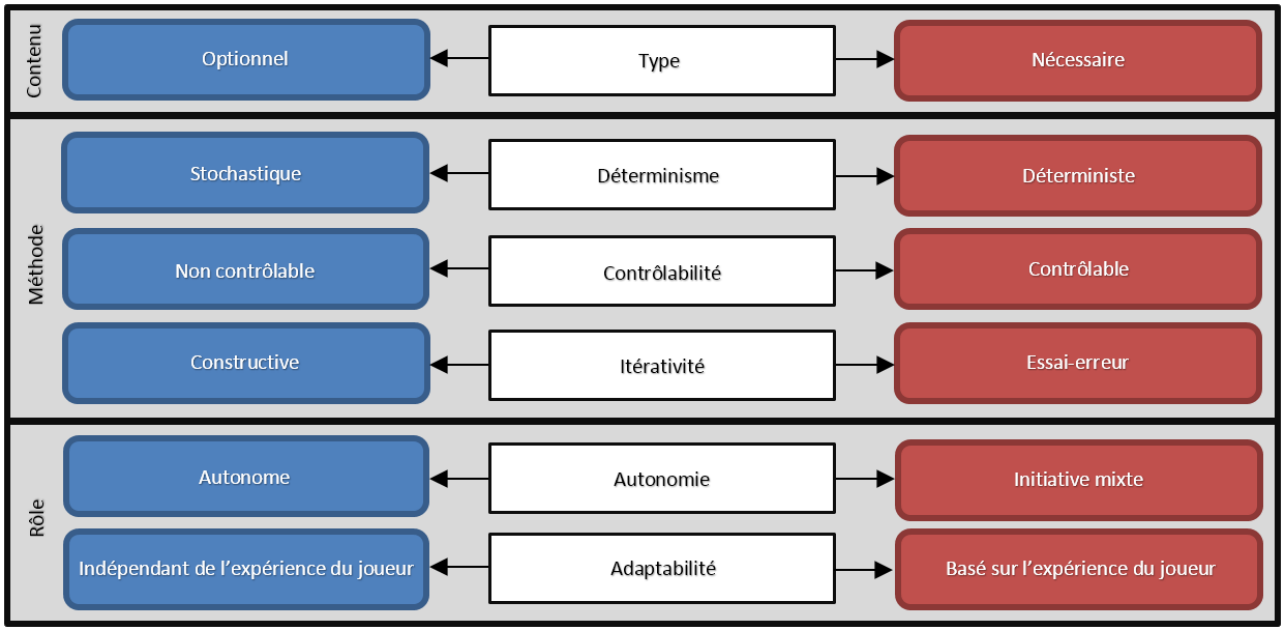


FIGURE 2.2 – Taxonomie des approches de PCG par Georgios N. Yannakakis et Julian Togelius [4, Figure 4.2].

Il nous parait important de détailler une propriété supplémentaire par rapport à la taxonomie détaillée ci-dessus, à savoir « en ligne », ou *online* en anglais, et « hors ligne », ou *offline* en anglais, mentionnée par Noor Shaker [21, Chapitre 1]. Une génération dite « en ligne » signifie que le contenu est généré pendant l'utilisation du média par un utilisateur. À l'inverse, la génération dite « hors ligne » implique du contenu généré durant le développement du jeu ou antérieur au lancement d'une session de jeu. Ces deux approches présentent des avantages et des inconvénients que nous allons énumérer ci-dessous.

Premièrement, avec la création « en ligne », il est nécessaire de toujours garantir un contenu jouable. Cette approche offre la possibilité de créer des variations infinies et ainsi de rendre un jeu en théorie infiniment rejouable. Elle permet aussi de proposer un jeu adapté à chaque joueur et ainsi de générer du contenu personnalisé d'après chaque profil. Un exemple de ce type de génération « en ligne » se trouve dans le jeu *Left 4 Dead* [50] avec son expérience dynamique pour chaque joueur grâce à l'*AI Director* qui analyse les comportements des joueurs en temps réel et altère l'état du jeu en utilisant différentes techniques de PCG [51]. Ainsi, dans ce jeu, les joueurs sont confrontés à des rencontres imprévisibles d'ennemis, différentes à chaque partie ; le système renouvelle les expériences en faisant varier les fréquences d'apparitions de zombies et d'objets afin de maintenir une certaine tension. La

génération dite « hors ligne », quant à elle, offre la possibilité de créer du contenu en amont et permet d'allouer davantage de mémoire et de temps machine au processus, tout en maintenant un contrôle sur la création. De plus, le résultat peut être retravaillé manuellement avant d'être proposé au joueur. De ce fait, l'utilisation de génération « hors ligne » est utile afin de faciliter la création de contenus détaillés, tels que les terrains et les environnements réalistes, retouchés ensuite par les développeurs [52].

Une dernière distinction peut se faire selon le degré d'implication de la PCG dans un jeu. On peut ainsi distinguer trois catégories d'après Gillian Smith [53] :

- *Core*, quand la partie principale d'un jeu est générée procéduralement. Nous pouvons citer comme exemple *No Man's Sky* [5] avec ses systèmes planétaires, voir Figure 2.3, et *Minecraft* [36] avec son univers composé de *voxels*[6], car les terrains de jeu y sont entièrement procéduraux.
- *Partial Framing*, quand certaines parties du jeu sont réalisées par PCG, par exemple les armes dans Borderlands [48].
- *Decorative*, quand le contenu créé est uniquement esthétique, comme des textures ou de la végétation.

FIGURE 2.3 – Paysage d'une planète générée de manière procédurale dans le jeu *No Man's Sky* [5].

6. Le voxel est un pixel en 3D.

2.2.2 Utilité de la PCG

Il existe de nombreuses raisons qui justifient l'utilisation et le développement de solutions de PCG dans un jeu, que nous détaillons ci-dessous.

Compression Comme nous l'avons souligné précédemment, les premiers jeux exploitant un algorithme de PCG datent des années 80. À l'époque, il s'agissait avant tout d'économiser de l'espace de stockage dû aux limitations techniques. En effet, un des avantages notables de la PCG est sa possibilité de compresser du contenu. Avec des algorithmes, il est possible de générer par exemple des textures, des musiques et des niveaux de jeu à la volée et ainsi de réduire la taille des données sur le support de stockage. Cela a permis en premier lieu de produire des jeux avec davantage de contenu, comme dans le jeu *Elite* [41] avec ses centaines de systèmes planétaires qui nécessitent uniquement quelques dizaines de kilooctets. Plus récemment, en 2004, le projet *.kkrieger* [54], un jeu 3D de type *First Person Shooter*, jeu en vue subjective, proposa un contenu graphique impressionnant et ceci pour une taille très réduite de 96 ko. Les graphismes, principalement les textures, mais aussi les sons et les maillages 3D, y sont tous générés procéduralement.

Coût En 2005, lors de sa présentation *The Future of Content* à la *Game Developers Conference*, Will Wright a contribué à raviver l'intérêt de la PCG pour l'industrie du jeu vidéo. Selon lui, une entreprise pouvant remplacer des employés par des algorithmes aurait un avantage compétitif : la production des jeux pourrait se faire plus rapidement et à moindre coût, tout en conservant un contenu de qualité. L'augmentation de la complexité de la production d'un jeu, dit « AAA », implique un effort non négligeable en termes de coût humain et de temps [55]. En effet, bien que des outils de création automatisés soient présents dans des productions afin d'accélérer la vitesse de développement, la création d'environnements 3D réalistes, les nombreux détails et le *level design* sont généralement encore réalisés en grande partie manuellement. Ainsi, le nombre de designers et d'artistes représentent une part importante du budget de développement. En effet, d'après une analyse du budget des jeux AAA, la création artistique de contenu, graphique et sonore par exemple, constitue environ 40% du coût total de production [56].

Actuellement, la technologie ne permet pas encore de remplacer entièrement le travail d'un être humain. Cependant, grâce à des outils de PCG utilisés à bon escient, la création de contenu peut

s'accélérer et favoriser l'itération. La créativité des développeurs peut ainsi se retrouver accrue. En effet, le résultat d'une solution de PCG peut être inattendu et différent de ce qu'un humain aurait pu proposer. Par exemple, le système procédural du projet *Galactic Arms Race* [57] a créé des armes originales, surprenant les créateurs eux-mêmes [58]. La génération procédurale présente de ce fait un aspect de collaboration entre l'artiste et la machine [44]. Certaines approches de ce genre de PCG sont courantes dans l'industrie. En effet, les grosses productions utilisent généralement la génération procédurale pour créer de grandes quantités de contenu automatiquement, tels que les terrains [52] et les villes [59], et ceci grâce à divers outils mis à la disposition des développeurs. Le résultat peut ensuite être retravaillé manuellement. De plus, nous pouvons citer comme exemple d'outils, l'utilisation de *middleware* tels que *SpeedTree* [60] pour la génération d'arbres, *Euphoria* [61] pour l'animation en temps réel et *CityEngine* [62] pour les villes.

Finalement, la PCG a ce potentiel d'offrir à tous, de nouvelles approches fiables et accessibles, permettant le développement de meilleurs jeux riches en contenu en un temps réduit et donc d'être attractif pour de plus petites équipes de développeurs [63].

Nouveau genre de jeu Les méthodes de PCG ont également engendré l'apparition de nouveaux concepts de jeu, sans véritable achèvement possible. Nous pouvons citer *Minecraft* [36] ou encore plus récemment *No Man's Sky* [5] pour la génération de terrains et *Dwarf Fortress* [64], jeu de construction et de gestion de colonies de nains, pour la simulation d'un monde virtuel. Ce sont des jeux uniques en leur genre qui n'auraient pu exister sans l'aide de la PCG. En effet, le contenu y est sans cesse généré durant le temps de jeu avec la promesse d'une jouabilité potentiellement infinie.

De plus, la PCG peut aussi véhiculer une intention scénaristique, telle que dans *No Man's Sky* où la PCG a comme but de faire ressentir au joueur un sentiment de solitude [65]. Ce jeu, à l'instar de *Minecraft*, propose une nouvelle forme de narration dite émergente [66].

Rejouabilité Dans un jeu où le monde est généré procéduralement, le joueur est encouragé à l'exploration. A contrario, dans les jeux où uniquement les niveaux individuels sont procéduraux, le joueur est incité à rejouer la même partie dans une configuration différente. Cette dernière approche, modifiant les niveaux, est souvent appliquée dans les jeux de type *rogue-like*. La PCG offre ainsi la possibilité de créer des variations ou des mondes infinis et, grâce à cette diversité, d'augmenter la rejouabilité [57, 67].

Adaptabilité En combinant PCG et modélisation du profil du joueur, nous pouvons obtenir un jeu reflétant les envies et les attentes de ce dernier afin de maximiser son plaisir de jeu. Cela peut s'appliquer également à l'apprentissage des *serious game* et à l'addiction des *casual game*. En effet, en utilisant la PCG, le jeu peut fournir une expérience différente, propre à chaque utilisateur. Plusieurs études se sont déjà penchées sur cet aspect, par exemple [68, 69, 70, 71]. Avec un système d'intelligence artificielle déduisant les capacités du joueur, la PCG peut être utilisée comme un ajustement dynamique de la difficulté. Il est alors possible de modifier le contenu à la volée afin d'adapter le jeu au niveau de compétence du joueur [72, 73, 74].

Pour conclure, nous avons vu que la PCG a la capacité de changer radicalement notre façon de conceptualiser les jeux. Elle doit également faire face à des défis importants en ce qui concerne son intégration dans les pratiques de l'industrie du jeu vidéo [4, Chapitre 4]. En effet, l'utilisation de la PCG peut aider à accomplir plusieurs objectifs, mais il est important de considérer ses limites et ses nombreux écueils.

2.2.3 Propriétés souhaitables d'une solution de PCG

Une méthode de PCG implique toujours des compromis, souvent entre vitesse et qualité, créativité et fiabilité. Ci-dessous, nous détaillons une liste non exhaustive de propriétés modulables pour une solution de PCG proposée initialement par Noor Shaker [21, Chapitre 1].

Vitesse Les exigences de vitesse peuvent varier énormément, allant de quelques millisecondes jusqu'à potentiellement plusieurs mois. En effet, la génération en temps réel dans un jeu final ou durant le développement du jeu en studio, influence le temps alloué à l'algorithme afin de produire le contenu.

Fiabilité Certains contenus peuvent être considérés plus importants que d'autres, par exemple, la structure d'un niveau de jeu avec une entrée et une sortie par opposition à un objet purement décoratif. Les contenus doivent donc parfois garantir la satisfaction d'un certain nombre de contraintes de qualités afin de proposer du contenu jouable. Selon le cas, un générateur peut ou non respecter ces exigences.

Contrôlabilité Le générateur peut être piloté par un humain ou même directement par un autre algorithme, tel qu'un système d'adaptation personnalisé, basé sur un modèle du joueur. Il en résulte la possibilité de faire varier les paramètres et les aspects de la génération selon plusieurs dimensions de contrôle.

Expressivité et diversité Développer un générateur proposant des résultats diversifiés tout en préservant le niveau de qualité n'est pas une tâche aisée. En effet, il existe deux extrêmes quant à la diversité possible d'un générateur. Il peut, d'un côté, créer de façon aléatoire tout le contenu et ainsi produire un résultat chaotique et même injouable. À l'inverse, il peut reproduire à l'identique et modifier des paramètres considérés comme insignifiants, par exemple, la couleur d'une texture au milieu d'un niveau entier généré. En effet, la modification de petits détails ne sera pas perçue par le joueur comme forcément intéressante. Néanmoins, des études se sont penchées sur ce problème afin de fournir une mesure de l'expressivité des générateurs de contenu, à savoir [75, 76, 77, 78, 79, 80].

Créativité et crédibilité Le contenu généré doit, en général, être perçu comme produit manuellement et non par une machine. Par exemple, en évitant la répétition d'un élément précis de contenu, le niveau peut donner l'impression d'être original et ainsi d'avoir été conçu par un être humain.

2.2.4 Limites des systèmes de PCG

Un jeu sans fin, et possédant une rejouabilité infinie, peut utiliser une génération procédurale capable de produire du contenu varié et en quantité suffisante. Toutefois, cela ne garantit pas l'aspect qualitatif du résultat. En effet, la quantité de contenu ne rend pas forcément un jeu intéressant. Il y a donc un compromis à trouver entre quantité, diversité et temps alloué au développement du jeu. Le développeur devra chercher le bon équilibre et déterminer les limites de la PCG qu'il veut utiliser. Ceci explique que l'utilisation de techniques de PCG dans l'industrie est souvent limitée à un aspect bien défini du jeu.

Actuellement, il n'est pas encore possible de créer un jeu uniquement à partir de génération procédurale, c'est-à-dire de déléguer tous les choix du développement à des algorithmes, ceci allant de l'aspect visuel et sonore jusqu'aux mécaniques de jeu. Cependant, des recherches sont menées activement dans cette direction afin d'automatiser le design d'un jeu, comme le montre le système *Angelina* [81],

la création de mini-jeux [82] inspirés de la série *Wario Ware* [83], ou encore la génération de game design par contraintes [84]. D'autres études cherchent également à définir ce qui constitue un jeu et comment l'exprimer, par exemple, soit à l'aide d'un langage descriptif [85] ou, soit par des *design patterns*[7] [88, 89].

2.3 Principales méthodes de PCG

Les principales approches et méthodes de génération de contenu seront introduites dans cette section. Dans les deux premières parties, nous allons, d'une part, distinguer les algorithmes constructifs, générant en une seule fois le contenu, et d'autre part les méthodes essai-erreur, qui se concentrent plutôt sur l'évaluation jusqu'à l'obtention d'un résultat satisfaisant. Ainsi, le choix de l'une ou l'autre de ces méthodes dépendra de notre envie de se concentrer, soit sur l'algorithme de création, soit sur celui d'évaluation. Par la suite, nous présenterons des algorithmes de résolution de contraintes et des approches utilisant différentes grammaires, pouvant concevoir des niveaux entiers, basés sur des principes de gameplay. Ces méthodes peuvent également être considérées comme constructives ou comme essai-erreur, selon leur implémentation.

En outre, nous aurons l'occasion de citer des références d'applications de ces techniques, mais également de cerner les avantages et les compromis de chaque implémentation possible.

De plus, nous pouvons conseiller les taxonomies et les études de Mark Hendrikx [90] et Julian Togelius [12] qui offrent des détails supplémentaires sur les algorithmes existants. Le livre *Procedural Content Generation in Games* [21] donne également davantage de précisions et présente des applications concernant la PCG en général.

2.3.1 Méthodes constructives

Un algorithme constructif va générer le contenu étape par étape, pour aboutir à un résultat représentant une solution considérée comme correcte, selon des critères définis de qualité. Il n'y a pas de processus itératif d'évaluation permettant d'améliorer continuellement le contenu généré, contrairement à l'approche essai-erreur, détaillée Sous-Section 2.3.2 page 47.

7. Un *design pattern* est une solution communément admise en réponse à un problème récurrent de conception [86]. Dans le jeu vidéo, il peut s'agir d'un arrangement caractéristique de game design ou level design, par exemple : un passage stratégique étroit, *choke point* en anglais, pour les jeux de tir exposant ainsi le joueur aux attaques ennemies [87].

Cependant, ce type de méthode peut nécessiter beaucoup de travail manuel afin de fournir les règles, contraintes ou même encore des éléments pré-conçus à la main. En effet, une approche constructive intègre implicitement des connaissances en matière de conception. Elle consiste souvent à utiliser un certain nombre d'éléments de contenu préconçus et à les placer les uns à côté des autres de manière aléatoire, tel que des blocs de construction s'imbriquant ensemble [91].

Dans le milieu industriel du jeu vidéo, les méthodes constructives sont très employées dans le genre du *rogue-like* et ceci afin de fournir des niveaux différents à chaque partie.

Nous proposons, ci-dessous, de détailler certaines implémentations d'algorithmes constructifs.

2.3.1.1 Partitionnement de l'espace

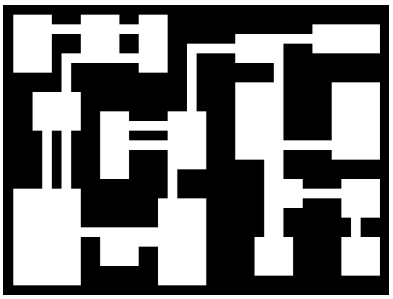


FIGURE 2.4 – Exemple de BSP représentant des salles connectées par des couloirs [6].

Une méthode souvent utilisée dans la génération de donjon car facile à mettre en œuvre, est le *space partitioning*.

Il s'agit d'un procédé de subdivision d'un espace 2D ou 3D, en deux ou plusieurs sous-ensembles disjoints. Tout point de l'espace peut alors être identifié comme appartenant précisément à l'un de ces ensembles. Les systèmes de partitionnement de l'espace sont souvent hiérarchiques, c'est-à-dire que la subdivision est réappliquée récursivement à chaque partie créée. En outre, les parties de l'espace peuvent être organisées dans ce que l'on appelle un *space partitioning tree*.

L'implémentation la plus populaire, le BSP ou *Binary Space Partitioning*, divise récursivement l'espace en deux sous-parties, aussi appelées régions, en alternant entre l'axe vertical et l'axe horizontal pour la découpe. Ainsi, en changeant la position du plan de découpe et son axe, la taille des régions diffère. De ce fait, dans le cas d'une génération de donjon, les salles peuvent être créées dans les limites des régions. De plus, l'espace peut être représenté par un arbre binaire, appelé *BSP tree*. Cet arbre

permet, par exemple, de connecter facilement les différentes régions de façon à relier les salles d'un donjon avec des couloirs, voir Figure 2.4 page précédente.

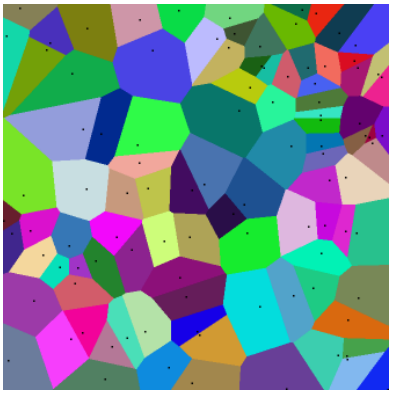


FIGURE 2.5 – Exemple d'un diagramme de *Voronoï* où chaque cellule, surface colorée, représente la zone d'influence d'un germe, point noir [7].

Une méthode alternative ressemblante est le diagramme de Voronoï. Ce dernier découpe l'espace du plan en régions adjacentes en fonction de la distribution d'un ensemble discret de points, voir Figure 2.5. Un exemple d'utilisation de l'algorithme de Voronoï, est la génération de cartes de jeu interactive proposée par Amit Patel [8]. Sa démonstration prend en compte, entre autres, la formation de rivières, l'élévation, l'intégration de plusieurs biomes différents et génère ainsi une carte détaillée, voir Figure 2.6.

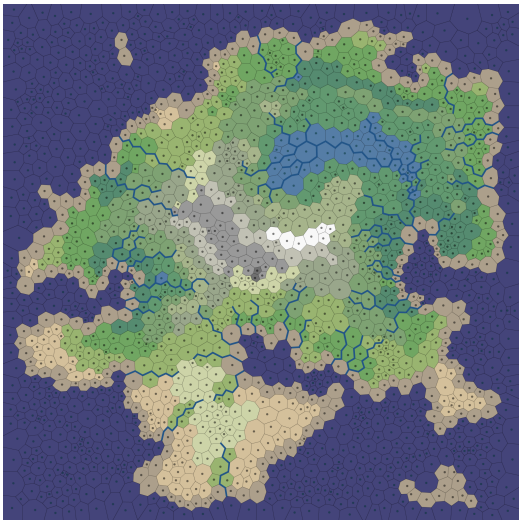


FIGURE 2.6 – Exemple d'une génération de carte proposé par le générateur d'Amit Patel [8].

2.3.1.2 Automate cellulaire

FIGURE 2.7 – Exemple de génération de cave par un automate cellulaire [9].

Modèle découvert dans les années 40, puis développé dans les années 50 principalement par John Von Neumann [92], l'automate cellulaire a été étudié dans de nombreux domaines tels que les mathématiques, la physique, et même la biologie.

Un automate cellulaire est une grille à n dimensions composée de cellules possédant chacune un état. Un état est choisi parmi un ensemble fini, dans le cas le plus simple 1 ou 0, et peut évoluer au cours du temps. L'état d'une cellule varie dans le temps en fonction de son propre état et de celui de ses voisins grâce à un ensemble de règles. Chaque cellule possède des règles identiques. À chaque avancée dans le temps, les mêmes règles sont appliquées simultanément à toutes les cellules de la grille, produisant ainsi une nouvelle étape de génération, dépendante de l'ancienne. Le nombre d'étapes temporelles est défini en amont.

En 1970, le « jeu de la vie » de John Horton Conway constitua une percée significative dans le domaine des automates cellulaires [93]. Son implémentation est un automate cellulaire bidimensionnel où chaque cellule est considérée comme vivante ou morte. L'état futur d'une cellule est déterminé, d'une part par son état actuel, et d'autre part, par le nombre de cellules vivantes parmi les huit qui l'entourent, de la façon suivante :

- Une cellule morte, possédant exactement trois voisines vivantes, devient à son tour vivante.
- Une cellule vivante meurt si elle ne possède pas deux ou trois voisines vivantes.

Cette approche a été expérimentée plusieurs fois dans des générateurs de niveaux. Nous pouvons citer, à ce titre, l'étude de Lawrence Johnson [94] permettant la génération de caves en temps réel, avec trois états : sol, roche et mur, et deux simples règles :

- Une cellule devient roche si au moins cinq de ses voisins sont des roches, dans le cas contraire, elle devient sol.
- Une cellule roche devient un mur si elle possède au moins un voisin à l'état de sol.

Il existe des publications explorant la création de donjons [95], de cavernes [96], de villes [97] et la formation de terrain [98] grâce à des règles similaires.

Implémenter un automate cellulaire peut être rapide et efficace. Cela permet alors une génération en temps réel et la création de niveaux infinis. Cependant, il y a un manque de contrôle direct sur le résultat produit. En effet, grâce à sa nature émergente, il est extrêmement difficile, voire impossible, de garantir par exemple des donjons ou des grottes toujours accessibles par le joueur. La Figure 2.7 page précédente, montre des grottes générées par un automate cellulaire dont certaines ne sont pas connectées entre elles. Des vérifications supplémentaires sont, dans ce cas, nécessaires pour produire un niveau jouable.

2.3.1.3 Marche aléatoire

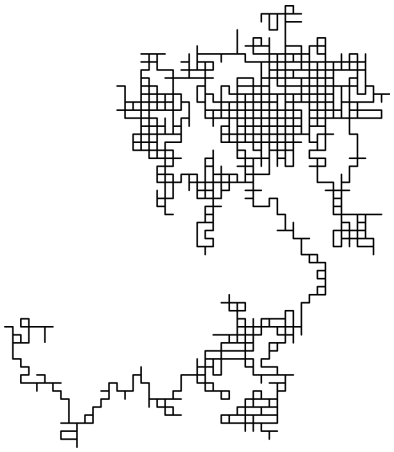


FIGURE 2.8 – *Random walk* en deux dimensions [10].

L'algorithme de la marche aléatoire, ou *Random Walk* en anglais, est également utilisé pour la génération de caves, de chemins ou même de donjons. Il s'agit d'un algorithme utilisable en 2D comme

en 3D. Il garantit la génération d'un niveau entièrement connecté, en offrant une multitude de variations entre chemins étroits et espaces ouverts. La marche aléatoire peut, en effet, générer des caves connectées, et sans sous-parties séparées par un mur, comme peut le produire un automate cellulaire. Cependant, cet algorithme présente un caractère aléatoire et chaotique [99], voir Figure 2.8 page précédente.

Le principe réside dans une succession de pas aléatoires, effectués au hasard. Ces derniers sont décorrélés les uns des autres, c'est-à-dire que l'algorithme ne garde pas en mémoire ses états précédents afin de générer l'étape suivante. Il s'agit ici de sa caractéristique fondamentale, appelée également caractère markovien, du nom du mathématicien Markov.

Le déroulé de cet algorithme, en 2D, s'effectue de la manière suivante :

1. Sélectionner un emplacement de départ aléatoire sur une grille ;
2. Choisir une direction aléatoire de mouvement : Haut, Bas, Gauche, Droite ;
3. Avancer dans cette direction et marquer la position comme visitée, si ce n'était pas déjà le cas de cet emplacement ;
4. Répéter les étapes 2 et 3 jusqu'à ce qu'un nombre souhaité d'emplacements visités aient été placés sur la grille.

2.3.1.4 Combinaison d'éléments pré-conçus ou méthode par templates

La combinaison d'éléments pré-conçus à la main, aussi appelés templates [8], est une approche hybride entre la génération procédurale et les niveaux réalisés manuellement. Cette méthode permet de conférer une impression de cohérence tout en conservant une grande diversité [100]. En effet, cette approche permet de réagencer des éléments différemment à chaque étape du jeu et ainsi d'offrir davantage de diversité à chaque partie. Ces éléments interchangeables représentent des sections d'un niveau, à plusieurs échelles possibles, que ce soit une salle ou un pan entier d'un niveau. Cette technique va utiliser un certain nombre de règles régissant la combinaison et le placement des différentes parties. Néanmoins, cette approche reste limitée par son nombre de résultats possibles, car elle est dépendante du nombre de combinaisons des éléments pré-conçus.

8. Un template peut posséder des parties fixes et des parties aléatoires.

Il existe de nombreux jeux qui s'appuient sur une approche par templates, notamment plusieurs *rogue-like*, tels que : *Spelunky* [101], *Dead Cells* [102], *Dungeon of the Endless* [103], et *Binding of Isaac* [104]. Ces derniers implémentent leur propre variation de cette technique permettant de générer leurs niveaux de jeu, à savoir des donjons.

D'un autre côté, cette méthode permet également de générer une multitude d'autres éléments dans un jeu. Par exemple, le jeu *Borderlands* [48] l'utilise dans la création aléatoire d'attributs pour les armes mis à disposition des joueurs. Le jeu *Spore* [32] l'emploie, quant à lui, dans son outil de création de créatures, en fournissant des éléments sélectionnables par le joueur.

Ce concept est illustré parfaitement par le système de génération de niveaux dans le jeu *Spelunky*. Il s'agit d'un jeu de plateformes en deux dimensions, développé par Derek Yu en 2008 et vendu à plus d'un million de copies en 2016 [105]. Son gameplay consiste à traverser des niveaux en 2D tout en collectant de nombreux objets, éliminant les ennemis, et trouvant une issue. Perdre une partie signifie recommencer le jeu du début dans un tout nouveau niveau généré. Chaque niveau est constitué d'une grille de 4x4 emplacements, donc 16 salles, dont l'une constitue le début et une autre la fin. Des corridors connectent les salles adjacentes, mais certaines peuvent se retrouver isolées du chemin principal reliant le départ à la fin du niveau. Les salles placées dans les emplacements de la grille sont sélectionnées parmi des templates pré-conçus et possédant des sous-parties. Ces dernières sont choisies aléatoirement et permettent de faire varier l'emplacement des pièges et des obstacles, mais aussi de modifier l'aspect esthétique du niveau. Cependant, le placement des salles n'est pas complètement aléatoire, car il doit permettre, au final, un chemin ouvert depuis le début jusqu'à la fin du niveau. La génération est réalisée grâce à l'utilisation de 4 types de salles possibles, voir illustration Figure 2.9 page suivante. Ces types de salles peuvent être détaillés de la façon suivante :

- 0 : Une salle optionnelle non présente sur le chemin.
- 1 : Une salle qui garantit une ouverture à gauche et à droite.
- 2 : Une salle qui garantit une ouverture à gauche, à droite et en dessous. Si une autre salle de type 2 est située au-dessus, une ouverture y est également présente.
- 3 : Une salle qui garantit une ouverture à gauche, à droite et en haut.

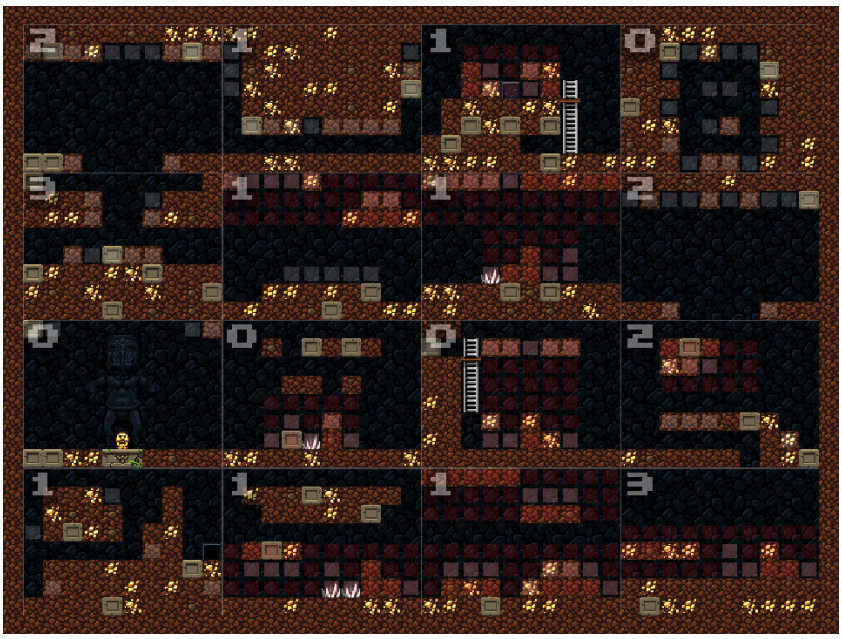


FIGURE 2.9 – *Spelunky Generator* [11], outil de génération de niveaux qui illustre l'algorithme utilisé dans le jeu *Spelunky*. Chaque type de salle est numéroté en haut à gauche.

La génération procédurale entière d'un niveau de *Spelunky* peut être découpée en 3 phases distinctes :

1. Choisir les templates de salles afin de remplir la grille comprenant une entrée et une sortie accessibles ;
2. Générer les obstacles selon des emplacements prédéfinis dans les templates choisis grâce à diverses heuristiques ;
3. Placer les ennemis selon des conditions et des critères de placement.

Une méthode similaire appelée *Occupancy-Regulated Extension* a été utilisée pour la création de niveau de plateformes *Super Mario* [106]. Il s'agit d'un algorithme d'assemblage de géométrie afin de créer un niveau de jeu avec des éléments pré-conçus. L'intervention humaine est possible lors de la conception du niveau.

Une autre approche, basée sur le rythme, a été étudiée également sur un jeu de plateformes 2D [47]. Le niveau, dans ce cas, est interprété comme un morceau de musique, où chaque note représente une petite partie du niveau et permet une sélection de templates correspondants.

2.3.2 Méthode essai-erreur

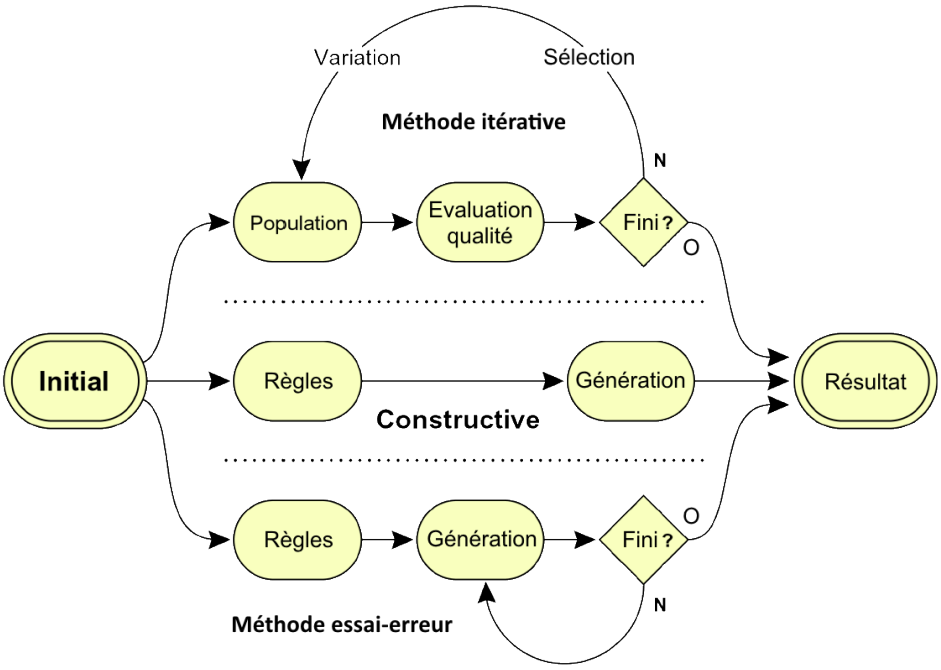


FIGURE 2.10 – Le déroulé de la méthode essai-erreur et de la méthode itérative, par rapport à l'approche constructive [12, Figure 1].

Une seconde famille d'algorithme, différente des méthodes constructives, est appelée approche par essai-erreur, voir l'illustration Figure 2.10. Celle-ci inclut un mécanisme d'évaluation en plus de la partie génération. Chaque résultat produit est testé selon plusieurs critères, par exemple, vérifier la navigabilité d'un niveau entre son entrée et sa sortie. Si le test échoue, le contenu est alors partiellement ou entièrement rejeté, puis regénéré. Le processus est effectué jusqu'à l'obtention d'un résultat satisfaisant.

2.3.2.1 Méthode itérative

La méthode itérative, se référant selon nous à l'approche appelée *Search-Based* en anglais, est un cas spécifique des méthodes essai-erreur, voir Figure 2.10. Elle est basée sur l'utilisation d'algorithmes d'optimisation, le plus souvent un algorithme évolutionnaire [12]. Le principe de base est le suivant : une solution satisfaisante à un problème existe dans un espace de solutions. Il suffit alors d'itérer et de modifier un ou plusieurs paramètres de la solution actuelle jusqu'à aboutir à un résultat satisfaisant.

Les éléments constituants cette méthode itérative sont les suivants :

- Un algorithme de recherche, pièce maîtresse, souvent un algorithme évolutionnaire ;
- Un moyen d'abstraction capable d'encoder le contenu pour qu'il soit manipulable par l'algorithme ;
- Une ou plusieurs fonctions d'évaluation pour indiquer la « qualité » du contenu généré, aussi appelées « fonctions objectifs » ou *fitness function* en anglais.

  Développer une fonction d'évaluation est sans doute la partie la plus difficile à mettre en œuvre, car son but est de noter, de manière fiable, la qualité d'un aspect du contenu généré. Il s'agit, en effet, d'attribuer un score ou une valeur de fitness.

L'approche par optimisation nécessite souvent un temps de calcul conséquent et de ce fait est difficile à appliquer dans la génération en temps réel dans un jeu. De plus, les métaheuristiques [9] tels que les algorithmes évolutionnaires et génétiques ne garantissent pas l'obtention de la meilleure solution possible, mais vont plutôt atteindre un maxima local.

2.3.2.2 Algorithme évolutionnaire

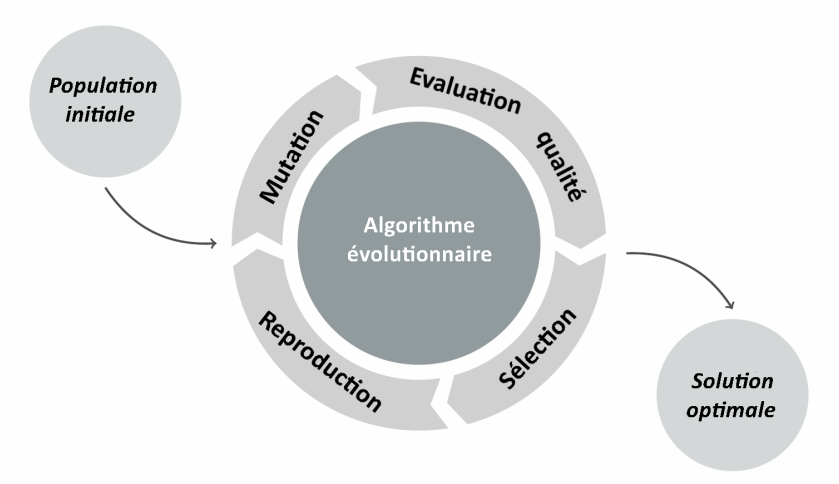


FIGURE 2.11 – La boucle principale d'un algorithme évolutionnaire [13, Figure 3.1].

9. Une métaheuristique est un algorithme d'optimisation.

La caractéristique principale de l'approche évolutionnaire, inspirée par la théorie de la sélection naturelle darwinienne, est l'évolution d'un ensemble de solutions afin d'aboutir aux meilleurs résultats possibles [21, Chapitre 2]. En effet, les algorithmes évolutionnaires font évoluer un ensemble appelé « population » composé de solutions, nommées « individus » ou « chromosomes ». Une telle solution dans l'espace de génération est généralement encodée sous forme de « génotype », utilisé pour une recherche et une évaluation efficientes. Le génotype constitue, en effet, l'information abstraite manipulée par l'algorithme évolutionnaire. Les éléments constituants un génotype peuvent également être appelés « gènes ». Afin d'évaluer un individu, il est nécessaire de construire son « phénotype » ; il s'agit de l'expression des données du génotype. Par exemple, dans la génération de contenu de jeu, le génotype peut être les instructions qui servent à concevoir un niveau de jeu, et le phénotype représente alors le niveau de jeu à proprement parler. Lorsque le phénotype est évalué par la fonction objectif, son individu est associé à un indice de qualité, ou valeur de fitness. Habituellement, plus la valeur de qualité d'un individu est élevée, plus il a de chances de transmettre son génotype à de prochaines générations de population.

Il nous paraît pertinent de noter qu'il est possible d'implémenter des algorithmes constructifs afin de créer les phénotypes des individus à partir des génotypes.

Le déroulé d'un algorithme évolutionnaire peut être illustré selon la Figure 2.11 page précédente, ou être décrit de la façon suivante [107, Figure 3.1] :

1. Construction et évaluation d'une population initiale ;
2. Répéter jusqu'à atteindre un critère d'arrêt :
   (a) Sélection d'une partie de la population qui participera à la reproduction ;
   (b) Reproduction des individus sélectionnés, des croisements sont alors effectués ;
   (c) Mutation de la descendance en modifiant aléatoirement leur génotype ;
   (d) Évaluation du degré d'adaptation de chaque individu ;
   (e) Remplacement de la population actuelle par une nouvelle sélection en choisissant un nombre de parents et d'enfants pour former la génération suivante.

Le critère d'arrêt peut, par exemple, être un nombre maximum d'itérations pour les générations d'individus ou une valeur de qualité atteinte, jugée suffisante. Il existe d'autres paramètres pour manipuler

l'algorithme évolutionnaire tels que la taille de la population, les possibilités de combinaisons, le choix de garder ou non des solutions jugées inadaptées.

De plus, il est important de noter l'utilisation de plusieurs processus aléatoires concernant la construction de la population initiale, la sélection des individus à reproduire, la reproduction en elle-même et la mutation. Les algorithmes évolutionnaires sont alors des algorithmes dits stochastiques.

Dans les algorithmes évolutionnaires, nous pouvons distinguer, entre autres, les algorithmes génétiques. Ces derniers opèrent, plus particulièrement, sur une structure de données de taille fixe et utilisent à la fois la mutation et le croisement pour accomplir la variation [108]. De plus, l'étape de sélection pour le remplacement de la population est souvent proportionnelle à la valeur de fitness. La représentation, ou l'encodage, d'une solution, c'est-à-dire le génotype, correspond généralement à une chaîne binaire, mais peut également être représenté par une chaîne de valeurs discrètes [4, Part. 1 Chap. 2].

2.3.2.3 Fonctions d'évaluation

Il existe plusieurs catégories de fonctions d'évaluation :

- Évaluation directe : le contenu généré est évalué grâce à la construction du phénotype. Il en résulte une valeur de qualité. Il est possible d'utiliser des fonctions d'évaluation, soit pré-établies par un être humain, soit acquises à partir de données existantes. Par exemple, une fois un niveau de jeu *Super Mario* [109] construit, l'évaluation d'un niveau de difficulté est possible en prenant en compte le nombre d'espaces fatals de vide entre les plateformes.

- Évaluation basée sur une simulation : utilisation d'agents, ou intelligences artificielles, abrégé IA, qui vont « jouer » le contenu généré pour en estimer la qualité. Il est alors possible d'utiliser les statistiques de jeu des divers agents pour établir un score, par exemple, en prenant en compte le nombre de sauts effectués, les ennemis vaincus et le temps restant dans un niveau *Super Mario*. Les agents peuvent également imiter des comportements humains selon des préférences d'objectifs, tels que par exemple : *killers*, *achievers*, *explorers*. Il s'agit ici de profils possibles, utilisés pour classifier les types de joueurs, et détaillés par Richard Bartle dans son étude [110].

- Évaluation interactive : un ou plusieurs joueurs humains vont interagir avec le contenu généré, et ainsi permettre l'obtention de statistiques de jeu et/ou des retours utilisateurs grâce à des

questionnaires d'évaluation personnels. Par exemple, grâce à l'utilisation de ce genre de processus dit *human in the loop*, il est possible de récupérer implicitement le nombre de morts, de sauts et le temps pour un niveau parcouru de type *Super Mario* et ensuite d'obtenir une appréciation générale donnée explicitement par le joueur. Cette technique d'évaluation a été utilisée, par exemple, pour des niveaux de jeu de tir à la première personne dont la génération s'adapte aux préférences des joueurs [111]. Ce procédé a été employé également dans les recherches [57, 70, 112, 113].

2.3.2.4 Évaluation par simulation

Il nous semblait pertinent d'approfondir cette catégorie précise qu'est l'évaluation par simulation. En effet, l'utilisation d'agents autonomes nous paraît prometteuse afin d'évaluer et de noter des niveaux générés dans le but d'obtenir un score reflétant une expérience joueur pertinente.

L'algorithme utilisé pour la simulation se doit de déterminer quelles actions doivent effectuer les agents IA, qui vont « jouer » au niveau candidat lors de la simulation, et ceci en fonction de l'état du jeu à chaque étape. Différents objectifs peuvent être définis pour les agents et influenceront alors leur prise de décision, comme gagner le jeu ou finir le niveau en restant en vie. D'autres buts comme finir le niveau le plus rapidement possible ou éviter les dégâts sont également envisageables et vont dépendre du profil de l'agent, ce qui n'est pas sans rappeler la classification émise par Richard Bartle [110]. Plusieurs études se sont penchées sur cette utilisation d'agents ayant des objectifs différents, nommés personas afin d'évaluer et de tester notamment la faisabilité de niveaux de donjons [114, 115, 116].

Afin de développer des agents, nous pouvons citer une méthode d'apprentissage appelée le *Monte Carlo Tree Search*, dit *Single Objective*. Elle est utilisée pour la prise de décision en explorant un arbre des possibles selon un objectif précis à atteindre [4, Chapitre 2].

Nous pouvons également combiner plusieurs buts différents, cependant il peut arriver qu'un conflit émerge entre ces objectifs. Afin de surmonter ce problème, il existe des concepts mathématiques permettant de trouver la meilleure balance possible pour plusieurs variables à la fois. Par exemple, le *Pareto Front*, offre la possibilité de sélectionner un ensemble de solutions optimales prenant en compte tous les critères [117].

2.3.2.5 Exemples de cas utilisant une méthode itérative

Bien qu'il existe de nombreuses applications de cette approche itérative de génération de contenu, nous pouvons en citer néanmoins quelques-unes qui nous semblent particulièrement pertinentes et qui montrent le potentiel d'une telle méthode.

Premièrement, le projet *Galactic Arms Race* est un jeu qui implémente un algorithme évolutionnaire, permettant de varier les systèmes de particules liées aux armes [57]. Il s'agit d'une utilisation « en ligne » qui est mise en place en vue de faire évoluer les armes du joueur au cours du jeu.

En ce qui concerne les jeux de plateformes, le *Search-Based* a été appliqué, par exemple, pour générer de nouveaux résultats créés à partir de segments, extraits de niveaux existants [118]. Une méthode similaire peut être retrouvée dans l'article de Valtchan Valtchanov [119] qui génère, dans son cas, des niveaux de type *dungeon crawler*. L'encodage de la solution se fait alors sous forme de structure d'arbre : un nœud représente alors les parties du niveau, le plus souvent une salle, et les arêtes, les connections entre les salles.

Une autre étude a permis de produire des labyrinthes à l'aide de diverses représentations d'encodage [120]. Cette recherche démontre également que les résultats générés avec différentes fonctions d'évaluation sont sensiblement différents les uns des autres.

Par ailleurs, nous pouvons mentionner la génération de terrain pour un jeu de type *RTS*, jeu de stratégie en temps réel, appelé *StarCraft* [121]. En effet, en 2010, Julian Togelius [14] a démontré que le *Search-Based* pouvait générer des cartes jouables multijoueurs grâce à plusieurs fonctions d'évaluation. Le génotype de la carte est ici encodé sous forme de tableau de coordonnées pour quatre types d'éléments : les bases [10], les ressources minérales, les puits de gaz et les zones infranchissables, c'est-à-dire les roches et l'eau. Les fonctions d'évaluation sont, quant à elles, principalement axées sur la vérification de chemins praticables pour les déplacements entre les bases et les différentes ressources. Le résultat est une carte entièrement jouable, voir Figure 2.12 page suivante.

10. Les bases dans *Starcraft* représentent les campements des joueurs et définissent le point de départ d'une partie sur la carte.

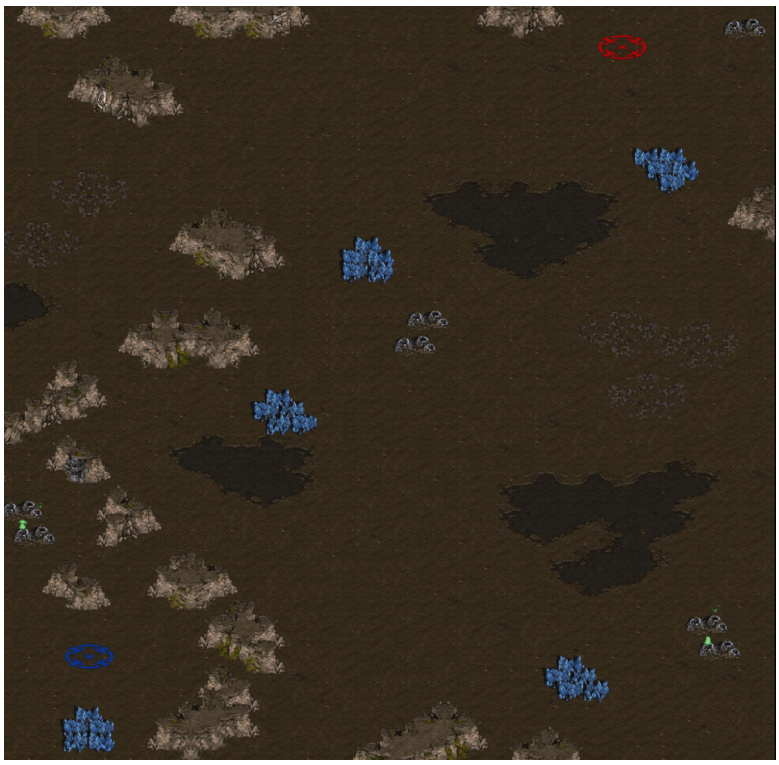


FIGURE 2.12 – Exemple de carte générée pour *StarCraft* par Julian Togelius [14, Figure 2].

2.3.2.6 Méthodes d'optimisation sous contrainte

La recherche évolutionnaire ou génétique, a très tôt démontré sa pertinence pour l'optimisation numérique dans une multitude de domaines, notamment dans la génération de contenu procédural par la méthode *Search-Based* [12]. Cependant, lorsque l'espace de recherche est contraint d'une manière ou d'une autre, une recherche évolutionnaire est souvent confrontée à d'importants défis. En effet, lors de la conception de contenu, la distinction entre solutions faisables et infaisables peut être inévitable. Une solution faisable peut représenter un niveau réalisable par le joueur, c'est-à-dire lui donner la possibilité de finir son parcours dans un niveau et d'atteindre un objectif. Comment traiter alors, au mieux, les individus infaisables dans un algorithme de recherche stochastique avec fonction d'évaluation ? Cette problématique est devenue un sujet de recherche qui a interpellé de nombreux scientifiques. De ce fait, de nombreuses études ont conduit vers une grande variété de méthodes d'optimisation sous contrainte, par exemple [122, 123, 124, 125]. Nous pouvons retrouver plusieurs compte-rendus résumant diverses approches, rédigés, entre autres, par Zbigniew Michalewicz en 1995 [126, 127] et plus récemment par Efrén Mezura-Montes en 2011 [128].

Nous avons choisi de présenter, succinctement, trois approches permettant d'appréhender ce problème de contraintes :

Pénalisation Nous pouvons pénaliser la fonction de fitness de l'algorithme de recherche à chaque fois qu'une contrainte est enfreinte. Cependant, une telle pénalité peut être difficile à concevoir. En effet, si nous la sanctionnons trop, nous risquons de perdre de bonnes caractéristiques composant l'individu, car il serait, dans ce cas, éliminé.

Deux populations Afin d'éviter d'équilibrer une fonction entre le score de l'évaluation et la satisfaction des contraintes, une autre possibilité consiste à placer les niveaux ne respectant pas les contraintes dans une population différente. Il s'agit de les faire évoluer d'une autre manière, et ceci uniquement afin de corriger les erreurs liées aux contraintes.

Nous pouvons citer l'algorithme *FI-2Pop* qui reprend ce concept [125]. En effet, il maintient deux populations d'individus, la faisable et l'infaisable. Tout individu peut passer d'une population à l'autre, s'il respecte ou non les contraintes. Il évolue ainsi, dans le but, soit de maximiser une fonction de fitness, soit de diminuer la quantité de contraintes transgressées.

Un exemple d'application, inspiré du *FI-2Pop*, a été proposé par Antonios Liapis dans le cadre de la recherche de nouveauté [78]. Celle-ci est une alternative à la recherche par objectif, et présente un potentiel dans les domaines où une fonction de fitness est difficile à quantifier. La recherche par nouveauté évolue vers la diversification des solutions d'une population, au lieu de maximiser une fonction objectif, approximant la qualité d'une solution. Cet algorithme sélectionne un individu en fonction de son score de nouveauté, qui représente la distance moyenne entre l'individu et ses plus proches voisins.

Opérateur de réparation Un autre moyen consiste à utiliser des étapes spécifiques de l'algorithme pour corriger les erreurs d'une solution donnée, appelé opérateur de réparation. Ces opérateurs ne sont pas, en général, des solutions génériques. De ce fait, ils doivent être conçus spécifiquement pour chaque problème d'optimisation, et peuvent être coûteux en termes de temps de calcul lors de la réparation des individus.

2.3.3 Programmation par contraintes

Les méthodes de satisfaction de contraintes sont des approches déclaratives qui tentent de définir les propriétés désirées d'une solution et non de décrire les étapes pour y parvenir. Certains algorithmes prennent en compte des contraintes qui peuvent être soit fortes, c'est-à-dire requises, soit optionnelles. Un solveur de contraintes est alors utilisé pour trouver toutes les solutions potentielles qui satisferont les contraintes spécifiées.

Cette approche présente l'avantage suivant : une fonction objectif ou fonction d'évaluation, n'est pas nécessaire pour guider l'algorithme. En effet, les solutions non valides sont systématiquement rejetées afin de répondre aux contraintes initiales. Néanmoins, la difficulté de cette approche réside dans le fait de devoir connaître suffisamment les spécificités du résultat recherché.

Avec la programmation par contraintes, il est possible notamment de proposer des outils de co-création entre le développeur et la machine. En effet, en 2010, Gillian Smith a travaillé sur un tel outil, *Tanagra* [45]. Ce dernier permet à un être humain de concevoir un niveau de plateformes 2D en éditant le rythme du jeu.

2.3.3.1 Wave Function Collapse

L'algorithme *Wave Function Collapse*, abrégé WFC, est un algorithme de génération procédurale basé sur la propagation de contraintes locales d'adjacences, et guidé par des données, ou *data driven* en anglais. Il a été introduit par Maxim Gumin en 2016, afin de reproduire des textures en 2D [15], voir exemple Figure 2.13 page suivante. Ce développeur s'est inspiré, notamment, des travaux de Paul Merrell [129, 130].

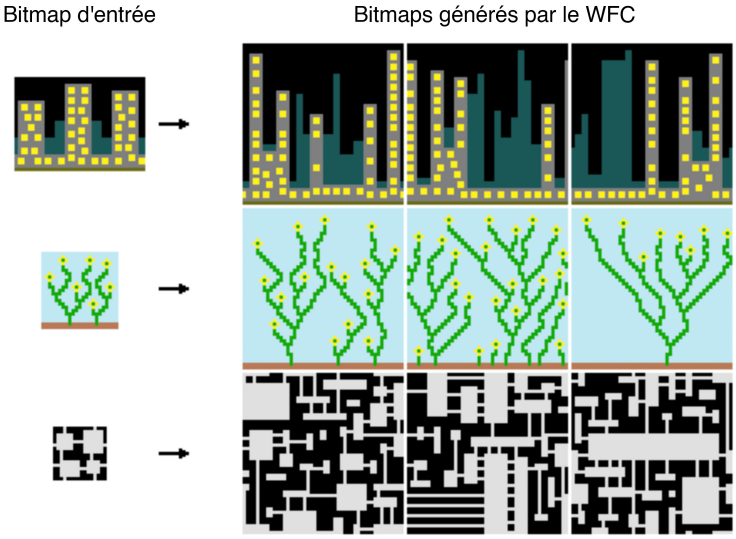


FIGURE 2.13 – Exemples de reproductions de textures en 2D grâce au WFC [15].

Dans le cas d'une représentation par grille pour l'algorithme de WFC, chaque cellule est définie comme un emplacement pour un module. Un module possède ses propres contraintes d'adjacences par rapport à ses voisins, par exemple sous forme de listes. Une rotation peut également être appliquée au module, toujours selon les contraintes d'adjacences. Il peut, de plus, détenir les informations d'un modèle à afficher, par exemple un objet 3D ou une texture 2D. Dans un premier temps, chaque emplacement se voit attribuer tous les modules possibles. Lorsqu'un emplacement voit ses possibilités réduites, il va influencer ses voisins adjacents et de ce fait propager les contraintes par rapport à ses propres modules restants. Ainsi, les emplacements vont peu à peu réduire leur choix de modules sur l'ensemble de la grille.

Le WFC peut être appliqué selon deux approches distinctes, le *Overlap Model* et *Simple Tiled Model* :

- Le *Overlap Model* se sert de patterns, souvent à partir d'une image initiale qu'il découpe en sous-parties afin de déduire les contraintes liant chaque voisin limitrophe, voir Figure 2.14 page suivante.
- Le *Simple Tiled Model* va, quant à lui, employer une liste de contraintes définie, restreignant les possibilités de voisin pour chaque module.

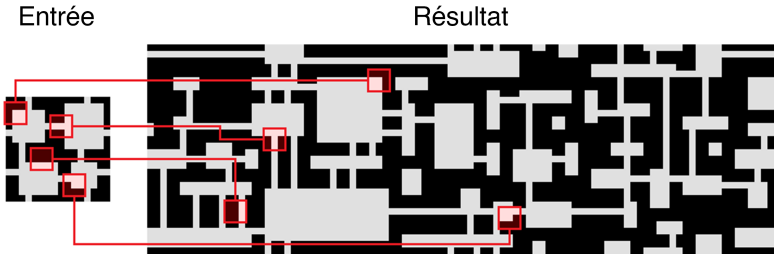


FIGURE 2.14 – Exemple d'art procédural en 2D, utilisant le *Overlap Model* de Maxim Gumin. Nous pouvons observer ici la position des différentes sous-parties réutilisées dans l'image finale [16].

Le WFC peut être utilisé en 2D, ainsi qu'en 3D. Le nombre de dimensions et la forme de la grille employée peuvent faire varier le nombre de voisins possibles par emplacement. Par exemple, avec une grille carrée en 2D, il est possible de choisir quatre emplacements voisins.

Le déroulé général d'un algorithme de WFC pour le *Simple Tiled Model*, basé sur la version de Maxim Gumin [15] et de Tobias Nordvig Møller [19], peut se décrire de la façon suivante :

1. Déclarer un tableau selon les dimensions du résultat de sortie, représenté par une grille. Chaque index du tableau représente un emplacement de la grille. Chaque emplacement possède la liste de ses modules possibles ; cette liste sera appelée l'espace des possibles.

2. Initialiser la grille avec tous les modules ajoutés dans chaque espace des possibles, associés à chaque emplacement.

3. Répéter les étapes suivantes :

    a) Observation

        i) Trouver l'emplacement avec l'entropie [11] minimale, c'est-à-dire l'emplacement avec l'espace des possibles le plus réduit. Si plusieurs emplacements sont en lice, l'un sera choisi au hasard.

        Si tous les emplacements n'ont plus qu'un seul ou zéro module restant, alors arrêter la boucle et aller à l'étape 4.

        ii) Réduire au hasard l'espace des possibles de l'emplacement sélectionné à un seul module, en supprimant les autres modules disponibles.

11. L'entropie est une mesure de l'incertitude et du désordre, et peut se calculer selon la formule de Shannon : $-\sum_{i=1}^{n} p_i \log(p_i)$ [131].

b) Propagation

L'information de la réduction précédente est propagée à travers toute la grille. Chaque emplacement actualise son espace des possibles, c'est-à-dire sa liste des modules restants, toujours selon les contraintes de ses voisins. Plusieurs itérations peuvent être nécessaires pour permettre la propagation sur toute la grille. Cette étape continue aussi longtemps qu'un emplacement modifie son espace des possibles. De ce fait, la propagation peut être une longue étape à calculer, car elle est dépendante de la taille de la grille, des modules possibles et du nombre de contraintes par module.

4. Tous les emplacements ont à présent leur espace des possibles définis à un ou zéro module restant. S'il n'y a aucun choix possible pour l'un des emplacements, cela signifie que l'algorithme a rencontré une contradiction par rapport aux contraintes définies. Une contradiction survient lorsque l'algorithme se retrouve dans une impasse où les contraintes réduisent à zéro le choix de module pour un emplacement. Dans le cas contraire, la génération est terminée, et le résultat est retourné.

Dans le cas d'une contradiction, il est possible d'implémenter une solution dite de « retour sur trace », ou *backtracking* en anglais, où la réduction qui conduit à l'erreur est annulée et une autre observation est tentée avec un choix différent. L'algorithme doit alors garder en mémoire ses décisions précédentes.

De plus, il est envisageable d'intégrer une notion de fréquence de probabilité pour l'apparition d'un module, c'est-à-dire une contrainte non locale ou non spatiale, ce qui permettrait d'éviter de choisir aléatoirement le module sélectionné lors de la réduction de l'espace des possibles. La probabilité pourrait être, soit spécifiée en amont, soit être ajustée au fur et à mesure de la génération afin de limiter l'apparition du même module. Il serait également possible de spécifier cette fréquence selon l'image d'entrée. Cela permettrait alors de reproduire un résultat similaire en termes du nombre de présences de chaque module.

Le WFC a le mérite de proposer une approche flexible et modulaire. En effet, il peut s'adapter en tant qu'outil d'aide à la création, par exemple, en permettant l'utilisateur de définir manuellement des modules dans certains emplacements et de laisser l'algorithme compléter le résultat.

En dernier lieu, il nous semble intéressant de citer les différents articles web suivants, qui détaillent

le fonctionnement du WFC plus précisément et présentent diverses implémentations de l'algorithme [132, 133, 16, 134].

2.3.3.2 Utilisations existantes du WFC

L'algorithme du WFC a été initialement développé par Maxim Gumin en 2016 [15] pour générer de l'art procédural en 2D à partir d'une image. Cette approche a rapidement suscité un grand intérêt grâce, notamment, à sa simplicité de mise en œuvre et de compréhension, tout en générant des résultats de qualité sans configuration complexe [135]. Ce concept s'est propagé sur les réseaux sociaux avec diverses images et animations [136]. Le WFC a rapidement été adapté afin de répondre à d'autres besoins, par exemple la génération en 3D [137]. Il a, de plus, été réécrit dans différents langages de programmation.

Nous allons, ci-dessous, présenter divers jeux et projets utilisant le WFC. Puis, nous nous intéresserons aux différentes études réalisées dans le domaine de la recherche.

Jeux et prototypes L'utilisation de cet algorithme a permis l'élaboration de nombreux jeux et prototypes.

Joseph Parker a, par exemple, proposé une version pour le moteur de jeu *Unity* sous forme d'outil [138]. Ce développeur a également conçu lors de la *ProcJam 2016*, avec Ryan Jones et Oscar Morante, le premier jeu *Pro Skater 2016* ayant comme générateur de niveau, le *Wave Function Collapse* [139].

Nous pouvons retrouver plusieurs implémentations dans le domaine du jeu vidéo, notamment avec des projets sur la console *Pico 8* [140]. Par exemple, Andy Wallace, avec son jeu de game jam *Maureen's Chaotic Dungeon* [141], implémente le WFC afin de régénérer les parties du niveau détruites.

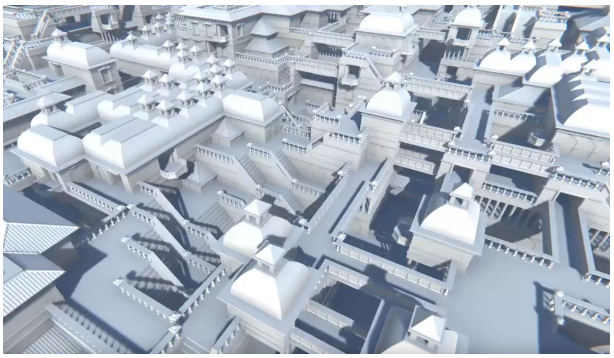
Concernant les prototypes, nous pouvons citer Marian Kleineberg et sa ville construite à l'infini en temps réel [142], voir Figure 2.15b page suivante, et Oskar Stålberg avec de nombreux exemples sur son compte Twitter [143], et sa démonstration interactive 2D [144].

En ce qui concerne les jeux commerciaux, nous pouvons citer *Bad North* [145], voir Figure 2.15a page suivante, développé entre autres par Oskar Stålberg qui implémente le WFC afin de générer des niveaux sous forme de petites îles. Ce concepteur a également publié un outil interactif de création de villes avec le WFC, *Townscaper* [17], voir Figure 2.16 page suivante. Le jeu *Caves of Qud* de

Brian Bucklew [146], utilise aussi ce concept dans la génération des niveaux avec plusieurs passes de génération [136].

D'un autre côté, cet algorithme peut s'étendre à bien d'autres domaines, par exemple pour produire de la poésie, comme le montre Martin O'Leary [147].

(a) *Bad North*, jeu commercialisé [145]. Génération procédurale de petites îles à l'aide du WFC.



(b) *Infinite City*, prototype [142]. Génération procédurale d'une ville à l'infini à l'aide du WFC.

FIGURE 2.15 – Exemples d'applications du WFC.

FIGURE 2.16 – Jeu *Townscaper*, un outil interactif de création de villes, du *mixed-initiative* avec le *Wave Function Collapse* [17].

Domaine de la recherche Bien qu'il n'existe pas de papier publié directement par Maxim Gumin, Isaac Karth et Adam M. Smith ont analysé et détaillé cet algorithme, ainsi que quelques applications déjà réalisées en 2017 [136]. Par ailleurs, de nombreux autres chercheurs se sont également penchés sur diverses applications du WFC.

Le WFC a ainsi été abordé afin de répondre, entre autres, aux problématiques liées à la conception et le peuplement de villes virtuelles. Les articles suivants présentent des recherches menées sur la génération procédurale de villes dans les jeux vidéo [148, 149], et la synthèse d'espace urbain [150].

D'autre part, Adam Newgas présente *Tessera* [135], un outil et une interface utilisateur, pour le moteur de jeu *Unity*. Il permet la configuration et l'exécution de génération procédurale, basée sur des contraintes, avec des extensions utiles à l'algorithme de base du WFC.

Par ailleurs, il semble intéressant de mentionner l'article de Darui Cheng [151]. Celui-ci présente un système de règles automatiques, simplifiant leurs écritures. Il introduit également diverses fonctionnalités supplémentaires telles que, par exemple, la génération multicouche et un principe de contrainte non locale selon une distance.

Quentin Edward Morris, quant à lui, a modifié le WFC avec des contraintes supplémentaires, non locales également [18]. Il voulait permettre à un concepteur de jeux d'avoir plus de contrôle sur l'algorithme afin de produire des résultats plus variés et plus spécifiques. Pour ce faire, il s'est basé sur le jeu *Super Mario Bros.* [109] dans le cadre de la génération de niveaux de jeu, voir exemple Figure 2.17. Afin d'évaluer la qualité de ses résultats, il s'est inspiré d'un ensemble de métriques, proposé par Gillian Smith et Jim Whitehead [75], permettant d'examiner la capacité de diversité d'un algorithme.

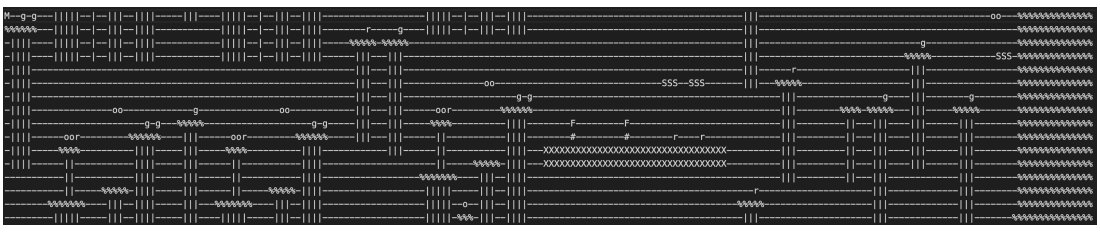


FIGURE 2.17 – Exemple de génération obtenue par Quentin Edward Morris [18]. La représentation textuelle du niveau 3 de l'original de *Super Mario Bros.* a servi comme image d'entrée pour le WFC.

Dans le cadre des rajouts de fonctionnalités, Arunpreet Sandhu a, de son côté, exploré le WFC en tant que solveur de satisfaction de contraintes afin d'intégrer des principes et des pratiques de

design [152]. Il a ainsi modifié le WFC en rajoutant, par exemple, des contraintes non spatiales, c'est-à-dire non locales, et en recalculant des poids. Son article décrit une mise en œuvre technique de l'intégration de contraintes de conception dans le WFC et analyse les compromis liés au temps de calcul et à la mémoire utilisée.

Pour finir, nous pouvons présenter la recherche de Jonas Aksel Billeskov et Tobias Nordvig qui combine le WFC et le principe de *Growing Grids*[12] pour créer un générateur de contenu procédural amélioré [19]. Les niveaux ont été évalués selon la difficulté de navigation des cartes produites. Contrairement à des résultats organisés sur une grille régulière, carrée ou rectangulaire, la proposition ici permet de créer une plus grande diversité de résultats. En effet, un niveau peut être généré, par exemple, sur la base de la grille Figure 2.18.

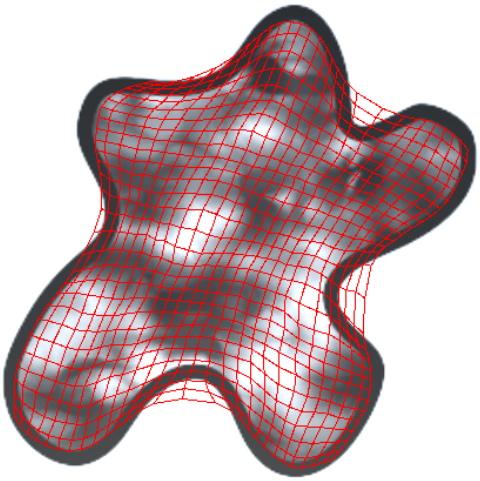


FIGURE 2.18 – Exemple d'une *Growing Grid*, selon la forme en arrière-plan et son gradient, comme paramètres [19].

2.3.4 Grammaires génératives

Les grammaires formelles ont été originellement introduites dans les années 1950 par le linguiste Noam Chomsky dans le but de décrire le langage naturel [154]. Elles sont également appelées « grammaires génératives », car elles décrivent des structures linguistiques et détaillent la manière de les générer : nous pouvons utiliser une grammaire générative pour produire de nouvelles phrases repre-

12. Le concept de *Growing Grid* a été introduit par Bernd Fritzke en 1995 [153]. L'algorithme prend une image binaire 2D comme entrée, puis agence une grille de manière à couvrir au mieux la forme de cette image.

nant les structures décrites [21, Chapitre 3]. Le concept original permet la transformation de chaînes de caractères et se décrit de la manière suivante :

- Un alphabet et un ensemble de règles de production sont nécessaires pour définir une grammaire.
- Une règle décrit quel symbole de l'alphabet sera remplacé par un ou plusieurs nouveaux symboles.
- Chaque règle est de la forme (symbole(s)) → (autre(s) symbole(s)) où la partie de gauche sera remplacée par la partie de droite.
- Les règles sont appliquées itérativement jusqu'à détection d'une condition d'arrêt ou lorsque aucun symbole ne peut être réécrit.

Voici un exemple comprenant deux règles de production :

1. A → AB
2. B → b

Une convention stipule que les caractères en majuscule représentent les symboles non terminaux, situés à gauche des règles et donc réécrits. D'un autre côté, les caractères en minuscule désignent les symboles terminaux et ne sont donc pas réécrits. Cependant, une grammaire peut ne pas utiliser ces restrictions.

On peut distinguer deux types de règles dites « hors contexte » et « contextuelle » :

- Une règle hors contexte s'écrira A → AB, car seul un symbole est réécrit.
- Une règle contextuelle s'écrira xAx → xyx car le symbole A ne sera réécrit que s'il est entouré par deux symboles x.

Une autre distinction est faite entre les grammaires déterministes et celles non déterministes. Les grammaires déterministes possèdent uniquement une règle applicable à chaque symbole ou séquence de symboles, de sorte que, pour une chaîne de caractères donnée, les règles à utiliser pour la réécrire ne comportent aucune ambiguïté. En revanche, dans les grammaires non déterministes, plusieurs règles peuvent s'appliquer à une chaîne donnée, produisant alors différents résultats possibles à chaque étape de réécriture. Dans ce cas, le choix entre les règles peut se faire, par exemple, de manière probabiliste.

Les règles de réécriture peuvent être utilisées pour analyser et décrire des structures existantes en les décomposant en un arbre de dérivation. En effet, il existe deux manières de décrire comment une chaîne de caractère a été produite à partir d'une grammaire donnée ; la première consiste à énumérer la suite des applications des règles, la deuxième reprend cette liste en tant qu'arbre de dérivation. Celui-ci peut être également généré en débutant avec un symbole initial et par applications itératives des règles de productions. Dans ce cas, le résultat est alors soit un arbre ou soit une chaîne de caractères, représentant le contenu.

Un arbre de dérivation peut facilement être représenté avec une grammaire non contextuelle, c'est-à-dire une grammaire composée uniquement de règles hors contexte, car chaque nouvelle partie générée peut être perçue comme un élément séparé. La Figure 2.19 illustre un arbre de dérivation appliqué à un donjon.

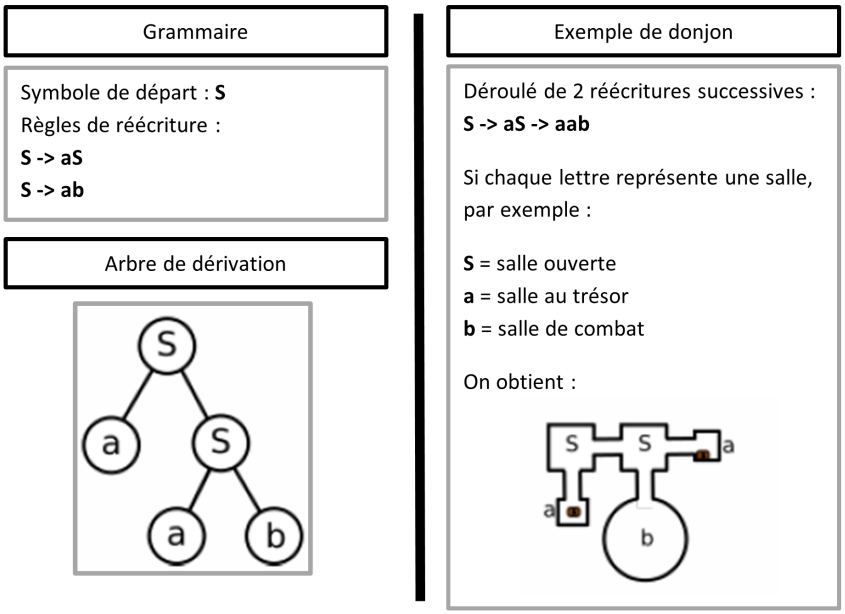


FIGURE 2.19 – Exemple d'un arbre de dérivation extrait d'une simple grammaire, à gauche, ainsi qu'un donjon généré à partir de cet arbre, à droite [20, Figure 2.3].

Une grammaire peut également se caractériser selon l'ordre d'application de ses règles de réécriture. De manière séquentielle, la chaîne de caractères est lue de gauche à droite et est réécrite au fur et à mesure qu'une règle s'applique à un symbole. D'autre part, en appliquant l'ordre parallèle, toutes les règles sont appliquées simultanément à chaque symbole. Il est intéressant de noter que le choix

de l'ordre peut entrainer des différences majeures sur le résultat final. Un exemple connu utilisant la réécriture parallèle est le *L-System* que nous allons aborder dans la section suivante.

2.3.4.1 L-System

Le *L-System* a été introduit par le biologiste Aristid Lindenmayer en 1968 afin de modéliser l'évolution cellulaire et la croissance des plantes [39]. Cette variation des grammaires formelles a comme caractéristique principale, la réécriture parallèle.

Par exemple, pour représenter la croissance de la levure, Aristid Lindenmayer a décrit un *L-System* avec les règles suivantes :

- A $\rightarrow$ AB
- B $\rightarrow$ A

En commençant par l'axiome A ou chaîne de caractère initiale, les premières expansions de la grammaire seront :

1. A
2. AB
3. ABA
4. ABAAB
5. ABAABABA

Quelques années plus tard, Przemyslaw Prusinkiewicz a généré de la végétation procéduralement en employant ce *L-System* [155]. Il encode des instructions graphiques supplémentaires dans la chaîne de caractère produite par cette grammaire afin de dessiner une plante, voir l'exemple Figure 2.20 page suivante.

Un *L-System* est facile à définir en théorie, car une seule règle peut être suffisante. Cependant, en pratique, les résultats sont difficiles à prévoir et de nombreux essais peuvent être nécessaires afin d'obtenir le résultat souhaité. De plus, l'un des inconvénients de cette grammaire est sa limitation à la génération de formes géométriques qui possèdent des propriétés fractales. Ceci explique son usage fréquent dans le domaine de la botanique.

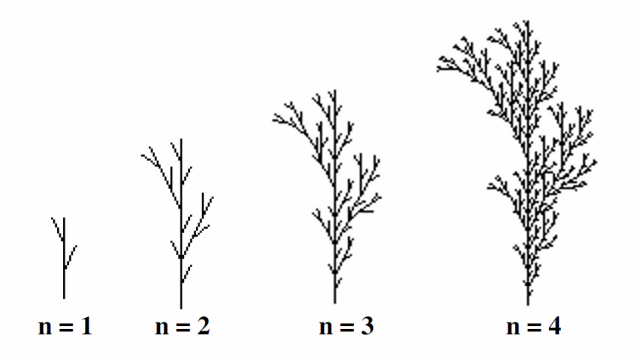


FIGURE 2.20 – Exemple d'interprétation graphique d'un *L-System* [21, Chapitre 5, Figure 5.3].

2.3.4.2 Missions et espaces de jeu

Les grammaires génératives peuvent être utilisées pour décrire des jeux où l'alphabet va consister en un ensemble de symboles, représentant les concepts spécifiques dudit jeu. Les règles définissent alors les façons dont ces concepts peuvent être combinés pour former un niveau du jeu.

Par exemple, comme l'a proposé Joris Dormans dans un de ses exemples [156], un donjon basique peut être conceptualisé en utilisant une grammaire avec les règles suivantes :

1. Donjon → Obstacle + trésor
2. Obstacle → clef + Obstacle + verrou + Obstacle
3. Obstacle → monstre + Obstacle
4. Obstacle → salle

Les symboles commençant par une majuscule sont considérés comme non terminaux et seront remplacés ; à l'inverse, ceux débutant avec des minuscules sont terminaux. Ci-dessous, nous trouvons des résultats possibles de l'application de ces règles :

- clef + monstre + salle + verrou + monstre + salle + trésor
- clef + monstre + clef + salle + verrou + monstre + salle + verrou + salle + trésor
- salle + trésor
- monstre + monstre + monstre + monstre + salle + trésor

Néanmoins, cette grammaire de donjon basique ne peut fournir que des niveaux linéaires de part leur cheminement. Plusieurs chercheurs se sont déjà penchés sur l'application de deux autres types de grammaires, basés sur la grammaire formelle définit par Noam Chomsky, afin de générer des donjons plus complexes et plus proches des niveaux créés par des *level designers* : la « grammaire de graphes » [157] et la « grammaire de formes » [158]. Ces grammaires diffèrent par leur type de symboles employés : des nœuds et des arêtes pour les grammaires de graphes et des formes géométriques, en 2 ou 3 dimensions, pour les grammaires de formes.

En 2002, David Adams proposa une génération de niveaux de jeu de type FPS s'apparentant à des donjons [159]. Grâce à l'utilisation de grammaires de graphes, il dépeint l'agencement topologique général du niveau, puis y place divers objets et ennemis. Les nœuds symbolisent les salles, et celles-ci sont adjacentes lorsqu'elles sont connectées par une arête. En plus de cette représentation topologique abstraite, David Adams a expérimenté divers paramètres de génération tels que la difficulté, une valeur de fun ou la taille du graphe. Il est également intéressant de noter que sa génération procédurale comporte un algorithme de recherche et d'évaluation. Ce dernier essaie de respecter les paramètres d'entrée, en analysant le résultat de chaque application de règles de réécriture. Cela permet de choisir, par la suite, la règle la mieux adaptée aux critères spécifiés. Lors de ses recherches, David Adams a observé que l'utilisation de grammaires de graphes dites non contextuelles sont restrictives en termes de possibilités de génération de graphes. En effet, il n'a pas réussi à produire certaines structures de niveaux de jeu souhaités. Il a, de ce fait, privilégié les grammaires de graphes contextuelles afin de produire des résultats non triviaux. Cependant, le système de génération procédurale de David Adams a été développé afin de répondre à un besoin spécifique et comporte des paramètres propres à sa génération de niveau. Par conséquent, appliquer cette représentation à différentes catégories de jeu, nécessiterait le développement de nouveaux paramètres et règles adaptés à chaque cas. Malgré cela, les travaux de David Adams montrent une approche intéressante visant à décrire la topologie d'un donjon.

En 2010, Joris Dormans a poursuivi dans cette voie et a développé un système similaire à celui de David Adams [156]. Dans son approche, un niveau de jeu est considéré comme la combinaison de deux structures interagissantes mutuellement : une mission et un espace de jeu. Une mission décrit les actions que peut ou doit entreprendre un joueur pour compléter le niveau. Un espace de jeu décrit l'environnement physique géométrique du niveau. Joris Dormans commence par générer une mission

pour le joueur avec l'utilisation de grammaires de graphes, puis créé l'espace de jeu, c'est-à-dire le donjon, adéquat grâce aux grammaires de formes. D'après lui, une grammaire de graphes convient parfaitement à la génération de missions, car ces dernières s'expriment mieux sous la forme de graphes non linéaires, ce qui, pour les jeux d'exploration, est préférable par rapport aux structures linéaires. Pour ce faire, il génère donc la mission sous la forme d'un graphe orienté, qui modélise les séquences de tâches à accomplir par le joueur. La Figure 2.21 illustre ce concept, avec une mission sous forme de graphe orienté et son adaptation en espace physique de jeu. La traduction du graphe est effectuée grâce à l'utilisation de grammaires de formes. Le schéma, voir Figure 2.22 page suivante, montre un exemple d'une grammaire de forme où l'alphabet est constitué de trois symboles : une « connexion », un « espace ouvert » et un « mur ». Seule la « connexion » est un symbole non-terminal et possède un marqueur carré avec un triangle indiquant son orientation.

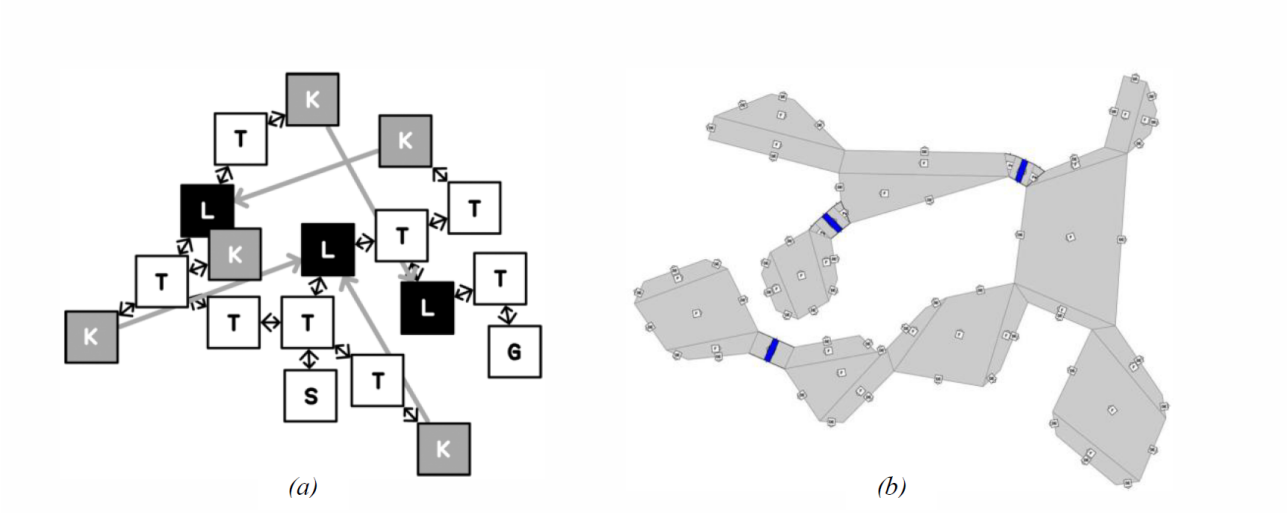


FIGURE 2.21 – Exemple de graphe de mission et son adaption en espace physique de jeu [22, Figures 13 et 15]. (a) Une mission avec des *Tasks* (tâches), *Keys* (clefs) et *Locks* (verrous). (b) Structure de la mission a, traduite en représentation spatiale.

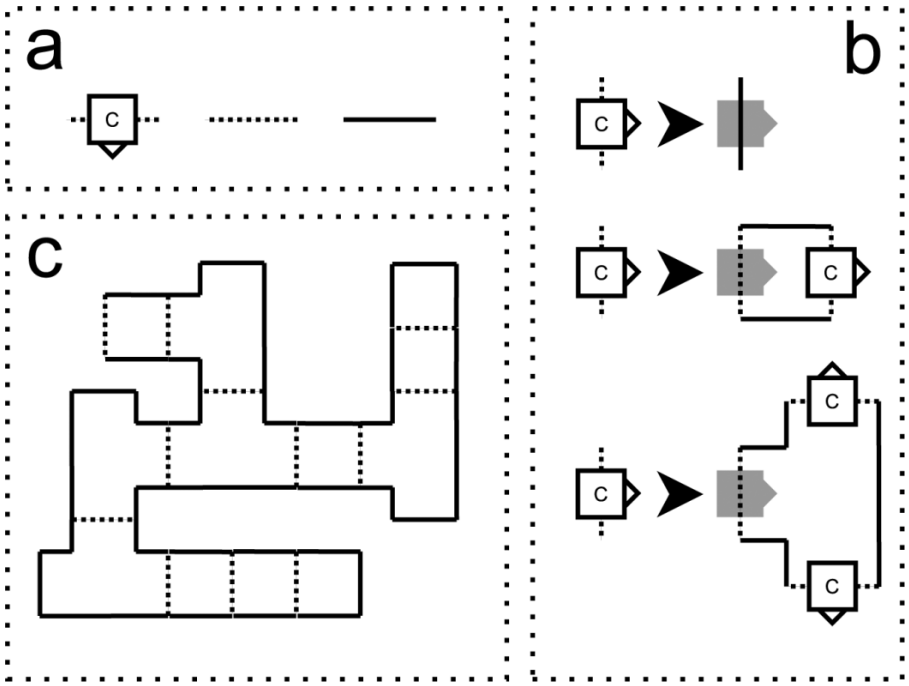


FIGURE 2.22 – Exemple d'une grammaire de forme : (a) alphabet, (b) règles et (c) résultat [22, Figure 16].

Cette approche est la première à introduire avec succès la génération d'éléments correspondants à des mécaniques de jeu, notamment grâce au concept de *mission grammar*. L'alphabet proposé est composé de différentes tâches pour le joueur, à savoir combattre des ennemis, résoudre des énigmes ou récupérer un trésor. Cela permet la prise en compte de la notion de défis et de récompenses. Cette méthode se révèle très ingénieuse, car, en considérant le concept de graphe de missions, l'algorithme de génération est ainsi plus adapté à la création de structures non linéaires favorisant l'exploration. De plus, son application a été possible pour la production d'un jeu complet. En effet, en 2017, Joris Dormans a publié son jeu *Unexplored* [160] basé entièrement sur la génération procédurale, en utilisant essentiellement des grammaires de graphes et de formes. Pour ce faire, il a, entre autres, approfondi son approche avec les grammaires de graphes en incorporant le principe de cycle, comme nous le montre l'exemple Figure 2.23 page suivante, afin de générer des graphes cycliques. Ces derniers offrent davantage de possibilités pour la génération des quêtes.

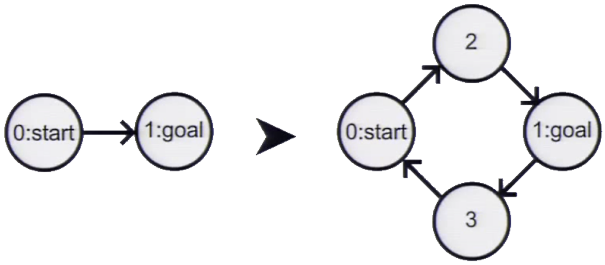


FIGURE 2.23 – Exemple d'une règle de réécriture engendrant un cycle [23].

Inspiré par les travaux de Joris Dormans, Roland van der Linden a proposé également en 2013 l'utilisation d'une grammaire de graphe afin de générer des niveaux de donjon [161]. Cette grammaire est basée sur la description d'une mission selon des mécaniques de jeu, à savoir le gameplay. Elle s'exprime selon des contraintes de design à partir d'un vocabulaire orienté d'après des concepts de gameplay : les actions que le joueur va pouvoir réaliser dans le jeu. L'objectif de la recherche de Roland van der Linden avait pour but d'obtenir un système plus généralisé que les autres approches existantes de génération de donjons basées sur les grammaires. De ce fait, son système s'appliquerait alors à un plus large éventail de jeux et de genres variés.

2.3.4.3 Autres applications des grammaires

Nous avons montré que les grammaires ont été étudiées pour générer des missions et des espaces de jeu de type donjon. Il existe également de nombreuses autres grammaires développées dans le domaine de la génération procédurale. Nous pouvons citer la génération de ville grâce à l'utilisation de *L-System* [162], et de buildings avec la grammaire de forme [163]. Nous avons montré précédemment l'application du *L-System* dans le domaine de la végétation, tels que les arbres et les plantes [164], couramment utilisé dans la production de jeux vidéo, grâce en outre au logiciel *SpeedTree* [60]. D'autre part, le *L-System* a servi à générer des environnements sous-marins [165], des grottes [166], mais aussi de la musique [167] et de l'architecture [168].

Par ailleurs, cette approche est également étudiée dans le domaine de la génération de narration avec plusieurs travaux notables avec notamment le projet *Façade* [169], et dans la génération de niveau pour jeu de plateformes 2D, basé sur du rythme [47].

Il semble pertinent de noter que dans les études et algorithmes cités précédemment, les grammaires

ont été principalement utilisées comme méthodes constructives. En effet, elles peuvent être implémentées dans une méthode *Search-based* avec un algorithme évolutionnaire, par exemple, dans le cadre de la génération de donjons [170] ou de roches [171].

2.4 Conclusion

Dans ce chapitre, nous avons étudié le concept de génération procédurale de contenu, différentes méthodes et divers algorithmes. Cela nous a permis d'acquérir une vision plus complète de ce sujet et de disposer des éléments nécessaires à la compréhension de notre champ d'étude.

Nous nous sommes penchés sur les raisons justifiant son utilisation et son utilité, que ce soit, par exemple, pour des questions de coûts, de rejouabilité ou d'adaptabilité. Nous avons, par la suite, pu constater les propriétés, les limites et les compromis de telles implémentations. Dans un second temps, nous avons détaillé les principales approches et méthodes de génération, qui nous paraissaient pertinentes. Nous avons distingué les algorithmes constructifs, qui produisent le contenu en une seule fois, et les algorithmes essai-erreur et itératifs, plutôt basés sur l'évaluation jusqu'à l'obtention d'un résultat satisfaisant. Nous avons également exploré la programmation par contraintes et les grammaires génératives.

Finalement, nous avons constaté que la PCG est un domaine de recherche particulièrement vaste. Beaucoup de chercheurs ont approfondi ce sujet, et ce, depuis de très nombreuses années. Cependant, les problématiques liées à la PCG sont toujours au cœur de nouvelles études. De nouvelles propositions et découvertes enrichissent sans cesse ce domaine.

De notre côté, nous avons choisi de nous concentrer sur le critère de la diversité possible d'un algorithme, c'est-à-dire sa liberté d'expression, avec une certaine qualité dans ses résultats. Nous pensons que les concepts d'algorithmes constructifs et itératifs présentent un intérêt certain par rapport à notre sujet d'étude.

Dans le prochain chapitre, nous allons présenter plus précisément notre problématique de thèse et nos différents axes de recherche. Nous allons ensuite nous interroger sur les formalismes et les approches, mentionnés dans cette revue de littérature, qui semblent convenir à notre objectif d'étude. Nous proposerons finalement une méthode de PCG, que nous pensons innovante, axée sur une combinaison de divers d'algorithmes.

Chapitre 3

Problématique

Contenu

3.1 Introduction

Dans le précédent chapitre, nous avons abordé le thème de la génération procédurale. Nous nous sommes penchés sur différentes méthodes et algorithmes qui nous semblaient pertinents. Nous allons, à présent, nous concentrer plus particulièrement sur notre problématique.

Pour cela, nous proposons de définir, en premier lieu, l'objectif général de cette thèse et de poser nos premières questions de recherche. Puis, nous adapterons notre sujet à une problématique plus ciblée. Nous rechercherons, ensuite, à l'aide de formalismes existants, présentés dans l'état de l'art, des concepts qui nous semblent utiles à la résolution de notre étude. Pour finir, nous présenterons l'approche que nous estimons innovante dans le domaine de la PCG.

3.2 Objectif général de notre recherche

Notre travail de recherche se positionne dans le domaine de la génération procédurale de contenu dans le jeu vidéo. Cette thèse s'intéresse plus particulièrement aux questionnements liés à la diversité et à la qualité du contenu généré. Nous avons choisi de nous concentrer sur le critère de la diversité possible d'un algorithme, c'est-à-dire sa liberté d'expression, tout en conservant une certaine qualité dans ses résultats. Nous nous sommes donc, tout d'abord, interrogés sur la question suivante :

> Comment générer procéduralement une forte diversité d'espaces de jeu, tout en conservant une certaine qualité ?

Cette problématique nous semble importante à approfondir. La génération de niveaux de jeu est, en effet, à la fois une question de qualité et de diversité [77]. Dans notre recherche, nous choisissons d'explorer cet aspect de la diversité de la génération procédurale de contenu, car nous pensons que ce critère est fondamental également pour stimuler la curiosité, l'un des principaux facteurs de motivation incitant le joueur à poursuivre son jeu [172, 173].

Afin de proposer du contenu diversifié et captivant pour le joueur, nous choisissons d'aborder la génération procédurale du côté de l'expérience de jeu. Nous pensons que les niveaux de jeu existent principalement pour l'expérience de jeu qu'ils procurent : toute règle, contrainte ou étape de conception, ajoutée à un générateur, est censée garantir que les niveaux produits fourniront un type d'expérience recherché. Nous pensons que la diversité, évaluée en terme d'expérience de jeu, devrait ainsi être

pertinente du point de vue du joueur.

Cependant, l'expérience de jeu peut être aussi émergente et difficile à prévoir. Par exemple, si le joueur a le choix entre plusieurs chemins disponibles, nous devons alors prendre en compte ces possibilités afin d'avoir une estimation du type d'expérience que ce niveau peut fournir, plutôt que d'analyser uniquement les conditions de départ.

Dans notre étude, nous choisissons, d'autre part, de considérer la qualité comme un critère uniquement structurel. Nous avons, en effet, décidé de nous concentrer sur des questions de *level design*, plus précisément sur les erreurs de placement d'objets lors de la conception du niveau. Il s'agit pour nous d'un point essentiel dans le but d'obtenir des niveaux jouables.

En d'autres termes, nous cherchons à atteindre le plus grand espace possible de diversité d'expériences de jeu tout en limitant le plus possible les erreurs de *level design*. Il s'agit de maintenir une certaine qualité tout une conservant un maximum de diversité. Nous pouvons reformuler ainsi notre problématique :

> Comment générer procéduralement une forte diversité d'expériences de jeu, tout en conservant une certaine qualité structurelle ?

Afin de parvenir à cette diversité dans la PCG, nous avons choisi d'aborder la question suivante : « Pour un gameplay donné, peut-on proposer un algorithme qui génère des niveaux avec une qualité structurelle suffisante, tout en visant une expérience spécifique de jeu ? » Une certaine diversité devrait être atteinte en recherchant des niveaux qui sont éloignés les uns des autres dans l'espace d'expériences de jeu possibles.

L'objectif de cette thèse est donc de chercher à proposer une nouvelle approche innovante dans ce domaine, au travers de deux axes principaux de recherche :

- Comment atteindre une diversité pertinente d'expériences de jeu ?
- Comment obtenir des niveaux avec une certaine qualité structurelle ?

3.3 Focalisation de notre étude

Nous sommes conscients que le sujet de la problématique s'inscrit dans un domaine de recherche d'envergure. Nous avons donc dû adapter notre question à une étude plus ciblée et focalisée sur des critères précis. Nous avons choisi, pour cela, une application spécifique qui nous semble représentative d'un large éventail d'expériences de jeu et qui devrait nous permettre d'analyser nos résultats de manière significative. Les différentes caractéristiques et nos divers choix retenus concernant cette application sont détaillées ci-après.

Choix de la progression dans le niveau Nous avons décidé de nous concentrer uniquement sur la génération de niveaux ouverts non-linéaires en 3D. Ceux-ci ne contraignent pas le joueur à suivre un chemin unique et restreint, mais lui permettent de se déplacer plus librement dans son environnement. De tels niveaux devraient favoriser également une expérience de jeu intéressante au travers d'un gameplay émergent.

Choix du genre de gameplay Nous avons également décidé d'employer un gameplay spécifique. Nous avons, pour cela, choisi de développer un jeu de tir à la première personne. En effet, de notre point de vue, le *level design* représente un aspect fondamental de ce type de gameplay. De ce fait, notre générateur de niveau devrait avoir, ainsi, un impact significatif sur l'expérience de jeu finale. La caméra à la première personne met également l'accent sur le fait que le parcours du joueur dans le niveau détermine les expériences vécues.

Choix spécifique d'application Dans un jeu de tir à la première ou à la troisième personne, FPS/TPS en anglais, un niveau contient habituellement une géométrie obstructive, à la fois pour la vision et la navigation du joueur. De plus, des ennemis peuvent parcourir le niveau dans le but d'attaquer le joueur et de l'empêcher d'atteindre son objectif trop facilement. En effet, le joueur peut avoir une position précise à atteindre dans le niveau.

Les campements ennemis ou fortins des jeux *Ghost Recon* et *Far Cry* offrent typiquement ce genre d'expériences dans le cadre d'un jeu de tir FPS/TPS, il s'agit de jeux propres à Ubisoft [24, 174]. De plus, ils sont généralement utilisés conjointement avec une progression non-linéaire dans un niveau ouvert. Parmi les jeux d'action-aventure, les camps ou même les donjons de *Elden Ring* [175] peuvent

également s'inscrire dans ce contexte.

Nous pensons que cette application s'inscrit dans notre démarche et nos choix énumérés précédemment. Nous avons ainsi porté notre attention sur l'attaque et/ou l'infiltration de fortin dans un monde ouvert, comme dans le jeu *Ghost Recon : Wildlands* [24].

Dans cette optique, nous nous sommes également penchés sur la description de la structure des fortins peuplés de personnages non-joueurs et de leur gameplay dans la série *Ghost Recon* d'Ubisoft. Nous nous sommes ainsi intéressés à l'étude de la variabilité des dynamiques de jeu [1] offerte dans ces campements, à savoir par exemple, les combinaisons de compétences du joueur, les différentes situations et les parcours possibles permettant de réaliser les objectifs du fortin, voir Annexe A page 213.

Choix de la représentation concernant la structure du niveau Dans le cadre d'une reproduction de campement de « type Ghost Recon », nous avons choisi de travailler sur une grille 2D de placement de bâtiments et d'objets. En effet, la structure globale et le positionnement modulaire des diverses infrastructures composant un fortin, rappellent cette forme de quadrillage, voir Figure 3.1. De plus, le choix de cette implémentation devrait faciliter notre compréhension du *level design* concernant le placement d'objets et ceci dans le but d'atteindre une qualité structurelle.

FIGURE 3.1 – Exemple de campement dans *Ghost Recon : Wildlands* [24], à gauche la carte du jeu et à droite une vue aérienne du même emplacement.

En résumé, afin de cibler davantage notre étude, nous avons choisi de privilégier une application précise, à savoir un niveau 3D ouvert non-linéaire de type jeu de tir à la première personne, avec une infrastructure de campement dont le contenu est positionné sur une grille 2D. Ce choix de gameplay

1. Les dynamiques de jeu correspondent au comportement émergent qui découle du gameplay, lorsque les mécanismes du jeu sont exploités [176].

et de structure de niveau permettra de tester les capacités de génération de notre future proposition et d'évaluer ses performances de manière adéquate. Ce principe de campement étant suffisamment générique, il permettra éventuellement à notre algorithme de s'adapter à divers genres de jeux partageant ces mêmes concepts de design.

Notre application spécifique

- Niveau ouvert 3D non-linéaire
- Genre FPS
- Infrastructure de campement
- Disposition sur une grille 2D

Finalement, ces différents choix nous permettent de formuler une problématique plus précise que nous pouvons à présent appréhender :

Comment générer procéduralement une forte diversité d'expériences de jeu, tout en conservant une certaine qualité structurelle dans le placement des objets, pour des niveaux 3D ouverts non-linéaires de type campement du genre FPS, placés sur une grille ?

3.4 Étude des concepts présentés dans l'état de l'art

Nous allons à présent distinguer différents formalismes, détaillés précédemment dans l'état de l'art, qui nous semblent convenir dans la résolution de notre problématique de recherche. Nous allons ainsi introduire les concepts et les algorithmes, déjà existants, utiles à notre questionnement, à savoir proposer une méthode de génération procédurale qui offre le plus de diversité tout en maintenant la qualité structurelle dans ses résultats. Notre volonté est d'avoir un minimum de contraintes possibles sur la production des niveaux afin d'obtenir un maximum de diversité qualitative, et ceci pour des éléments de campements disposés sur une grille. Cette réflexion va nous permettre de proposer, à terme, une nouvelle approche grâce à l'appropriation de ces formalismes énoncés.

Il est important, à présent, de définir clairement nos besoins pour la génération de notre contenu. Tout d'abord, il est nécessaire de trouver un algorithme de recherche capable d'explorer un espace d'expériences de jeu, mais également une méthode d'évaluation des niveaux du point de vue de l'expérience de jeu afin de guider cette recherche. Enfin, nous voulons éliminer les niveaux que nous jugeons

invalides, c'est-à-dire qui possèdent des erreurs dans la disposition des éléments.

3.4.1 Search-Based avec évaluation par simulation

Comme nous sommes inscrits dans une recherche plus générale liée à l'expérience de jeu, nos contraintes de génération portent principalement sur la jouabilité de la solution générée et non sur l'algorithme de génération en lui-même. Dans la génération d'un niveau, nous cherchons à atteindre une certaine expérience de jeu, mais nous ne connaissons ni les règles ni les contraintes applicables à un algorithme uniquement constructif, c'est-à-dire qui génère en une seule fois le contenu. En d'autres termes, nous ne cherchons pas à formaliser des règles de *level design*.

Dans le cadre de cette idée, nous nous sommes penchés sur la convenance d'une méthode essai-erreur appelée *Search-Based*, dont le principe de base est le suivant : une solution satisfaisante à un problème existe dans un espace de solutions. Il serait ainsi possible de diriger l'exploration dans cet espace grâce à une évaluation par simulation à l'aide d'agents autonomes.

En effet, comme nous l'avons expliqué précédemment, l'expérience de jeu peut être émergente. Ainsi, un simple changement local d'un placement d'objet peut modifier l'expérience résultante, par exemple, en bloquant un chemin, en ouvrant un raccourci ou en fournissant un abri. Dans le cadre d'un campement, plus précisément, le simple fait d'ajouter une caisse par-dessus une autre peut bloquer la visibilité d'un sniper qui patrouille à proximité, et créer une opportunité pour le joueur de se déplacer en toute sécurité vers son objectif, ce qui peut changer toute l'expérience de jeu. En simulant le gameplay, nous devrions être en mesure d'évaluer cette expérience de jeu émergente.

Ainsi, évaluer cette expérience de jeu, avec la simulation d'une intelligence artificielle paramétrable et en se plaçant du point de vue du joueur lors de son parcours du niveau, semble particulièrement prometteur. Cette IA, ou agent autonome, représentant un joueur, simulerait la traversée de chaque niveau généré, et noterait un ressenti selon des heuristiques de perception. Notre joueur synthétique naviguerait dans un niveau et pourrait être amené éventuellement à l'explorer afin d'atteindre un certain objectif, ou être confronté à des ennemis.

Finalement, une évaluation par simulation guiderait la méthode itérative vers une certaine expérience de jeu selon les préférences initiales de l'agent joueur. Nous estimons qu'utiliser cette approche nous permettrait d'obtenir une diversité de résultats significative. Cependant, nous ne pensons pas

approfondir, lors de notre recherche, la notion d'affrontement avec des ennemis.

3.4.1.1 Face aux études existantes

Dans l'état de l'art, nous avons observé que le genre FPS a été étudié dans le cadre de recherches évolutionnaires et formalisé vis-à-vis de *design patterns*.

Nous pouvons citer en premier lieu Luigi Cardamone, qui a travaillé sur une solution *Search-Based* afin de générer des niveaux FPS, avec une fonction d'évaluation basée sur le temps moyen en combat des joueurs [177].

De son côté, Kenneth Hullett a recherché des modèles de conception de niveaux, ou *design patterns*, appliqués dans les jeux de tir à la première personne. Il a également étudié les relations de cause à effet entre ces *design patterns* et l'intention du concepteur dans les niveaux de FPS [87, 178].

Jan Kruse a proposé, quant à lui, une nouvelle approche semi-automatique de la conception de cartes urbaines. Il s'appuie sur un système évolutionnaire interactif et sur des agents autonomes afin de produire des niveaux de jeu FPS entièrement jouables [179].

Toutefois, dans le cadre de notre démarche, nous visons plus particulièrement à optimiser une recherche évolutionnaire selon des expériences de jeu spécifiques, sans formaliser le gameplay. De plus, nous cherchons à nous focaliser sur des niveaux ouverts non-linéaires de type campement.

Nous avons également constaté que plusieurs études ont été réalisées par rapport à l'utilisation d'agents autonomes, appelés personas, avec différents objectifs permettant d'évaluer et de tester la jouabilité de niveaux de donjon.

Antonios Liapis, dans le cadre d'une optimisation sous contrainte, a introduit différents personas, afin d'évaluer la jouabilité et la qualité de niveaux de donjons [114]. Sa recherche propose deux profils de personas différents, à savoir le tueur de monstres et le collectionneur de trésors, qui parcourent les niveaux. Finalement, l'auteur montre que l'algorithme génétique à deux populations, implémenté, découvre des niveaux jouables rapidement, et ce, de manière fiable. Les différents personnages, de surcroît, affectent la disposition, le niveau de difficulté et la portée tactique des donjons générés. L'étude d'Antonios Liapis nous semble particulièrement prometteuse, car elle démontre la faisabilité d'une telle approche reposant sur une simulation d'agent.

Les expérimentations de Christoffer Holmgård ont aussi retenu notre attention [115, 116]. Il pro-

pose, quant à lui, d'explorer la modélisation de comportements de joueur à travers l'entraînement de personas selon des archétypes différents, par exemple : le tueur de monstres, le collectionneur de trésors, le complétionniste et celui qui cherche à atteindre la sortie uniquement. Ces joueurs synthétiques apprennent ainsi à jouer à des niveaux de donjon. Ils pourraient, par la suite, tenir le rôle de critiques automatisées dans une génération procédurale de contenu.

La simulation par agent a ainsi été abordée de différentes manières, plus particulièrement dans le cadre de l'évaluation de donjons. Les archétypes proposés sont intéressants, cependant, nous cherchons plutôt à employer des heuristiques plus générales de perception qui décrivent une expérience de jeu. De plus, notre approche ne vise pas à mettre en place un système d'apprentissage pour son agent.

Face à ces différentes études, nous espérons aboutir à une nouvelle approche grâce au *Search-Based* et la simulation par agent autonome, dans le contexte d'un jeu de tir en environnement ouvert 3D non-linéaire de type campement.

3.4.2 Combiner deux aspects de la PCG

Comme présenté dans notre état de l'art, la génération procédurale de niveaux de jeu peut être appréhendée à l'aide d'algorithmes dits constructifs ou d'algorithmes dits essai-erreur [21].

Les algorithmes constructifs s'appuient principalement sur les connaissances en matière de design afin de créer des niveaux en une seule fois, sans itération ni évaluation. Ils peuvent être rapides à exécuter et sont souvent utilisés dans les jeux de type *rogue-like*. Toutefois, la mise en œuvre d'un tel algorithme nécessite le développement de règles et de contenus initiaux permettant au générateur de produire les résultats souhaités. En conséquence, la valeur des niveaux est définie par la quantité et la qualité de ces règles et contenus pré-conçus.

D'un autre côté, les approches essai-erreur incluent, quant à elles, un mécanisme d'évaluation en plus de la partie génération. Elles créent itérativement des niveaux dans le but de valider des critères d'évaluation spécifiques.

Les approches constructives et les approches essai-erreur peuvent être complémentaires. En s'appuyant sur des connaissances de design, les méthodes constructives peuvent construire rapidement des niveaux de qualité. Cependant, si un niveau de jeu peut être simulé et évalué grâce aux méthodes essai-erreur, il est, dans ce cas, alors possible de l'améliorer afin de le rendre « meilleur », ou encore

d'ajuster l'expérience de jeu qu'il procure. De plus, les niveaux moins qualitatifs pourraient également être détectés et écartés.

Un cas plus particulier des algorithmes essai-erreur, que nous avons mentionné précédemment, est appelé *Search-Based*. Celui-ci se concentre essentiellement sur l'utilisation d'algorithmes d'optimisation dans le but de maximiser une ou plusieurs fonctions d'évaluation. Cette méthode recherche un résultat satisfaisant en fonction d'un certain objectif dans un espace de solutions. Cependant, l'étape d'optimisation peut nécessiter un temps de calcul conséquent. En effet, l'un des inconvénients majeurs de cette méthode est sa faible efficacité, en particulier dans les grands espaces de recherche. Il est nécessaire, d'un autre côté, de définir un moyen adéquat d'encodage et d'évaluation concernant les niveaux. De plus, une métaheuristique telle que l'algorithme évolutionnaire ou l'algorithme génétique ne garantit pas nécessairement l'obtention de solutions optimales globales. Dans la plupart des cas, elle ne trouvera que des maxima locaux [13].

Les méthodes *Search-Based* explorent un vaste espace de génération, dans la mesure où elles ne sont limitées que par le score d'évaluation obtenu par la modification des niveaux. Ainsi, la recherche dans l'espace des niveaux peut être un long processus, car de nombreux niveaux incorrects ou même irréalisables seront quand même générés et testés. C'est pourquoi, nous proposons d'utiliser une méthode constructive générique [2] dans la boucle itérative permettant de générer des niveaux d'une certaine qualité structurelle, associés à chaque individu de la population, et cela, dans le but de limiter l'espace de recherche à des solutions uniquement valides, c'est-à-dire, pour nous, sans erreurs de placement d'objets dans la structure du niveau.

Notre idée principale serait donc d'utiliser, d'une part, une méthode itérative afin d'atteindre une forte diversité, et d'autre part, d'implémenter un concept constructif pour corriger les niveaux incorrects, mais avec un minimum de contraintes pour limiter le moins possible la méthode. Ainsi, restreindre l'espace de recherche à des solutions uniquement valides grâce à l'inclusion d'un algorithme constructif dans une méthode *Search-Based*, nous paraît être la démarche la plus appropriée face à notre objectif de thèse.

2. Le terme générique est employé ici dans le sens où l'algorithme n'est pas limité à un genre de jeu ou à des contraintes spécifiques de gameplay.

3.4.2.1 Face aux autres études

Comme nous avons pu le constater dans l'état de l'art, Sous-Section 2.2.3 page 37, les problématiques liées à la diversité et à la qualité ont déjà été abordées dans plusieurs études, mais selon une autre perspective.

Nous pouvons citer, en premier lieu, Antonios Liapis, qui s'est penché sur le problème de l'évaluation de la qualité des niveaux dans différentes catégories de jeux [180]. Son article présente une méthode d'évaluation des niveaux de jeu consistant en six formules mesurant trois patterns de la conception de jeux : *area control*, *exploration* et *balance*. Il applique ces différentes métriques dans le cadre de cartes multijoueur dans un jeu de stratégie et dans le cadre de niveaux pour un jeu solo de donjons *rogue-like*. Il a également employé une optimisation sous contrainte, avec deux populations.

Mike Preuss, quant à lui, a exploré plusieurs approches *Search-Based* permettant de créer un contenu de jeu de qualité et diversifié [77]. Dans ce but, il a employé, plus particulièrement, des approches basées sur des stratégies d'évolution avec ou sans mécanismes de préservation de la diversité, sur la recherche de nouveauté ou encore sur la recherche aléatoire. Il propose, ainsi, différentes mesures intéressantes selon la composition du niveau, l'aspect visuel ou encore selon des métriques inspirées par l'article de Antonios Liapis, mentionné ci-dessus. Son domaine d'étude est celui des niveaux de jeu de stratégie. Il constate que l'utilisation de mécanismes de préservation de la diversité peut aboutir à un contenu diversifié de qualité.

Bien que ces recherches soient intéressantes, nous cherchons, quant à nous, à atteindre une forte diversité avec des résultats éloignés dans l'espace de recherche grâce potentiellement à une simulation et à des heuristiques spécifiques de perception d'un agent. La qualité structurelle, quant à elle, serait déléguée à un algorithme constructif. De notre côté, nous ne souhaitons pas employer de préservation de diversité, ni de recherche de nouveauté. Notre démarche ne vise pas à comparer la différence entre les résultats dans un algorithme évolutionnaire.

3.4.3 Algorithme constructif pour générer et corriger les niveaux

Du point de vue des algorithmes évolutionnaires, nous essayons de résoudre un problème d'optimisation sous contrainte. En effet, nous souhaitons que nos niveaux fournissent à la fois une certaine expérience de jeu et respectent des contraintes locales relatives à des placements d'objets.

Cependant, la mutation et le croisement, mis en œuvre dans un algorithme évolutionnaire, peuvent produire des solutions considérées comme erronées, selon nos contraintes. Nous devons donc mettre en place un moyen de traiter les individus incorrects. Dans l'état de l'art, Sous-Section 2.3.2.6 page 53, nous avons détaillé plusieurs solutions déjà existantes.

Nous aurions eu la possibilité de nous baser uniquement sur une méthode *Search-Based*, purement aléatoire, sans algorithme constructif. Les erreurs de placement et les connexions faussées entre objets auraient dû, dans ce cas, être pénalisées au travers de malus arbitraires, c'est-à-dire de règles prédéfinies. Cela correspondait, en soi, à un système de correction similaire à des méthodes constructives.

De ce fait, au lieu de pénaliser arbitrairement le résultat ou d'implémenter un algorithme *FI-2Pop* [125], nous avons choisi de privilégier, comme déjà mentionné, un algorithme constructif, tel un opérateur de réparation, permettant d'éliminer les erreurs de placement. Cette proposition semble être une alternative intéressante dans ce cas particulier.

Nous avons vu, précédemment, que nous avons des contraintes de génération principalement sur la solution et sa jouabilité. Nous cherchons donc un algorithme pouvant proposer une qualité structurelle dans sa génération de niveaux et permettant d'aboutir à des résultats considérés valides, c'est-à-dire sans erreurs de placement d'objets, afin de limiter l'espace de recherche de la méthode itérative.

De plus, nous visons à utiliser le moins possible de restrictions pour atteindre cette qualité structurelle. En effet, nous voulons que notre générateur ait peu de contraintes préalables ou de règles de design concernant la structure du niveau, pour qu'elle puisse émerger lors de la recherche d'une expérience spécifique de jeu. Limiter le moins possible le générateur lui donnera également plus de liberté, et ainsi, lui permettra d'atteindre potentiellement plus de diversité dans ses résultats.

3.4.3.1 Analyse de méthodes constructives

De manière à trouver un algorithme constructif générique adapté à notre solution d'approche, nous allons confronter, ci-dessous, différentes méthodes avec une possible implémentation constructive, vues dans l'état de l'art.

Template-based Même si nous pouvons considérer que l'avantage de cette méthode réside dans le contrôle sur la génération, la diversité des niveaux dépend, néanmoins, principalement de la quantité et de la qualité des templates créés à la main. En effet, les templates contiennent déjà les éléments

pré-conçus en lien avec la conceptualisation du niveau. Cependant, nous nous intéressons à l'aspect émergent du gameplay pour lequel nous générons des niveaux, c'est-à-dire l'expérience de jeu. Nous n'avons donc pas besoin de templates qui restreindraient l'espace de recherche. Notre objectif est de fournir la plus petite quantité de contenu initial, avec un minimum de règles, afin de générer de nombreux niveaux variés.

Partitionnement de l'espace Un tel algorithme est très utile à partir du moment où l'on considère l'espace selon différentes zones, par exemple lorsque l'espace est organisé en pièces comme dans les bâtiments. Toutefois, nous ne cherchons pas à appliquer des règles de séparation en zones pour notre espace de jeu. C'est pour cela que le partitionnement ne nous semble pas approprié dans notre cas.

Automate cellulaire Ce type d'algorithme est particulièrement intéressant. En effet, il se base sur un ensemble simple de contraintes locales pour faire émerger l'espace de jeu. Malheureusement, par sa nature émergente, il est difficile de prédire le résultat final et sa jouabilité. Dans notre cas d'étude, il aurait été possible de complexifier les règles, mais cela aurait été contraire à notre volonté de ne pas formaliser des règles de *level design*.

Marche aléatoire Le caractère aléatoire et chaotique de cet algorithme représente pour nous un inconvénient majeur. En effet, cette démarche ne s'inscrit pas dans le processus recherché.

Grammaire générative Notre objectif étant de nous placer du côté de l'expérience de jeu, nous cherchons, de ce fait, à formaliser un minimum de règles. En effet, nous voulons donner autant de liberté que possible au générateur en ce qui concerne la structure du niveau. Cependant, les grammaires génératives adoptent l'approche opposée, fournissant principalement un moyen d'encoder des connaissances préalables concernant la conception du niveau sous la forme d'un ensemble de règles. Il convient également de noter que la génération d'un espace de jeu à partir d'un graphe de mission n'est pas simple et nécessite de nombreuses étapes [22]. Néanmoins, le résultat final est toujours correct par rapport aux règles de production initiales, constituant, tout de même, un atout significatif.

Programmation par contrainte Cette approche présente l'avantage d'obtenir des résultats toujours valides selon des contraintes initiales. Cependant, l'inconvénient majeur réside dans la nécessité de

connaître suffisamment les caractéristiques du résultat souhaité afin d'établir des contraintes adaptées.

3.4.3.2 Wave Function Collapse

Le WFC se place dans la dernière catégorie vue ci-dessus, celles des méthodes de satisfaction de contraintes, et nous semble être l'algorithme le plus prometteur par rapport à nos choix de recherche, c'est-à-dire :

- Obtenir des niveaux corrects dans leur structure avec le moins de contraintes possibles ;
- Travailler uniquement sur le positionnement spatial des objets dans un niveau de jeu.

Le WFC s'accorde parfaitement avec notre volonté de travailler sur le positionnement spatial des objets. Il peut, en effet, propager des contraintes locales d'adjacence selon la disposition des objets.

De plus, cet algorithme permet avec un niveau de jeu pré-existant, à savoir un modèle initial, d'extraire des contraintes d'adjacence de *level design*, mais également de répliquer des fréquences d'apparition de tel ou tel objet. Ainsi, il est possible d'obtenir de nouveaux niveaux crédibles de taille arbitraire selon ces contraintes.

D'un autre côté, il est possible, avec l'intervention d'un utilisateur, d'interagir avec la génération dans le cadre d'une approche mixte, par exemple en verrouillant des sections de la grille avec des objets prédéfinis à la main.

Cet algorithme est suffisamment générique, car il n'est pas limité à un genre de jeu spécifique. En effet, il peut être appliqué à n'importe quel générateur procédural de niveaux reposant sur un type de grille, et utilisant des contraintes d'adjacence.

En respectant les contraintes, les niveaux produits ne présentent pas d'erreurs de placement d'objets, tels que des escaliers ne menant nulle part. Le WFC permet, ainsi, d'éviter de spécifier manuellement toutes les erreurs possibles de placement ou de définir des malus arbitraires.

Grâce à cette méthode, il serait possible de ne pas limiter la capacité du générateur, car nous disposerions de très peu de contraintes et de règles de conception concernant la structure du niveau. En effet, uniquement des contraintes simples d'adjacence de *level design* pourraient être utilisées grâce au WFC. Avec peu d'informations structurelles, nous pourrions, dans ce cas, éviter des erreurs de conception évidentes.

Il convient néanmoins de souligner que les contraintes utilisées ici dans le WFC pourraient être difficiles à évaluer du point de vue d'une simulation par agent autonome. C'est pourquoi, il serait envisageable d'utiliser l'agent joueur afin de parcourir le niveau uniquement dans le but d'évaluer sa perception. Ainsi, il ne disposerait pas d'informations sémantiques concernant les objets disposés dans le niveau. Pour le joueur synthétique, les escaliers ne représenteraient qu'une autre case navigable, et donc uniquement qu'une façon d'explorer le niveau. Pour résumer, le WFC, en corrigeant les erreurs de placement, permettrait à la recherche *Search-Based* de se focaliser seulement sur l'évaluation grâce à la perception de l'agent.

En définitive, cet algorithme semble être apte à reproduire des niveaux de type campement sur une grille, grâce à la disposition modulaire des infrastructures. Par exemple, lors de la construction d'un camp ennemi, les escaliers ne peuvent pas être placés au milieu de la route, et doivent mener à une position valide. Les clôtures doivent être placées autour du camp et non à l'intérieur, et les véhicules ne peuvent pas être garés n'importe où. Ces différentes règles contraignent alors le placement relatif des objets et peuvent être utilisées dans le cadre d'un algorithme WFC qui serait alors capable de générer un camp sans erreurs de contraintes de positionnement.

Pour conclure, le *Wave Function Collapse* nous semble particulièrement adapté pour corriger les erreurs d'une méthode *Search-Based* en tant qu'opérateur de réparation.

3.5 Proposition de méthode de PCG

En vue de répondre à notre problématique, nous proposons de développer et d'évaluer un générateur de niveaux. En nous appuyant sur les idées de concepts émises précédemment grâce aux méthodes et formalismes existants, nous proposons d'opter pour une approche mixte qui s'articule autour de deux aspects de la PCG, à savoir :

- L'évaluation du contenu généré grâce à une méthode itérative, *Search-Based*, associant une recherche évolutionnaire et une simulation d'agent autonome. Un joueur synthétique va explorer les niveaux produits et nous permettre de noter son parcours selon différents critères, à savoir des heuristiques, dans le but de guider l'algorithme évolutionnaire vers une expérience spécifique de jeu ;
- La génération du contenu par l'implémentation de l'algorithme par contraintes *Wave Function*

Collapse, de façon constructive dans l'algorithme évolutionnaire, et qui permettra la création de niveaux.

En d'autres termes, nous proposons un pipeline de génération procédurale de niveaux de jeu qui combine une approche *Search-Based* avec une méthode constructive générique, dans notre cas le WFC. Plus précisément, nous voulons montrer qu'un algorithme constructif générique peut aider un algorithme *Search-Based* à traiter un problème d'optimisation sous contrainte sans pénaliser une valeur de fitness. Dans ce cas, la méthode constructive se charge seulement des contraintes liées à la génération du niveau afin de proposer des résultats uniquement valides dans l'espace de recherche, tel un opérateur de réparation pour chaque individu de la population de l'algorithme évolutionnaire. L'approche *Search-Based*, quant à elle, peut alors se concentrer sur l'amélioration de l'expérience de jeu simulée.

Il est important de noter que le WFC que nous pensons implémenter sera considéré comme constructif, car nous n'inclurons pas le concept de *backtracking* au sein même de cet algorithme. Celui-ci va donc générer en une seule fois le contenu pour chaque exécution. Nous voulons que seul l'algorithme de recherche effectue des itérations et des évaluations.

Pour conclure, nous pensons que cette combinaison d'algorithmes est novatrice dans le domaine de la PCG, et nous permettra d'obtenir des résultats intéressants et pertinents. Dans ce sens, elle nous semble appropriée à notre objectif de recherche. En effet, nous estimons qu'une diversité d'expériences de jeu devrait être atteinte grâce à l'implémentation d'une méthode *Search-Based* pour explorer et simuler le contenu. La qualité structurelle, quant à elle, pourrait être alors obtenue grâce à l'utilisation du WFC pour corriger les niveaux de l'espace de recherche. Le cas d'application spécifique d'un campement nous permettra d'avoir un banc d'essai intéressant pour cet algorithme.

Cependant, une telle proposition soulève des questions supplémentaires quant à sa faisabilité, ce qui va nous amener à soumettre différentes hypothèses d'implémentation, détaillées dans le chapitre suivant. Il s'agira également d'évaluer les performances et les capacités d'un tel générateur de niveaux.

3.6 Conclusion

Dans ce chapitre, nous avons commencé par détailler notre objectif général et nos axes de recherche. Puis, nous avons focalisé notre étude sur une problématique plus précise à laquelle nous pouvions proposer une résolution concrète. Notre thèse s'articule, ainsi, autour de la question suivante :

« Comment générer procéduralement une forte diversité d'expériences de jeu, tout en conservant une certaine qualité structurelle dans le placement des objets, pour des niveaux 3D ouverts non-linéaires de type campement du genre FPS, placés sur une grille ? » Nous nous sommes appuyés, par la suite, sur des méthodes de génération et de formalismes existants afin d'aboutir à une méthode adaptée à notre problématique. Nous avons ainsi exposé nos arguments concernant les différents choix possibles pouvant mener à une proposition appropriée. Finalement, ces décisions ont mené à la description d'une nouvelle méthode dans le domaine de la recherche de génération procédurale de contenu dans les jeux vidéo, avec l'intégration d'une approche constructive dans une méthode itérative.

Le développement de cette étude pourrait amener à la création d'un générateur d'expérience spécifique de jeu à l'aide d'une méthode permettant de générer des niveaux de qualité, variés et intéressants. Dans ce but, nous envisageons d'utiliser le WFC comme opérateur de réparation dans une approche *Search-Based* avec la simulation d'un agent à l'aide de différentes heuristiques de perception. Cette combinaison nous permettrait de générer des niveaux de jeu 3D ouverts non-linéaires de type FPS, avec une infrastructure de campement dont le contenu est disposé sur une grille 2D.

Caractéristiques de notre proposition de méthode de PCG

- La génération de niveau par WFC, inclue dans une approche *Search Based*, comme opérateur de réparation dans une optimisation sous contrainte ;
- L'évaluation des résultats par la simulation du parcours d'une IA paramétrable ;
- L'utilisation d'heuristiques de perception dans la notation de l'agent, afin d'obtenir une diversité d'expériences de jeu, qui a du sens du point de vue du joueur ;
- Le tout dans le cadre de niveaux 3D ouverts non-linéaires de type FPS, avec une infrastructure de campement sur une grille 2D.

Nous abordons, ainsi, nos deux axes de recherche de la façon suivante : l'utilisation du *Search-Based* avec une simulation pour atteindre une forte diversité d'expériences de jeu et l'emploi du WFC dans le but d'obtenir des niveaux d'une certaine qualité structurelle. En effet, grâce au WFC, nous pourrons utiliser des exemples de niveaux pré-existants et extraire leurs contraintes de *level design* afin de reproduire leur structure dans de nouveaux résultats. Cela devrait nous permettre, également, d'obtenir des niveaux plausibles et crédibles dans la disposition des éléments. D'un autre côté, notre simulation d'agent inclura le parcours du niveau selon le point de vue du joueur et évaluera son « expérience » selon diverses heuristiques de perception uniquement. Grâce à cette évaluation, nous

espérons atteindre une diversité pertinente d'expériences de jeu.

Afin d'évaluer et de démontrer le bien-fondé de l'utilité de cette nouvelle méthode de PCG, nous serons amenés à comparer notre approche à d'autres méthodes similaires déjà existantes et de prouver l'efficacité de notre combinaison novatrice. En centrant notre étude sur le *level design* et le placement d'objets sur une grille, dans des niveaux 3D ouverts non-linéaires de type FPS, avec une infrastructure de campement sur une grille 2D, nous espérons faciliter nos expérimentations en nous focalisant sur ce seul concept.

Comme nous cherchons à intégrer le moins de contraintes et de règles possibles, notre proposition pourrait être généralisée éventuellement à d'autres types de gameplay. De plus, nous espérons que les heuristiques de perception que nous allons développer, nous permettrons de nous rapprocher de la véritable expérience joueur afin d'orienter la génération vers un objectif précis.

Dans le prochain chapitre, nous allons nous pencher sur les divers algorithmes de notre proposition et son éventuelle mise en place. Nous pourrons ainsi voir concrètement la faisabilité d'une telle combinaison. Nous détaillerons différentes hypothèses afin de pallier d'éventuelles difficultés pouvant se présenter lors du développement de cette méthode de PCG.

Chapitre 4

Difficultés et hypothèses d'implémentation

Contenu

4.1 Introduction

Nous avons détaillé, précédemment, notre problématique et suggéré une méthode de PCG adaptée. Nous allons à présent aborder les risques et difficultés de l'utilisation des algorithmes proposés. Dans ce chapitre, notre réflexion va nous amener à émettre plusieurs hypothèses de développement afin de pallier différents obstacles sur la mise en œuvre de notre combinaison de méthodes. Nous verrons, au final, l'impact de nos choix sur le déroulé de notre approche.

4.2 Difficultés envisagées

Comme nous l'avons présenté précédemment, nous envisageons d'allier la mise en place d'un WFC avec la puissance du *Search-Based*. Cependant, nous devons tenir compte de diverses difficultés et inconvénients qui peuvent résulter de cette implémentation spécifique et de ces algorithmes.

Notre premier questionnement repose sur l'inclusion du WFC dans une approche *Search-Based* et son pilotage. En effet, pour le bon fonctionnement de notre proposition, nous devons prouver, en premier lieu, qu'influencer la génération d'un algorithme par contraintes, autosuffisant de base, avec une méthode itérative, est réalisable. Pour ce faire, il nous faut également envisager un encodage adéquat des solutions, c'est-à-dire de nos niveaux de jeu, afin qu'ils soient manipulables par la méthode itérative.

Deuxièmement, nous avons retenu que l'inconvénient majeur de chacune des méthodes choisies, à savoir le WFC, le *Search-Based* et la simulation par agent, réside potentiellement dans la quantité de calcul nécessaire. En effet, dans le cadre du WFC, la propagation des contraintes prendra plus ou moins de temps, en fonction du nombre de modules, de contraintes et de la taille du résultat final. L'un des principaux désavantages d'une méthode itérative réside également dans sa faible efficacité, en particulier pour explorer et simuler de grands espaces de recherche de niveaux de jeu.

Enfin, dans le cadre de la simulation par agent, nous devons définir différentes heuristiques pertinentes, en vue d'évaluer une expérience de jeu du point de vue du joueur. Dans notre application, les heuristiques représentent des mesures de perception pour l'agent lors de son parcours du niveau.

Finalement, nous pouvons résumer nos diverses interrogations comme suit :

- Comment inclure et influencer le WFC et comment encoder un niveau ?
- Comment limiter l'impact sur le temps de calcul ?
- Comment évaluer l'expérience de jeu de manière pertinente ?

Il convient également de noter qu'une métaheuristique tel que l'algorithme évolutionnaire ou l'algorithme génétique ne garantit pas l'obtention de solutions optimales globales. Dans la plupart des cas, elle ne trouvera que des maxima locaux [13]. Lorsqu'un local maxima est atteint par un individu particulier, celui-ci va potentiellement rester en tête du classement dans la population pendant un certain nombre d'itérations dans un algorithme d'optimisation, ce qui peut amener au problème suivant : les individus commencent à se ressembler. En effet, comme le leader survit à travers les générations, il va participer à de nombreux croisements, distribuant ainsi ses gènes aux autres membres de la population. Les individus éloignés du leader sont alors surclassés et sont progressivement éliminés. Ce maxima local entraîne, de ce fait, petit à petit une perte de la diversité dans la population [181]. Cette notion, inhérente à l'utilisation d'un tel algorithme, mérite certes réflexion, mais ne sera pas prise en compte dans la résolution de notre sujet de recherche.

4.3 Hypothèses de résolution

Nous allons à présent énumérer nos diverses propositions de résolutions concernant les difficultés et questionnements, soulevés précédemment.

4.3.1 Combinaison et pilotage

Notre première interrogation porte sur l'inclusion possible de l'algorithme constructif, le WFC, avec un algorithme évolutionnaire dans une approche *Search-Based*. Tout d'abord, nous devons ainsi démontrer que piloter le WFC vers un résultat précis, selon différents paramètres, est faisable. En effet, cet algorithme a été initialement développé dans le but de propager automatiquement les contraintes sans intervention extérieure.

Nous proposons ainsi l'hypothèse suivante : l'implémentation de « zones », dispersées sur la grille de génération, modifiant la probabilité de sélection de certains modules dans le déroulement du WFC.

Une zone représenterait, dans notre cas, une surface d'une ou plusieurs cellules de la grille, et modifierait la probabilité de sélection d'un module précis avec une valeur arbitraire que nous pourrions définir. La Figure 4.1 illustre un premier concept de cette proposition. Ces « zones de probabilités » auraient ainsi pour rôle de modifier et de contraindre le déroulement de la génération du WFC lors du choix des assets [1]. De ce fait, si un module n'est pas disponible, il ne sera pas influencé par cette modification. Ainsi, nous ne contournerons pas le fonctionnement du WFC, son déroulé restera le même et respectera les contraintes initiales d'adjacence. Nous devrons ainsi, en premier lieu, valider lors de notre développement, l'implémentation de ces zones de probabilités.

De surcroît, ces zones permettront éventuellement d'encoder le niveau et feront ainsi la liaison entre le WFC et la méthode *Search-Based*. En effet, la méthode itérative manipulerait alors uniquement les valeurs liées à ces zones, pour un individu précis, en tant que génotype. La construction du phénotype, à savoir le niveau de jeu, serait réalisée, dans ces conditions, par le WFC grâce à l'influence de ces zones.

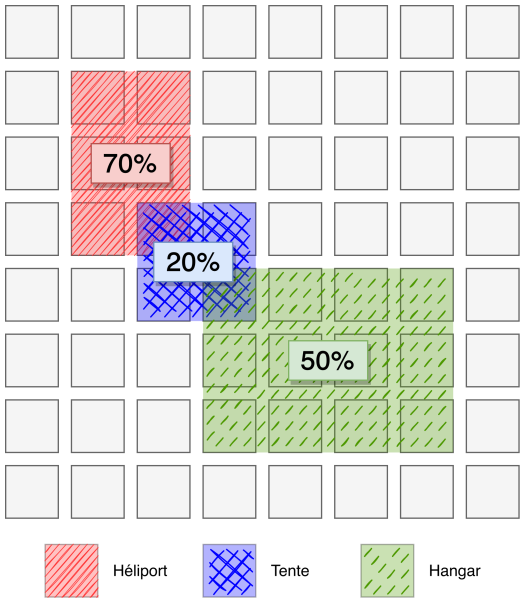


FIGURE 4.1 – Concept envisagé : les zones de probabilités.

1. Le terme asset sera utilisé dans cette thèse et dans le contexte de l'implémentation d'un *Wave Function Collpase*, comme synonyme de module. Le terme module représente l'élément spécifique manipulé dans le déroulement d'un WFC.

4.3.2 Temps de calcul

4.3.2.1 Greyblocking

La recherche évolutionnaire va explorer l'espace des solutions afin d'obtenir le meilleur résultat possible par rapport à certains objectifs. Cependant, cette démarche va nécessiter un temps de calcul conséquent selon les paramètres de recherche, que ce soit le nombre de boucles à effectuer ou la population totale à prendre en compte, par exemple. De plus, la durée de l'algorithme de génération, le WFC, semble dépendre directement du nombre d'assets disponibles qu'il peut utiliser et de la taille de la grille du résultat désiré.

Afin d'économiser du temps sur la création de niveau dans la génération du WFC, il nous paraît pertinent d'accorder moins d'importance à l'aspect visuel des objets 3D à afficher, qui sont associés aux modules du WFC, et de nous concentrer plutôt sur les mécaniques de jeu propre au *level design*. En effet, de nombreux éléments constituant un niveau possèdent la même utilité d'un point de vue du gameplay. Nous proposons donc l'utilisation du concept de *greyblocking*[2] permettant de remédier à un temps de calcul potentiellement excessif pour le WFC. Ce terme s'inpire directement de la notion de *greyboxing* [182, Chapitre 9]. Cette étape intermédiaire de conception pourrait générer un niveau composé uniquement de catégories d'assets, c'est-à-dire des modules *greyblocks* regroupant les objets ayant la même utilité de gameplay. Nous pourrions ainsi éviter d'employer plusieurs variations visuelles, par exemple celles représentant un escalier. D'un autre côté, ces modules ne représenteraient pas les objets finaux à afficher pour le joueur, mais conviendraient néanmoins pour la simulation de notre agent dans la méthode itérative. Finalement, cela nous permettrait de limiter l'impact de la complexité en temps de génération en diminuant le nombre de modules manipulés par le WFC, mais aussi de nous approprier une méthode de travail propre aux *level designers*.

Par la suite, nous pourrions utiliser un résultat final de *greyblocking* obtenu dans notre méthode, afin de le remplacer par des objets 3D plus visuels lors d'une seconde étape de génération, voir Figure 4.2 page suivante. Le résultat représenterait alors un niveau pouvant être présenté à un joueur.

Au travers de notre étude, nous serons amenés à déterminer le meilleur ratio possible pour le nombre de *greyblocks* utilisables dans le WFC afin de conserver un temps de calcul raisonnable. Pour cela, nous devrons référencer les temps de génération requis selon la taille du niveau et le nombre

2. Le terme *greyblocking* provient de notre choix de combiner les mots *greyboxing* et *blockout*.

d'assets disponibles.

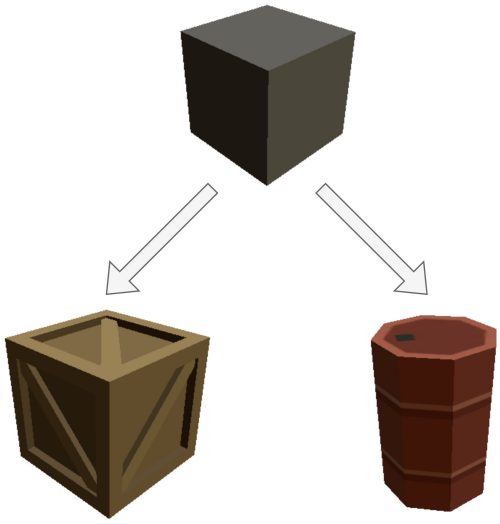


FIGURE 4.2 – Concept envisagé : le *greyblocking*. Par exemple, le *greyblock* peut être remplacé par deux objets ayant la même utilité.

4.3.2.2 Abstraction du gameplay

Nous prévoyons que notre fonction d'évaluation aura également un impact significatif sur le temps de calcul lors de la notation de chaque résultat.

Nous proposons, ainsi, d'abstraire le comportement possible d'un joueur, car simuler entièrement un déplacement et des interactions dans un environnement de jeu nécessiteraient un temps conséquent supplémentaire. En effet, combiner la génération par WFC, la recherche évolutionnaire et une simulation par agent nous semble déjà trop chronophage en matière de complexité algorithmique temporelle. C'est pourquoi, nous proposons de simplifier le gameplay de notre agent joueur. Nous pensons ainsi évaluer uniquement son parcours selon ses déplacements, sur une grille de navigation prédéfinie, grâce à sa perception sur chaque nouvelle case visitée. Cela devrait nous permettre de garantir une exploration réalisable de l'espace de recherche.

Il serait envisageable, par la suite, d'intégrer des comportements habituels d'IA dans les jeux de tir, c'est-à-dire, des prises de décisions et des déplacements en temps réel, par exemple grâce à un cône de vision ou à un cercle d'alerte.

4.3.3 Valeurs de perception

Notre volonté étant de guider l'algorithme évolutionnaire vers une expérience de jeu spécifique, nous devons donc définir une méthode d'évaluation des niveaux. Comme nous nous plaçons du point de vue du joueur en réalisant une simulation du parcours, nous devons établir des facteurs précis que l'agent pourra quantifier. En faisant varier ces derniers dans l'évaluation de l'agent, nous serions ainsi à même d'influencer l'expérience finale du niveau.

Afin de trouver des valeurs de perception pertinentes, pour le joueur synthétique, nous nous sommes penchés sur les travaux de recherche de Thomas W. Malone dans l'étude de la motivation dans les jeux vidéo [172]. Ce chercheur s'est inspiré des travaux de D. E. Berlyne, entre autres. Nous allons, de ce fait, nous concentrer également sur les études menées par ce dernier concernant l'influence de la nouveauté et de la complexité sur la curiosité [183, 184]. Nous pouvons aussi citer le rapport de Thomas Constant ayant pour sujet « La perception de la variabilité et facteurs de la curiosité dans les jeux » [185], nous permettant de mieux cerner ces différents aspects de la psychologie humaine.

Nous proposons, en conséquence, d'évaluer le niveau selon deux valeurs de perception, mentionnées par D. E. Berlyne, que nous pouvons attribuer à l'agent, à savoir : la nouveauté et la complexité. Nous pensons rajouter, en outre, une troisième et dernière valeur, portant sur le genre FPS/TPS, à savoir : la sécurité. La Figure 4.3 page suivante présente ces trois concepts que nous visons à intégrer dans notre évaluation. Nous supposons que ces heuristiques intéressantes vont nous permettre d'évaluer une expérience de jeu pertinente du point de vue du joueur. Cependant, nous serons amenés à les adapter à l'implémentation de notre approche.

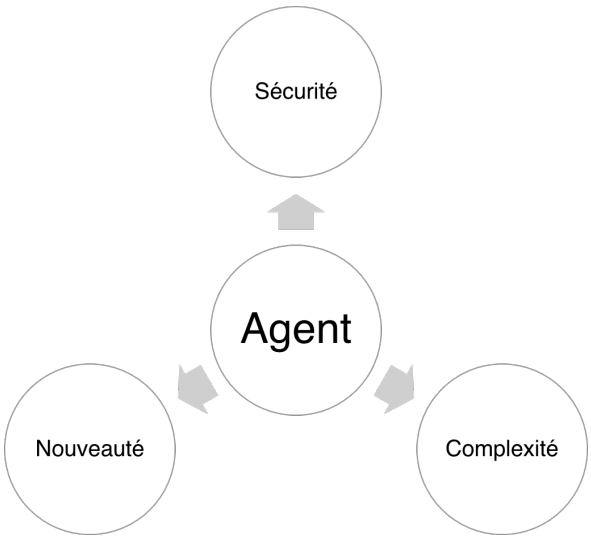


FIGURE 4.3 – Concept envisagé : les trois valeurs de perception, à savoir nos heuristiques, pour la notation de l'agent.

Nous souhaitons à présent apporter quelques précisions concernant ces trois valeurs de perception choisies. La nouveauté et la complexité seront abordées du point de vue des travaux de D. E. Berlyne, alors que la sécurité décrira notre réflexion quant à son apport.

Nouveauté La nouveauté constitue pour D. E. Berlyne l'un des facteurs les plus importants de la curiosité [183]. Dans le cadre de ses études, ce chercheur a tenté de caractériser la nouveauté qui représente pour lui une notion centrale à l'émergence de l'état de curiosité. Par exemple, il décrit trois facettes de la nouveauté face à la mémoire d'une expérience :

- Une nouveauté complète : pour une expérience qui n'a jamais été vécue ;
- Une nouveauté à court terme : pour une expérience analogue vécue dans les dernières minutes ;
- Une nouveauté à long terme : pour une expérience analogue vécue il y a déjà plusieurs jours.

Ce concept de nouveauté, basé sur le temps lié à la mémoire, nous semble prometteur en tant qu'heuristique dans le parcours de l'agent. En effet, il serait envisageable de considérer une mémoire des cases visitées afin de quantifier une certaine nouveauté.

Complexité La complexité est étudiée par D. E. Berlyne au travers de tâches basées sur la reconnaissance de formes géométriques. D'après ce chercheur, plus une forme présente d'éléments, ou encore de dimensions, plus elle est décrite comme complexe [183, Page 54].

Ce concept de complexité, basé sur la reconnaissance de formes géométriques, nous paraît être une piste à approfondir en tant qu'heuristique de perception de l'agent. En effet, il serait envisageable d'adapter ce concept et de prendre en compte, par exemple, le nombre d'assets différents dans la vision de l'agent afin de quantifier une certaine complexité du niveau lors de son parcours.

Sécurité Nous proposons également d'introduire une variable propre au genre FPS/TPS, à savoir la notion de sécurité : être à couvert ou à découvert par rapport aux ennemis, ou encore se retrouver en ligne de mire des tirs. Ainsi, dans cette catégorie de jeu, la géométrie du niveau permet de se cacher des adversaires et de leurs projectiles. Les joueurs, en raison d'une caméra en première personne, peuvent seulement percevoir l'environnement dans une direction donnée. Ils se servent donc de la géométrie à leur avantage pour être en sécurité d'un côté et tirer sur les opposants de l'autre, par exemple avec la mécanique de couverture.

Nous pensons qu'un aspect important de la perception de sécurité dans un niveau peut être lié à la visibilité qu'il offre. Ce facteur permet aux joueurs, par exemple, d'évaluer le niveau sur une longue distance et de tirer sur un ennemi éloigné, ou encore de réagir rapidement à un adversaire apparaissant soudainement devant eux dans des niveaux plus étriqués et plus encombrés. Nous pouvons nous référer à notre formalisation des fortins concernant ces mécaniques de jeu, voir Annexe A page 213.

Ce concept de sécurité, basé sur la visibilité dans le niveau, nous paraît ainsi intéressant en tant qu'heuristique de perception de l'agent. En effet, il serait envisageable d'adapter ce concept et de prendre en compte, par exemple, le nombre de cases visibles dans la vision de l'agent, permettant alors de quantifier une certaine visibilité du niveau lors de son parcours. Nous pourrions ainsi en déduire une certaine valeur de sécurité.

4.4 Notre Search-Based

Les différentes hypothèses, face aux difficultés de développement de notre proposition d'approche, nous ont permis de clarifier la partie recherche de notre méthode *Search-Based*. En effet, l'algorithme

évolutionnaire que nous pensons utiliser, serait, plus précisément, un algorithme génétique. Encoder un niveau avec des zones de probabilités, en tant que génotype, pourrait être implémenté sous forme d'une chaîne de plusieurs nombres entiers, manipulable dans un algorithme génétique. Un gène précis, correspondant alors à un nombre entier, pourrait être, par exemple, un numéro de module à influencer ou une coordonnée pour la zone de probabilité. Dans le cadre d'un tel algorithme évolutionnaire, nous devrons également définir nos différents opérateurs génétiques, comme la mutation ou le croisement, manipulant ainsi la valeur des gènes.

D'après nos hypothèses énoncées précédemment, les éléments constituants notre choix d'approche *Search-Based* seraient, par conséquent, les suivants :

- L'algorithme de recherche : un algorithme génétique ;
- Le moyen d'abstraction, ou d'encodage du niveau : les zones de probabilités ;
- La fonction d'évaluation : la simulation « simplifiée » d'un agent autonome selon trois heuristiques ;
- Notre étape supplémentaire :
  - Génération du niveau avec un opérateur de réparation : le *Wave Function Collapse*, avec des assets de type *greyblock*.

Finalement, le déroulé envisagé de notre méthode pourrait être illustré selon la Figure 4.4. Nous avons décidé de nommer cette approche *Genetic-WFC*.

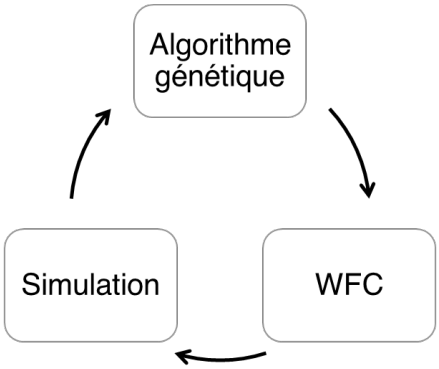


FIGURE 4.4 – Le déroulé de notre méthode : l'algorithme génétique va piloter la création du niveau par WFC, qui sera ensuite évalué par la simulation d'un agent autonome représentant le joueur. La notation du niveau influencera à son tour l'algorithme génétique.

4.5 Conclusion

Pour conclure, nous avons présenté, dans ce chapitre, diverses hypothèses nous permettant de pallier les problèmes de développement inhérents à notre proposition de méthode, à savoir combiner WFC, *Search-Based* et simulation. Nous pensons utiliser des zones de probabilités pour influencer et encoder le niveau du WFC, recourir à des assets *greyblocks* et simplifier la simulation de l'agent pour réduire le temps de calcul, et implémenter trois heuristiques spécifiques pour quantifier l'expérience de jeu lors du parcours du niveau.

Finalement, par rapport aux divers choix précédents, les différentes étapes de la boucle de recherche pour notre méthode itérative, pourraient être détaillées de la façon suivante :

1. Génération du niveau par WFC sur une grille.
2. Évaluation du niveau selon l'expérience de jeu souhaitée :
   - Navigation et découverte de l'espace par l'agent autonome joueur ;
   - Notation du déplacement selon des critères de perception de l'agent.
3. Variation des gènes dans l'algorithme génétique.

Dans les prochains chapitres, nous chercherons à répondre aux interrogations suivantes, qui nous semblent essentielles, et ceci afin de démontrer la faisabilité et l'efficacité de notre méthode :

- Pouvons-nous démontrer la possibilité de contrôler le WFC et de contraindre la génération grâce à des zones de probabilités ?
- Pouvons-nous inclure le WFC dans un algorithme génétique en tant qu'opérateur de réparation ?
- Est-ce que le temps de calcul total requis sera suffisamment acceptable ?
- Est-ce que les valeurs de perception choisies pour l'agent seront-elles pertinentes et suffisamment significatives, nous permettant d'aboutir à des résultats intéressants ?

Nous espérons, également, que l'abstraction de la simulation proposée, nous rapprochera suffisamment de la véritable expérience joueur, permettant d'orienter la génération vers un objectif précis.

Dans le chapitre suivant, nous allons aborder, plus précisément, l'aspect technique de notre approche. Nous y décrirons notre prototype de générateur de niveau finalement réalisé, selon les conjec-

tures proposées précédemment.

Chapitre 5

Genetic-WFC

Contenu

5.1 Introduction

Nous avons précédemment énoncé notre problématique et présenté une méthode de PCG pouvant convenir. Puis, nous nous sommes penchés sur les hypothèses de développement et les difficultés d'une telle approche.

Dans ce chapitre, nous allons décrire notre méthode développée dans le but de répondre aux questionnements soulevés auparavant. Nous allons ainsi présenter notre implémentation appelée *Genetic-WFC* et ses différentes fonctionnalités. Notre approche a été réalisée dans le moteur de jeu *Unity*.

Dans un premier temps, nous nous concentrerons sur une présentation générale du déroulé de notre *Genetic-WFC*. Puis, par la suite, nous détaillerons respectivement chaque partie composant notre méthode, à savoir, le *Wave Function Collapse*, l'algorithme génétique et l'évaluation par simulation.

5.2 Présentation générale

Le *Genetic-WFC* est un pipeline de génération procédurale qui combine une approche *Search-Based*, à savoir un algorithme génétique et une évaluation par simulation, avec une méthode constructive, le *Wave Function Collapse*, afin de générer des niveaux ciblant des expériences de jeu spécifiques. Le WFC s'intègre comme opérateur de réparation dans l'algorithme génétique, et génère ainsi uniquement les niveaux candidats tout en respectant des contraintes spécifiques de placement d'assets. Il est également possible d'effectuer une seconde étape de génération en utilisant le résultat *greyblock*, obtenu par la méthode itérative, afin de le transformer en un niveau final composé d'assets graphiques. La Figure 5.1 page suivante représente une vue d'ensemble de ces différentes étapes, que nous allons détailler par la suite.

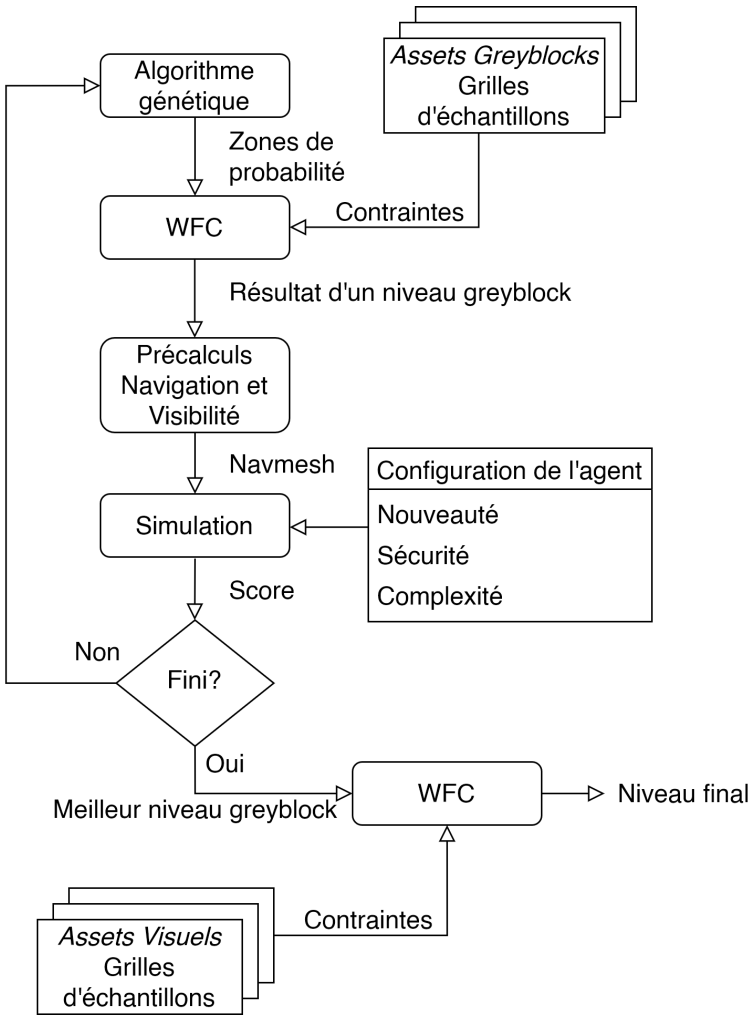


FIGURE 5.1 – Description de notre méthode *Genetic-WFC*.

Comme nous pouvons le constater, en haut de la Figure 5.1, notre algorithme génétique pilote le WFC dans le but de générer les niveaux, à l'aide de zones de probabilités influençant la sélection d'assets, comme décrit dans la Section 5.4 page 118. Le WFC extrait automatiquement les contraintes de placement des objets à l'aide de grilles d'exemple. De plus, comme nous le détaillons dans la Sous-Section 5.3.4 page 113, nous ne manipulons que des assets de type *greyblock* à ce stade de notre pipeline. En effet, vu que le temps de calcul du WFC augmente avec le nombre de modules, et comme de nombreux assets graphiques ont la même utilité en termes de *level design*, nous préférons générer ainsi, tout d'abord, le niveau en utilisant un nombre restreint de modules, c'est-à-dire nos *greyblocks*.

Par la suite, chaque cellule de la grille d'un niveau *greyblock* résultant du WFC, est complétée

d'informations sur sa navigabilité et sa visibilité. Ces données sont ensuite utilisées par le joueur synthétique, et ceci au cours de l'étape de simulation, détaillée dans la Section 5.5 page 125. Notre agent évalue le niveau en fonction d'une configuration spécifique, plus précisément à l'aide de pondérations paramétrables par nos soins concernant les valeurs de perception. Ces dernières heuristiques portent sur les notions de nouveauté, de sécurité et de complexité.

Les précédentes étapes sont effectuées de manière itérative, avec une population de niveaux candidats, et ceci tant que nous ne répondons pas à un critère d'arrêt. Dans le cas de nos expérimentations, nous avons choisi d'effectuer un nombre fixe de boucles d'itération.

Pour finir, le niveau *greylock*, ayant le meilleur score, peut être repris dans une seconde étape de génération à l'aide d'un autre WFC. Nous pouvons définir ce principe comme une génération « multicouche ». Cette étape supplémentaire est réalisable, car chaque module *greyblock* correspond à une certaine catégorie d'assets avec une hauteur spécifique, par exemple des escaliers allant du sol vers le premier niveau de hauteur. Comme expliqué dans la Sous-Section 5.3.4 page 113, nous utilisons ces catégories de modules, ainsi que des contraintes d'adjacence relatives à d'autres assets graphiques, pour remplacer les *greyblocks* afin de générer le niveau final. Il en découle que la structure d'un tel niveau correspond au niveau *greyblock* initial, que ce soit, par exemple, en termes de navigabilité ou de visibilité. De plus, cette étape peut être exécutée plusieurs fois, permettant d'obtenir des variations visuelles d'un même niveau, selon le nombre d'assets graphiques appartenant à une même catégorie et les relations d'adjacence possibles.

5.3 Wave Function Collapse

Notre implémentation de l'algorithme du *Wave Function Collpase* est basée sur une grille 2D carrée, dont chaque cellule représente un emplacement pour un module possible. Nous utilisons également des objets 3D associés à nos modules. Notre version est basée sur le déroulé du WFC détaillé dans l'état de l'art, le fonctionnement de base étant identique, voir Sous-Section 2.3.3.1 page 55.

Il est important de noter que le WFC que nous avons développé peut être considéré comme constructif, car nous n'incluons pas le concept de *backtracking* au sein même de cet algorithme. Celui-ci génère en une seule fois le contenu pour chaque exécution.

Nous pouvons également préciser que le déroulement de notre WFC est déterministe, c'est-à-

dire qu'avec les mêmes paramètres initiaux, le résultat sera toujours identique. Pour ce faire, nous définissons, en amont, une graine aléatoire pour le générateur de nombres pseudo-aléatoires, qui est utilisé lors du choix d'un module précis dans l'emplacement avec l'entropie minimale.

Nous allons à présent décrire les multiples fonctionnalités propres à notre implémentation du WFC.

5.3.1 Modèle Simple Tiled avec extraction automatique des contraintes

Nous avons implémenté le modèle *Simple Tiled* du *Wave Function Collapse*, car il s'agit, pour nous, de l'algorithme le plus efficace à mettre en place et qui nous offre un contrôle direct sur les contraintes d'adjacence locales pour chaque module.

Dans le cadre de notre modèle *Simple Tiled*, nos contraintes d'adjacence sont automatiquement extraites à partir d'exemples de niveau créés à l'aide de modules disposés manuellement sur des grilles 2D, voir Figure 5.3 page 110. Ces modules sont, de surcroît, placés selon une rotation précise, à 90 degrés près autour de l'axe vertical. Chaque module présent est alors répertorié selon son objet 3D associé, et pourra être reproduit dans une grille de résultat avec une nouvelle rotation, voir Figure 5.4 page 110. Pour chacun des modules présents sur une grille, seuls quatre voisins mitoyens, c'est-à-dire à gauche, à droite, en haut et en bas, sont pris en considération pour les relations, autrement dit les contraintes d'adjacence. Ces relations entre modules sont calculées et s'adaptent selon la rotation de chacun, comme spécifié dans la Figure 5.2 page 109.

Dans notre implémentation du WFC, celui-ci prend en compte la fréquence d'apparition des assets disposés dans la grille d'exemple, également appelée grille d'échantillonnage. Ces fréquences influencent directement la sélection aléatoire, c'est-à-dire la probabilité d'être choisi, d'un module dans l'emplacement avec l'entropie minimale. Par conséquent, plus un module a été placé dans une grille d'échantillonnage, plus ses chances d'être sélectionné sont élevées dans le déroulement du WFC. Afin de mieux se rapprocher du nombre d'apparitions des assets présents dans la grille d'exemple, nous avons également introduit la notion suivante : plus un module a déjà été sélectionné dans le niveau résultat du WFC, moins il aura de chance d'être placé par la suite.

Nous pouvons également restreindre la sélection d'un module dans la grille de résultat, à l'aide d'un nombre minimal et maximal d'apparition spécifique à un objet 3D. Cette fonctionnalité est particulièrement utile pour définir l'émergence d'un point de départ unique dans le niveau.

Il convient également de préciser que la restriction du nombre minimal et maximal d'apparition d'un module ainsi que le respect de la fréquence d'apparition, mentionnés ci-dessus, représentent des contraintes non spatiales. Dans notre implémentation, ces critères influencent uniquement la probabilité de sélection d'un module et de sa rotation dans le WFC.

En définitive, nous pouvons exécuter l'algorithme WFC sur une grille de résultat de taille variable pour générer un niveau qui respecte les contraintes d'adjacence extraites en amont, ainsi que nos contraintes non spatiales. Lors de l'exécution du WFC dans une grille de résultat, les modules extraits précédemment pourront y être placés dans chaque cellule et selon 4 possibilités de rotation. La propagation des contraintes et la sélection des modules et de leur rotation limiteront alors peu à peu les choix disponibles jusqu'à proposer, par case, qu'un seul module et qu'une seule rotation associée, si aucune violation de contraintes d'adjacence n'est survenue.

Nous allons détailler, ci-dessous, les différents paramètres propres à nos modules, implémentés dans notre WFC.

Une instance de module, placée sur une grille d'exemple, possède plusieurs propriétés :

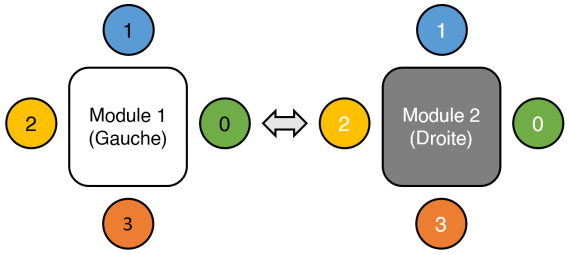
- L'objet graphique 3D à afficher ;
- La rotation à 90 degrés près, soit 4 possibilités ;
- Une taille précise dans le cas d'un *BigTile*, à savoir un module dont la dimension est supérieure à une cellule ;
- L'éventualité de prendre automatiquement toutes les rotations en compte dans le calcul des relations pour ce module précis, instancié sur la grille. Cela permet de rajouter des possibilités plus facilement sans représenter tous les cas de rotations possibles sur une grille d'exemple. Il est important de spécifier que prendre en considération toutes les rotations dans les relations est principalement utilisé pour le module d'« air », afin de ne pas limiter les rotations possibles des autres modules dans la grille de résultat.

Pour les modules, d'autres paramètres, uniques à l'objet graphique 3D, sont également modifiables :

- Un nombre minimal et maximal d'apparition, à répercuter dans le résultat du WFC ;
- Une catégorie, essentielle pour le fonctionnement de la multicouche, afin de remplacer ce module dans un autre WFC ;
- L'option de ne pas afficher l'objet 3D dans la grille de résultat, utile pour cacher le module d'« air » ;
- Les réglages concernant l'établissement du mesh de navigation [1] pour la simulation, par exemple la hauteur et la surface navigable de l'objet 3D.

Dans notre implémentation, après l'extraction des contraintes, chaque module répertorié, est complété par les informations suivantes :

- Un numéro d'identification, ou ID ;
- Son nombre initial de présences dans les grilles d'exemple, pour calculer la fréquence d'apparition ;
- La liste des contraintes d'adjacence envers chaque module répertorié, lui-même inclus.



Module 1 (Gauche)	0				1				2				3			
Module 2 (Droite)	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
Relation	***0***	***0***	***1***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***

FIGURE 5.2 – Exemple d'une relation d'adjacence calculée automatiquement entre deux modules, selon une rotation précise pour chacun.

1. Un mesh de navigation est une structure de donnée utilisée en intelligence artificielle permettant de représenter les espaces navigables d'un environnement 3D, afin de permettre aux agents de s'y déplacer [186].

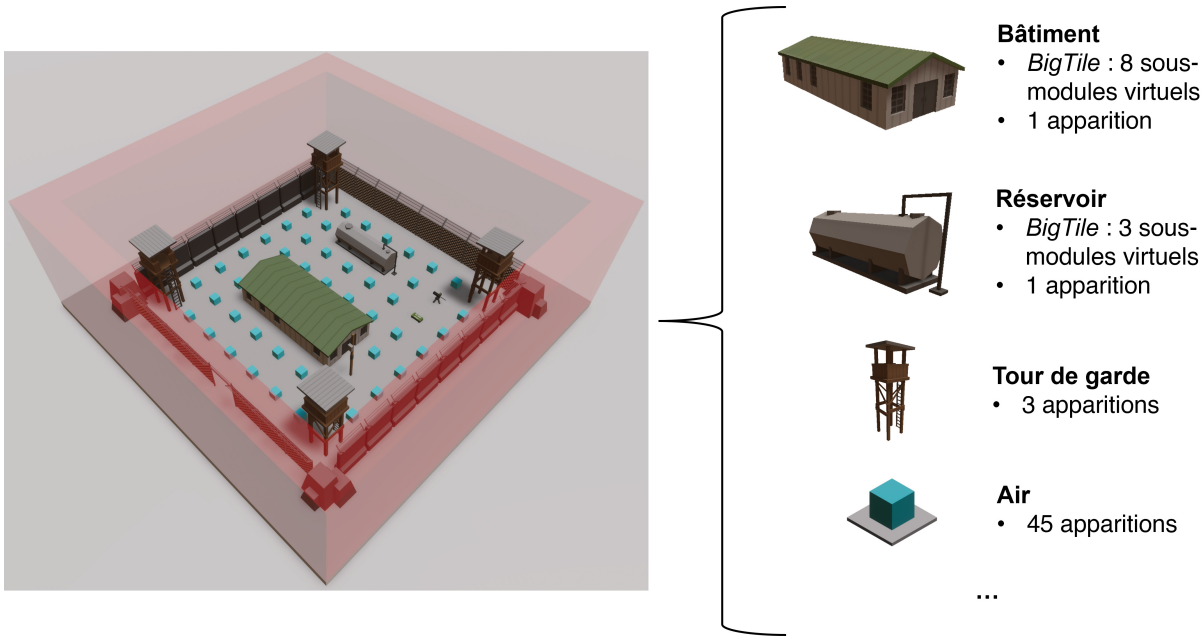


FIGURE 5.3 – Exemple d'extraction automatique, pour une grille d'échantillonnage, des modules présents. À gauche, la grille d'exemple et à droite les modules répertoriés et complétés par des informations supplémentaires, par exemple le nombre d'apparition. Dans cette grille, un total de 21 modules a été extrait et regroupé, dont 11 sous-modules virtuels. Il en résulte donc 84 possibilités par cellule pour la grille de résultat du WFC, grâce aux 4 rotations possibles.

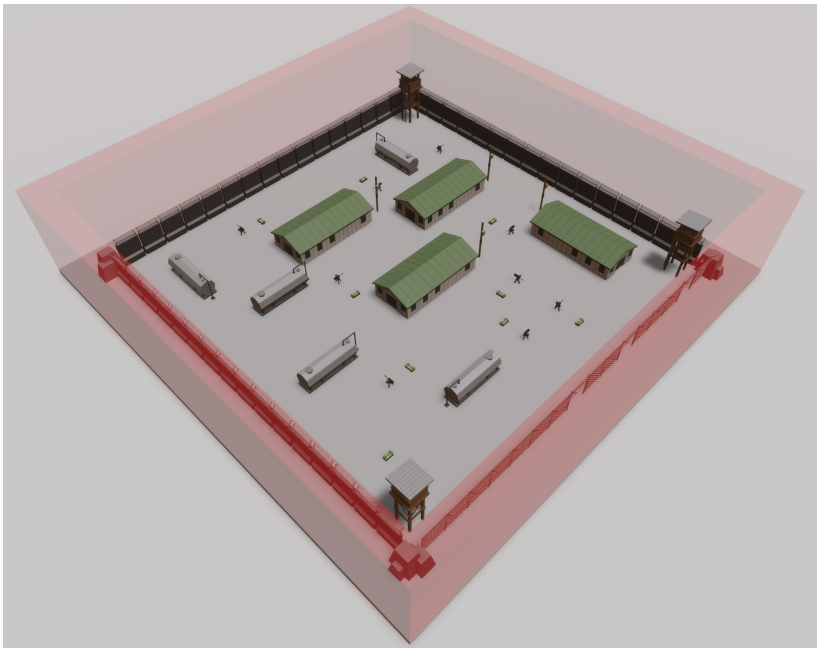
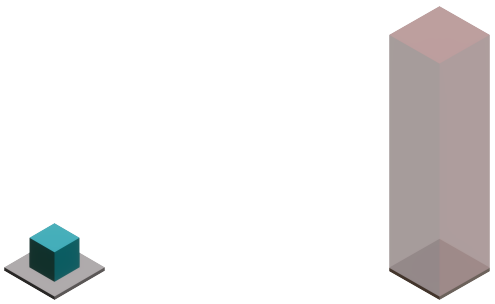


FIGURE 5.4 – Exemple de résultat du WFC, réalisé sur une grille de 20x20, à partir de la grille d'échantillonnage de la Figure 5.3.

5.3.2 Air et bordure



(a) Module d'« air » (b) Module de « bordure »

FIGURE 5.5 – Modules supplémentaires employés dans notre implémentation du WFC.

Afin d'extraire les contraintes d'adjacence d'un niveau conçu manuellement, nous utilisons deux modules spéciaux placés automatiquement, voir Figure 5.5. Ces modules ont été mis en place initialement pour le bon fonctionnement de notre WFC avec des objets 3D.

Air Tout d'abord, dans le cas de l'extraction automatique des contraintes, nous faisons apparaître un module d'« air » dans chaque cellule vide de la grille d'exemple afin de créer explicitement un lien entre les cellules vides et les modules présents. En effet, le principe de case vide n'existe pas pour le déroulement du WFC, car il a été conçu pour la reproduction de texture. Dans ce contexte, l'idée d'une case de la grille non définie n'a pas lieu d'être. Il convient de noter que nous pouvons également utiliser différents types de modules d'« air », par exemple, pour créer des contraintes entre des assets qui ne sont pas directement voisins, mais également pour faciliter la mise en place d'espaces spécifiques à l'intérieur de bâtiments. Ce module d'« air » est généralement caché dans la grille de résultat du WFC.

Bordure Nous plaçons également un module supplémentaire sur les bords, c'est-à-dire les limites du niveau, appelé un module dit de « bordure ». D'un point de vue de *level design*, ce module spécifique permet de restreindre l'espace du niveau ou dans le cas contraire d'autoriser le déplacement en dehors, autour de la carte par exemple. Cependant, dans notre cas d'application, la bordure permet d'indiquer à des modules de se placer sur les bords du niveau, par exemple des clôtures. Il s'agit également de mieux discerner la délimitation du niveau, et d'autre part, de mieux visualiser la portée de la perception

et du déplacement de l'agent à l'intérieur de la grille. En outre, le module de « bordure » sert à contenir les *BigTiles* à l'intérieur de la grille et à éviter ainsi tout dépassement.

5.3.3 BigTiles

Certains assets peuvent être plus grands qu'une seule cellule de la grille. Afin de conserver une grille uniforme, ces modules spécifiques, que nous nommons *BigTiles*, sont divisés en sous-modules virtuels de la taille d'une case. Ainsi, chaque cellule couverte par la surface de l'asset est considérée comme un sous-module. Tous les sous-modules obtenus représentent des cellules occupées par le *BigTile* et possèdent leurs propres relations avec les cellules adjacentes. Dans notre découpage, le sous-module situé en bas à gauche, selon les deux axes 2D de la grille, est considéré comme le point pivot du *BigTile*.

Il convient toutefois de préciser qu'un seul *BigTile* comme celui de la Figure 5.6 peut générer 8 sous-modules, avec 4 rotations possibles par module, et donc 32 nouvelles possibilités pour chaque cellule de la grille de résultat.

Plus le nombre de modules possibles augmente, plus le temps de calcul du WFC augmente également, comme décrit dans l'expérience 1, voir Sous-Section 6.2.1 page 135. De ce fait, l'inclusion de nombreux *BigTiles* peut impacter de manière significative le temps de calcul du WFC.

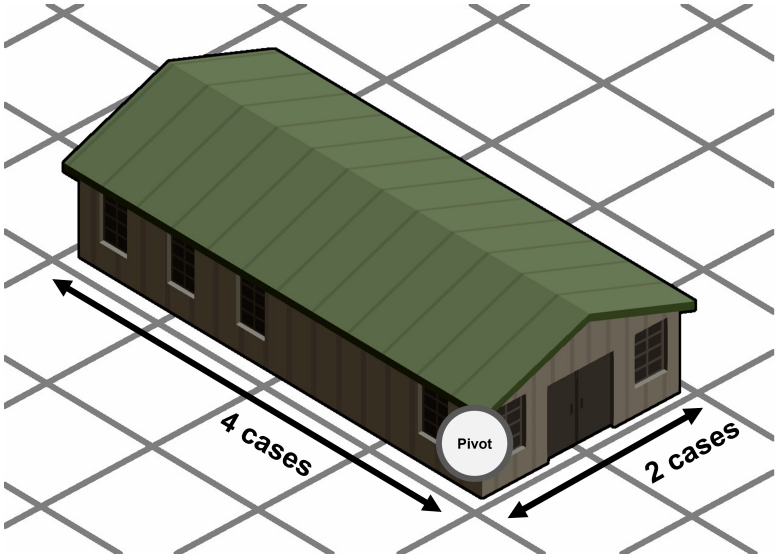


FIGURE 5.6 – *BigTile* : les assets dépassant la taille d'une cellule de la grille sont décomposés en sous-modules. Dans cet exemple, la taille est de 2 sur 4, donc 8 sous-modules virtuels sont créés.

5.3.4 Greyblock et multicouche

Module							
Nom	EscalierH2	EscalierH1	CubeH2	CubeH1	Point de départ	Bordure	Air
ID	0	1	2	3	4	5	6

TABLE 5.1 – Les différents modules de *greyblocks* placés puis extraits de la grille d'exemple Figure 5.7, avec un numéro d'identification associé, ou ID. H signifie hauteur.

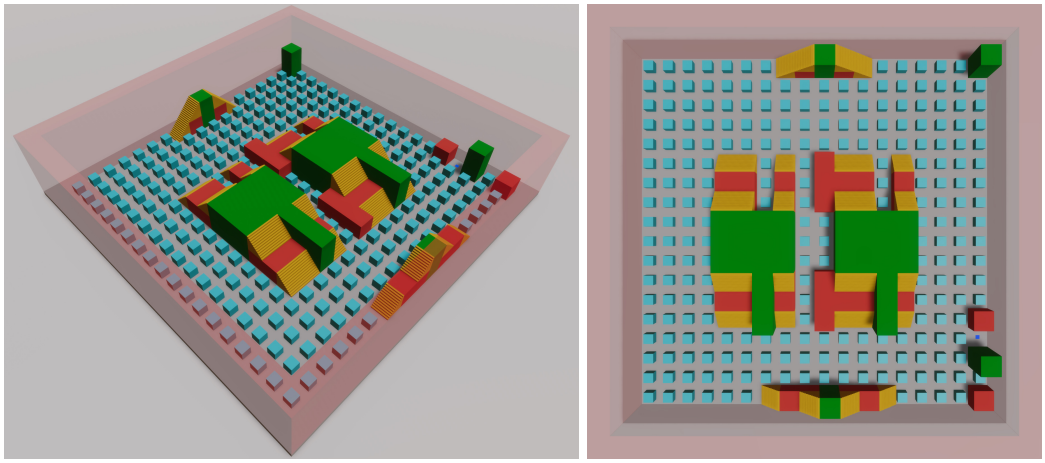


FIGURE 5.7 – Grille d'échantillonnage pour le WFC, avec les modules *greyblocks* détaillés Tableau 5.1. Cet exemple contraint le placement des escaliers de façon pertinente afin d'éviter que ces modules ne mènent nulle part. Le point de départ est réglé pour n'être présent au minimum et au maximum qu'une fois dans le résultat.

Comme nous l'avons déjà indiqué dans la Sous-Section 5.3.3 page précédente, plus le nombre de modules augmente, plus le temps de calcul du WFC croît également. Cependant, de nombreux objets graphiques possèdent la même signification du point de vue du gameplay dans le niveau. De ce fait, nous avons introduit la possibilité de générer des niveaux en deux étapes, et de nous appuyer sur le principe de *greyblocking*, une étape intermédiaire de prototypage utilisée par les *level designers*. Les *greyblocks* constituent des blocs de construction intermédiaire, avant une couche visuelle, uniquement

dans un but fonctionnel. Nous pouvons nous référer au Tableau 5.1 page précédente, à la Figure 5.7 page précédente et à la Figure 5.8 qui illustrent les *greyblocks* que nous employons.

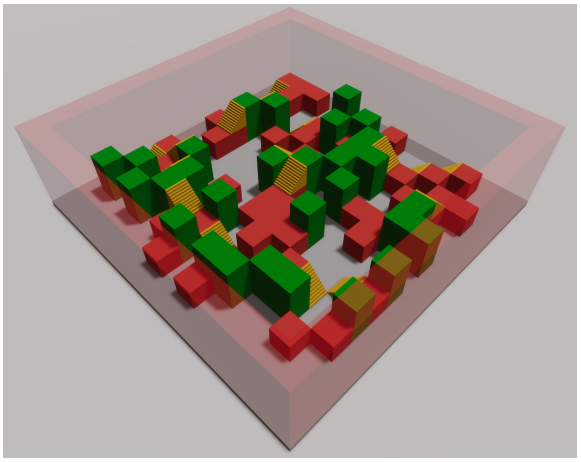


FIGURE 5.8 – Un exemple de résultat du WFC en utilisant la grille d'échantillonnage Figure 5.7 page précédente.

En utilisant uniquement ces *greyblocks* dans une première étape de génération, nous pouvons ensuite reprendre le résultat et le réutiliser dans une seconde couche de WFC afin d'obtenir un niveau avec des assets plus visuels. De plus, il est intéressant de noter, que cette seconde étape peut être relancée afin de créer diverses variations visuelles avec la même structure de niveau, voir Figure 5.10 page suivante.

Le fonctionnement de notre multicouche repose sur une étape spécifique lors de l'initialisation de la grille d'un second WFC. Nous supprimons simplement, pour chaque cellule, tous les choix qui ne sont pas de la même catégorie que le module *greyblock* déjà présent. Ainsi, nous pouvons utiliser le résultat d'une première couche afin de contraindre un second WFC pour générer un niveau final.

De plus, cette technique nous permet, dans le cadre de la recherche évolutionnaire, d'économiser du temps de calcul en utilisant uniquement les *greyblocks* dans la boucle itérative. Le résultat final de l'optimisation peut alors être repris dans une seconde étape de génération dans le but d'obtenir des niveaux plus visuels.

Il est intéressant de noter que notre multicouche peut utiliser des *BigTiles* dans la seconde étape, afin d'offrir plus de diversités dans le choix des modules, voir Figure 5.9 page suivante.

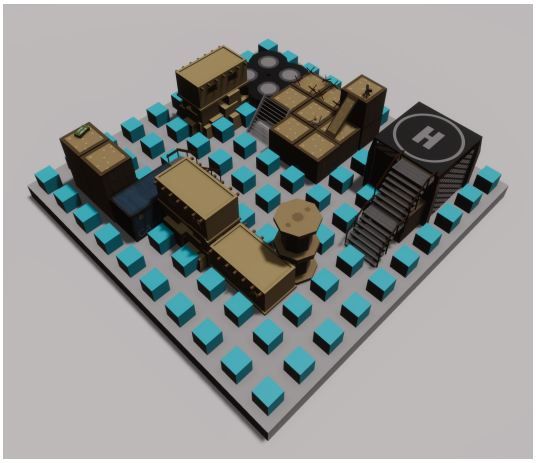


FIGURE 5.9 – Exemple de grille d'échantillonnage avec des assets visuels, dont des *BigTiles*, pouvant être utilisés dans une seconde couche de WFC.

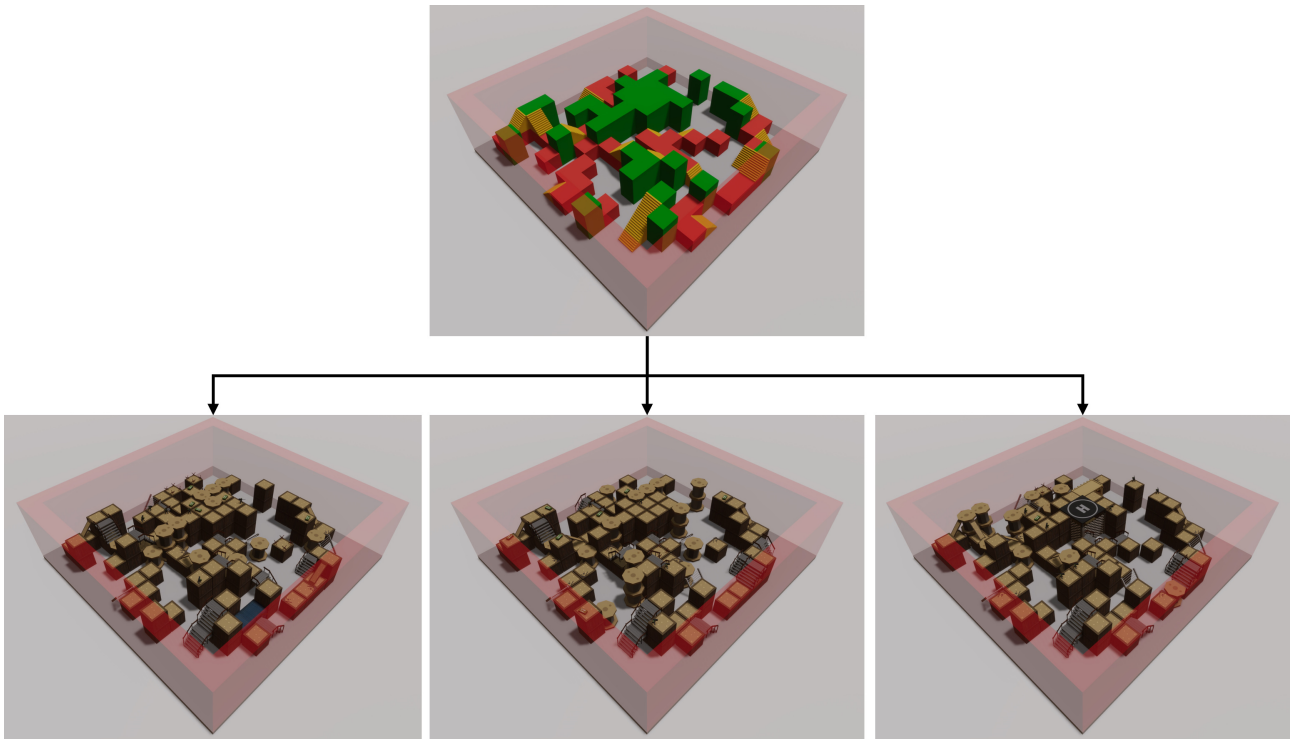


FIGURE 5.10 – Exemples de variations visuelles obtenues dans une seconde étape avec une structure initiale en *greyblock*. Le remplacement des *greyblocks* est possible grâce à la catégorie associée. Nous pouvons également mentionner que plusieurs grilles d'échantillonnage ont été utilisées dans cet exemple pour les assets visuels afin de compléter suffisamment les relations possibles.

5.3.5 Entropie et sélection des modules

Remarque : Les possibilités, par emplacement de cellule dans la grille du WFC, se réfèrent aux modules et ses rotations associées encore disponibles.

Lors du déroulé de l'étape d'observation du WFC, les cellules qui ne sont pas déterminées possèdent encore plusieurs possibilités, valides du point de vue des relations d'adjacence, et ceci en termes de choix de modules et de rotations. Chaque possibilité restante se voit attribuée dans ce cas la même probabilité de sélection correspondant à la fréquence d'apparition du module répertorié dans les grilles d'échantillonnage.

Il nous paraît également judicieux de préciser que cette probabilité de sélection est influencée par la valeur minimale et maximale de présence souhaitée du module dans la grille de résultat. Par exemple, si la valeur maximale est atteinte, la probabilité des possibilités du module et de ses rotations sera toujours fixée à 0, quelle que soit sa fréquence d'apparition. Dans le cas où une valeur minimale est souhaitée, la probabilité est artificiellement accrue tant que cette valeur n'est pas atteinte. Dans notre implémentation, les probabilités, associées aux choix des modules et de leur rotation pour une cellule, sont normalisées.

Ces probabilités de sélection nous permettent de calculer, pour chaque cellule de la grille, une valeur d'entropie. L'emplacement avec l'entropie minimale, c'est-à-dire l'emplacement avec l'espace des possibles le plus réduit, est sélectionné et se voit attribuer une seule solution parmi les possibilités restantes. Ce choix dépend des probabilités associées à ces possibilités, disponibles dans cet emplacement. Si plusieurs emplacements sont en lice, avec une entropie similaire, l'un sera choisi au hasard.

Pour chaque cellule ayant comme possibilités $N > 1$, nous calculons l'entropie à l'aide de la formule suivante :

$$H = -\sum_{i=1}^{N} \left| \frac{1}{N} - p_i \right|$$

Où N est le nombre de possibilités restantes dans la cellule, et où p_i est la probabilité de sélection associée à une possibilité restante dans cette même cellule.

Les cases de la grille où les possibilités sont équiprobables, auront l'entropie la plus élevée, soit zéro. Comme il nous suffit de sélectionner l'entropie la plus faible, nous n'utilisons pas le logarithme original de l'entropie de Shannon [2] pour gagner du temps de calcul.

2. Formule de l'entropie de Shannon : $-\sum_{i=1}^{n} p_i \log(p_i)$ [131].

5.3.6 Zones de probabilités

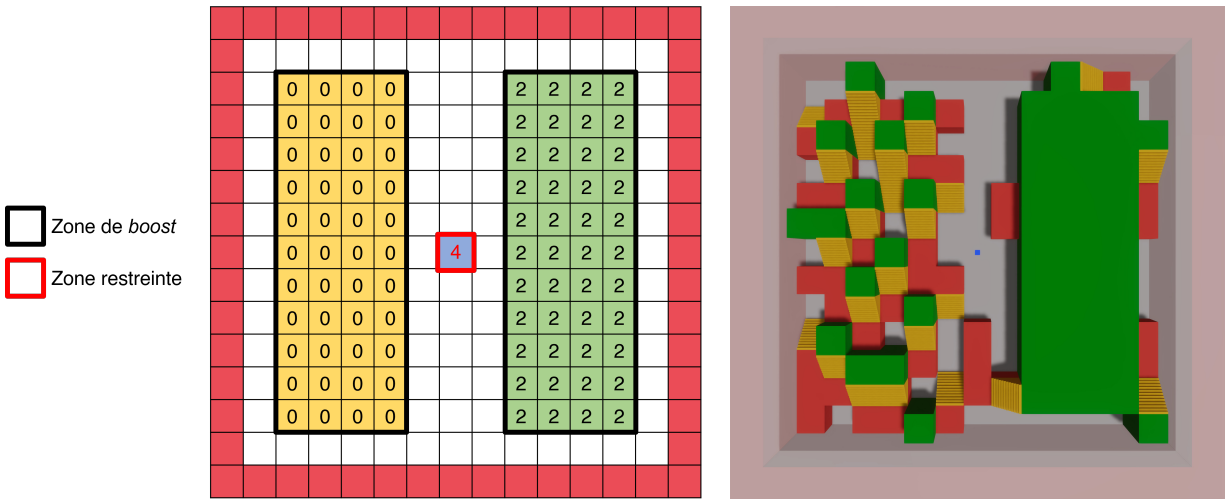
Afin de permettre à l'algorithme d'optimisation de piloter le WFC, nous avons ajouté des zones de probabilités, qui augmentent la probabilité de sélection d'un module selon une certaine rotation dans une région spécifique de la grille. Plus précisément, la probabilité de cette possibilité est substituée par une valeur arbitraire très élevée, mais uniquement dans le cas où ce choix était déjà envisageable, c'est-à-dire si la valeur de sa probabilité était supérieure à 0. De cette façon, nous pouvons influencer localement le module et la rotation sélectionnés par le WFC sans outrepasser le déroulement de l'algorithme. Nous ne pouvons donc pas forcer un module ou une rotation qui enfreindrait une contrainte d'adjacence.

Remarque : Il nous paraît important de préciser, que lors de la sélection des possibilités, si une zone a voulu augmenter la probabilité d'un choix qui n'était pas disponible, la sélection des autres possibilités fonctionnent comme détaillées dans la Sous-Section 5.3.5 page précédente.

5.3.7 Zones d'assets initiaux

Dans le cadre d'une *mixed-initiative*, à savoir une collaboration entre machine et utilisateur, nous avons également la possibilité d'indiquer et de contraindre un placement de module et/ou sa rotation sur la grille, en amont du WFC. Il s'agit ici d'utiliser des zones d'assets initiaux, ou zones de restriction, qui limitent l'espace des possibles d'une ou plusieurs cellules à un module spécifique avec ou non une rotation précise. Ainsi, la génération complète par la suite le résultat selon les choix imposés, en tenant compte de la faisabilité. Cette fonctionnalité agit en tandem avec les zones de probabilités.

Remarque : Il est important de souligner que les zones de restriction peuvent amener des erreurs de génération dans le WFC, et de ce fait, cette fonctionnalité ne sera pas utilisée dans les expériences du Chapitre 6 page 133.



(a) Schéma des zones utilisées, l'ID correspond au module associé à la zone. Pour simplifier, la rotation est implicite.

(b) Résultat du WFC selon les zones spécifiées.

FIGURE 5.11 – Exemple d'utilisation de deux zones de probabilités et d'une zone d'asset initial avec le WFC, en utilisant la grille d'échantillonnage vue Figure 5.7 page 113. Le point de départ est forcé au milieu du niveau grâce à la zone de restriction, alors que les zones de probabilités ont influencé au mieux le placement des modules et rotations souhaités.

5.4 Algorithme génétique

Cette partie concerne à présent l'algorithme génétique, qui guide la génération de niveau par le WFC, vers une expérience de jeu spécifique. Le déroulé de notre méthode d'optimisation se décrit à l'aide de l'Algorithme 1 page suivante, la boucle itérative est exécutée $NbMax$ fois. Nous utilisons également la bibliothèque *GeneticSharp* [187] permettant d'implémenter la partie génétique en langage C#.

Données : Une 1^{re} population P de taille Pop_{max}, initialisée avec des chromosomes aléatoires
Résultat : Une population P de Pop_{max} niveaux évolués

```
tant que NbMax itération n'est pas atteint faire
    P_n = Sélection par tournoi de Pop_min p ∈ P
    P_{n+1} = ∅
    pour tous i ∈ [1, taille(P_n)] faire
        si i est pair alors
            j = i − 1
        sinon
            j = i + 1
        fin
        si j ∈ [1, taille(P_n)] alors
            Prendre p_i, p_j dans P_n
            si rnd ∈ U(0,1) < p_cross alors
                c = Croisement(p_i, p_j)
            sinon
                c = p_i
            fin
            c = Mutation(c, p_mut)
            l = GénérationWFC(c)
            Évaluation(l)
            c = Réencodage(l)
            Ajouter c à P_{n+1}
        fin
    fin
    Ajouter les meilleurs (Pop_max - Pop_min) de P à P_{n+1}
    P = P_{n+1}
fin
```

Algorithme 1 : Déroulé de notre algorithme génétique.

5.4.1 Encodage : la représentation du chromosome

Afin de permettre cette recherche d'optimisation, nous avons introduit un moyen de contrôler le WFC et de l'encoder dans un chromosome, à savoir le génotype. Pour ce faire, nous utilisons les zones de probabilités, détaillées dans la Sous-Section 5.3.6 page 117. Nous influençons directement la probabilité de sélection d'un module et d'une rotation précise chaque fois qu'un tirage aléatoire est initié par le WFC sur la case avec la plus basse entropie.

Plus précisément, nous avons décidé d'utiliser dans notre méthode, une zone de probabilité par cellule de la grille de résultat du WFC. Nous n'avons donc pas besoin d'encoder une taille ou des coor-

données associées à une zone. Nous avons également défini une valeur arbitraire très élevée, commune à toutes les zones de probabilités ; cette valeur va nous servir à modifier les probabilités de sélection des possibilités. Dans notre implémentation, les gènes composant la séquence du chromosome encodent uniquement une seule valeur représentant une possibilité spécifique à influencer, et qui correspond à un numéro d'identification d'un module et à une rotation précise, voir Figure 5.12 et voir Figure 5.13. Chaque gène correspond ainsi à une zone de probabilité précise. La taille, soit le nombre de gènes, de notre chromosome correspond, de ce fait, au nombre exact de cellules dans la grille. En conséquence, l'algorithme génétique choisit un module et sa rotation à influencer pour chaque cellule de la grille. Le WFC traduit ensuite ces choix en un niveau qui respecte les contraintes d'adjacence.

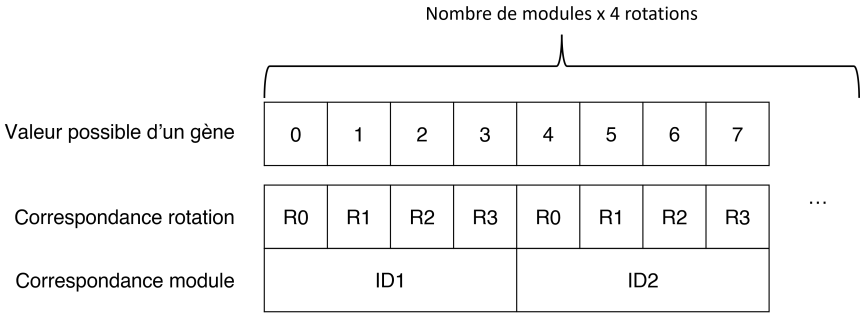


FIGURE 5.12 – Illustration des différentes valeurs possibles d'un gène, en fonction du nombre de modules extraits des grilles d'échantillonnage. Une valeur possible correspond à une rotation et au numéro d'identification, à savoir l'ID, d'un module précis. La correspondance des rotations est : R0 à 0°, R1 à 90°, R2 à 180° et R3 à 270°. Par exemple, la valeur 2 d'un gène va correspondre à la rotation R2 et à l'asset ID1.

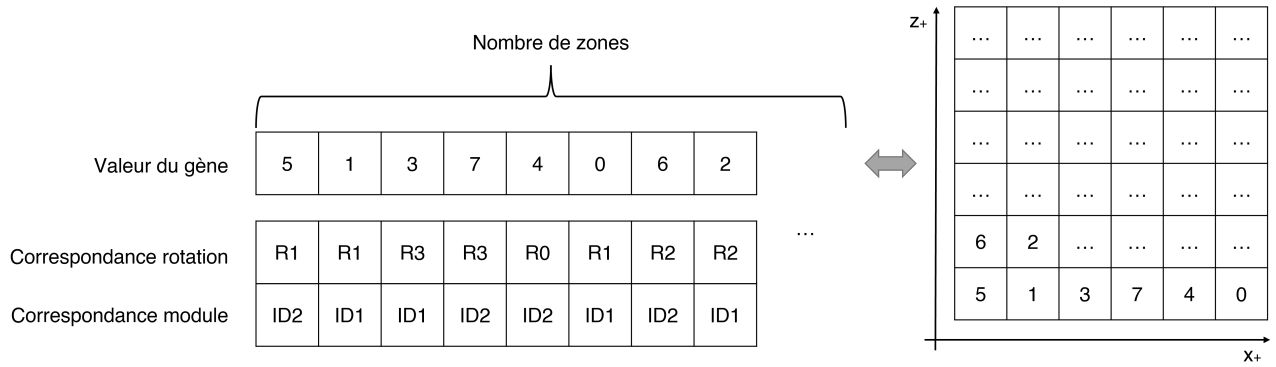


FIGURE 5.13 – À gauche, un exemple d'un chromosome, soit une séquence de gènes égale au nombre de zones, représentant un individu de la population. Les valeurs possibles de ces gènes sont présentées dans la Figure 5.12. À droite, un aperçu de la distribution de ces gènes, correspondant à des zones de probabilités, sur la surface de la grille du WFC.

5.4.2 Déterminisme et réencodage

Afin de garantir une retranscription déterministe du génotype au phénotype, nous utilisons la même graine de générateur aléatoire chaque fois que nous concevons un niveau. Cependant, pour disposer d'un opérateur de croisement efficace, nous devons également réencoder notre niveau résultant du WFC dans son chromosome après l'évaluation, voir Algorithme 1 page 119, ligne 20.

Nos chromosomes nous permettent d'augmenter la probabilité, dans chaque cellule, d'un module avec une rotation spécifique. Si la possibilité voulue d'une zone de probabilité ne peut être placée, car elle enfreint les règles d'adjacence, un autre module avec une rotation associée sera choisi par le WFC. Comme nous utilisons toujours la même graine du générateur aléatoire pour générer un niveau, ce choix est toujours identique pour une cellule donnée d'un chromosome donné. Toutefois, cela n'est pas le cas après un croisement, car le WFC choisit un module et sa rotation par rapport au nombre déjà présent de ce module dans la grille, afin de correspondre à la fréquence de l'asset dans les grilles d'échantillonnage. De ce fait, la suite de nombres produite par le générateur de nombres pseudo-aléatoires, pour le tirage des probabilités, diffère dans un nouveau niveau recomposé par croisement.

Afin d'éviter cette perte de déterminisme, nous réencodons simplement le phénotype dans le génotype après la génération du niveau par le WFC. Ainsi, pour chaque cellule, nous modifions le gène associé dans le chromosome selon le niveau généré finalement par le WFC. En effet, nous mettons à jour les valeurs des zones selon les modules et les rotations, effectivement présents dans le résultat, comme s'il s'agissait du choix du chromosome en premier lieu, voir Figure 5.14 page suivante.

Nous montrons dans la section Sous-Section 6.3.2.3 page 144 et la Figure 6.4 page 144 que sans ce réencodage, l'optimisation de la génération du WFC est nettement plus faible, voir plus précisément la courbe *Genetic-WFC NR*, NR signifiant Non Réencodé.

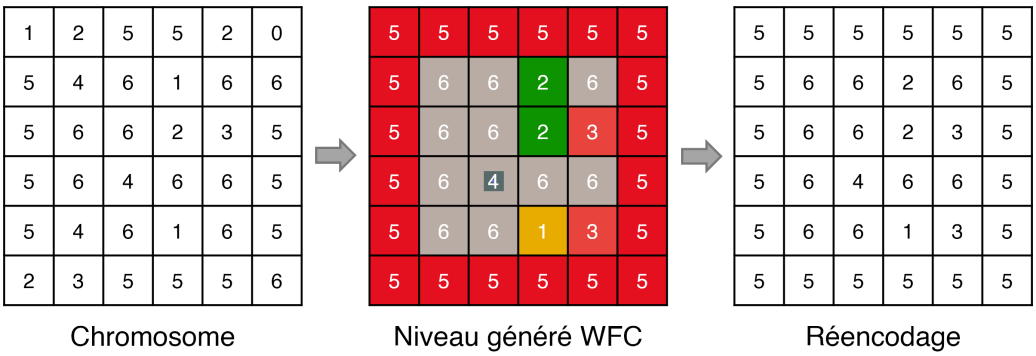


FIGURE 5.14 – Exemple de réencodage. La représentation des gènes est ici simplifiée, seul le numéro d'identification du module est illustré. L'ID correspond aux valeurs des *greyblocks* figurants dans le Tableau 5.1 page 113.

5.4.3 Opérateurs génétiques

Nous allons à présent détailler nos différents opérateurs génétiques utilisés dans l'ordre d'apparition dans l'Algorithme 1 page 119.

Sélection La liste des futurs parents, à savoir les chromosomes destinés à la reproduction, est déterminée lors de « tournois », voir Algorithme 1 page 119, ligne 2. En effet, cette sélection consiste à organiser des « tournois » entre plusieurs individus choisis au hasard dans la population, voir Figure 5.15 page suivante. Dans notre implémentation, il s'agit d'un « combat » entre deux individus. La meilleure valeur de fitness désigne le gagnant de chaque tournoi. Toutefois, chaque candidat peut participer à nouveau à d'autres « affrontements », toujours selon un tirage aléatoire. De ce fait, un même individu peut se retrouver dupliqué dans la liste finale. La sélection est effectuée jusqu'à obtenir une population de taille Pop_{min}.

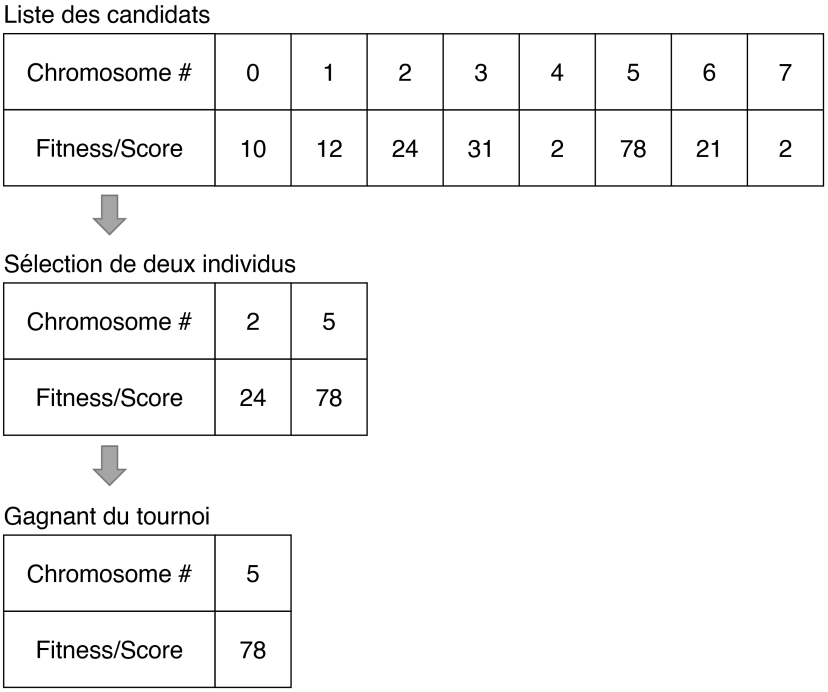


FIGURE 5.15 – Exemple de représentation d'un « tournoi » entre individus.

Croisement Les parents sélectionnés précédemment, sont regroupés par paires séquentiellement, voir Algorithme 1 page 119, ligne 5 à 11. Avec la probabilité p_{cross}, voir Algorithme 1, ligne 12 à 16, le premier parent peut être muté directement sans être combiné à un autre individu. Dans le cas d'un croisement entre deux parents, nous utilisons un opérateur *one-point crossover* personnalisé qui choisit, avec une probabilité égale, de séparer la grille horizontalement ou verticalement, comme proposé par Ming-Wen Tsai [188]. D'autre part, la position de fractionnement est choisie aléatoirement. La Figure 5.16 page suivante illustre un exemple de croisement horizontal et vertical pour nos parents.

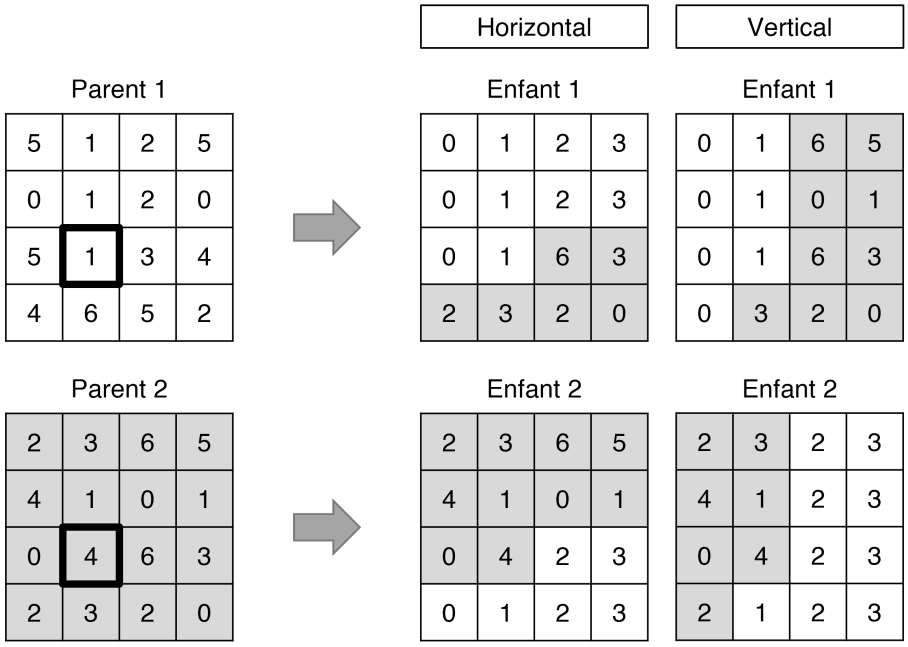


FIGURE 5.16 – Exemple de représentation de notre croisement *one-point crossover*, avec une découpe horizontale ou verticale selon un point de fractionnement, désigné par la case au contour accentué.

Mutation Nous appliquons ensuite un opérateur de mutation uniforme au résultat du croisement, voir Algorithme 1 page 119, ligne 17. Chaque gène du chromosome va ainsi pouvoir « muter » aléatoirement avec une probabilité p_{mut}, voir Figure 5.17. La mutation revient à sélectionner l'une des valeurs possibles d'un gène, voir Figure 5.12 page 120, et ce de façon aléatoire.

FIGURE 5.17 – Exemple de notre mutation uniforme sur une courte séquence d'un chromosome. À droite, les cases en rouges représentent celles qui ont « muté ».

Réinsertion Pour finir, notre réinsertion est élitiste. En effet, lors des étapes précédentes, nous avons généré Pop_{min} individus. Nous ajoutons, à présent, les $(Pop_{max} - Pop_{min})$ meilleurs parents de la dernière itération P_n à la nouvelle population P_{n+1}, voir Algorithme 1 page 119, ligne 24.

5.5 Évaluation par simulation

Afin de guider l'algorithme génétique, nous calculons un score pour chaque niveau généré, en fonction de l'expérience de jeu que nous essayons d'offrir. Nous allons donc attribuer une « note » ou un indice de qualité, à savoir une valeur de fitness, à chacun de nos individus.

Pour ce faire, nous utilisons la simulation d'un agent autonome représentant un joueur et adoptant un comportement simplifié. Cet agent simule son parcours dans chaque résultat généré par le WFC, et note un « ressenti » selon des heuristiques de perception, à savoir la nouveauté, la sécurité et la complexité. Cette évaluation par simulation guide la méthode itérative vers une certaine expérience de jeu selon les préférences initiales de l'agent joueur.

Dans notre simulation, le joueur synthétique effectue un nombre de pas définit dans le niveau. Chaque pas correspond à un déplacement sur une cellule de la grille. Afin de calculer la valeur de notation pour chaque avancée de l'agent, c'est-à-dire l'évaluation de son expérience de jeu, nous utilisons une pondération de la nouveauté, de la sécurité et de la complexité. Ces pondérations, représentant les préférences de l'agent, correspondent à nos paramètres d'influence sur l'expérience de jeu recherchée par la méthode itérative. La traversée du niveau donne ainsi lieu à un indice de qualité qui correspond alors à la moyenne des valeurs obtenues pour tous les pas effectués.

Nous pouvons décrire le calcul de la valeur de notation par « pas de l'agent », comme suit :

$$F_{Pas} = x * N + y * S + z * C$$

Où N, S, C sont respectivement les valeurs de nouveauté, sécurité, complexité, et où $x, y, z \in [-1; 1]$ sont les coefficients de pondération.

Le score global, à savoir la valeur de fitness, d'un niveau correspond alors à l'équation suivante :

$$F_{Niveau} = \sum_{i=1}^{N} \frac{F_{Pas_i}}{N}$$

Où N est le nombre de pas effectués par l'agent.

5.5.1 Navigation de l'agent

La navigation de l'agent autonome est uniquement basée sur la notion de nouveauté et l'accessibilité des cellules adjacentes. Dans ce but, nous précalculons également un mesh de navigation 2D, sur

la grille, par rapport aux modules placés dans le niveau, voir Figure 5.18. Ainsi, lors du prochain mouvement de case à effectuer, l'agent choisit la cellule adjacente suivante qui a été visitée le plus longtemps auparavant.

Se focaliser uniquement sur la nouveauté pour stimuler l'avancée de l'agent, représente pour nous un moyen d'explorer le niveau dans son ensemble, sans être influencé par les autres heuristiques. D'un autre côté, cela nous permettrait également de vérifier la navigabilité du niveau.

Il est intéressant de noter que notre agent :

- ne marche qu'en ligne droite selon les axes 2D, x et z de la grille, donc dans 4 directions possibles,
- ne peut pas sauter,
- n'évalue les directions que dans un ordre fixe, de sorte que les égalités de score de nouveauté mènent toujours au même chemin ; ce comportement déterministe permet d'obtenir toujours un score d'évaluation identique pour un même niveau et pour les mêmes pondérations.

Il nous paraît pertinent de spécifier que x et z correspondent également à nos axes 2D de placement des modules sur la grille, et que y correspond donc à notre axe ascendant dans l'espace 3D.

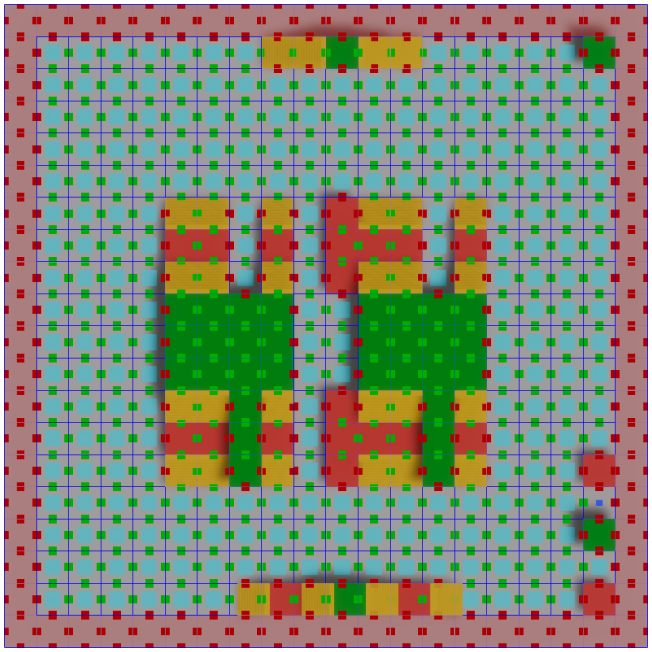


FIGURE 5.18 – Représentation de notre mesh de navigation 2D, calculé d'après la grille présentée Figure 5.7 page 113. Chaque cellule possède quatre connexions possibles, en vert celles navigables et en rouge celles inaccessibles.

5.5.2 Heuristiques

Nous présentons, ci-dessous, plus en détail, nos trois heuristiques permettant l'évaluation d'une expérience de jeu. Nous avons décidé d'adapter à notre méthode, les notions de nouveauté, de sécurité et de complexité, d'une manière qui nous paraissait la plus appropriée et la plus pertinente.

Premièrement, le facteur de nouveauté est pour nous lié à la dernière visite d'une cellule par l'agent. Ainsi, si une cellule a été visitée il y a un certain temps, s'y rendre fournira plus d'informations à l'agent. De ce fait, sa « curiosité » sera plus importante que dans le cas d'une case visitée plus récemment.

Deuxièmement, le sentiment de sécurité, est pour nous associé, dans un jeu de tir, à la géométrie du niveau, permettant de se cacher et d'éviter les tirs ennemis. D'un autre côté, nous ne simulons pas explicitement les agents ennemis, mais utilisons ce score de sécurité pour évaluer la couverture offerte par la géométrie contre les tirs ennemis.

Dernièrement, la perception de complexité, nous permet d'évaluer la simplicité géométrique du niveau d'un point de vue donné. Nous exprimons, ainsi, le fait qu'il y ait quelque chose « d'intéressant à voir ». Cette valeur nous permet donc d'éviter d'obtenir des niveaux totalement vides, homogènes ou composés de simples couloirs, par exemple.

Les équations détaillées ci-après sont des propositions d'implémentations, que nous avons utilisées, concernant ces heuristiques. Les valeurs de perception, de sécurité et de complexité sont mesurées avec une vision basée sur les deux axes de la grille, x et z, soit selon la moyenne des 4 directions, voir la représentation Figure 5.19 page suivante.

D'autre part, pour économiser du temps lors de la simulation, nous précalculons pour chaque cellule de la grille la valeur de sécurité liée à la visibilité, et la valeur de complexité.

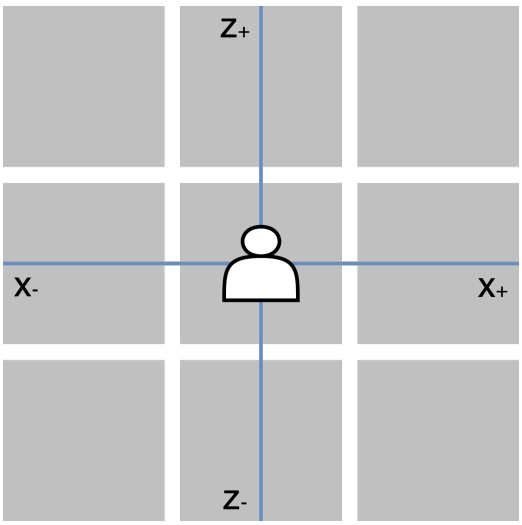


FIGURE 5.19 – Illustration de la vision de l'agent, par cellule, soit 4 directions prisent en compte.

Nouveauté La valeur de nouveauté de chaque cellule est calculée selon l'équation suivante :

$$N = 1 - \exp\left(-\frac{t_n - t_l}{200}\right)$$

Où t_n est le nombre actuel de pas, déjà effectués, lorsque l'agent évalue la cellule et où t_l est le nombre de pas enregistré lors de la dernière visite de cette case par l'agent. La valeur initiale de t_l est $-\infty$.

Avec l'exponentielle utilisée, l'agent n'oublie jamais une case visitée, elle ne sera jamais totalement nouvelle. Nous avons choisi arbitrairement la valeur de 200 pas pour la division.

Sécurité Pour évaluer l'expérience de jeu, nous utilisons également un sentiment de sécurité :

$$S = \left(\frac{V_{x+} + V_{x-} + V_{z+} + V_{z-}}{W * 4} + 1\right)^{-1}$$

Où les variables V_{x+}, V_{x-}, V_{z+} et V_{z-} représentent la racine carrée du nombre de cellules que l'agent perçoit dans chaque direction, et où W représente la taille de notre niveau en nombre de cellules.

De ce fait, plus l'agent dispose d'une visibilité sur le niveau, moins il se sentira en sécurité. En effet, pour nous, la sécurité est l'inverse de la visibilité. Pour calculer cette visibilité, nous utilisons un rayon qui émane de la position même de l'agent et traverse autant de cellules visibles que possible, et ce, dans chaque direction.

D'un autre côté, nous utilisons la racine carrée pour éviter une notation de la vision de façon linéaire. En effet, nous pensons que, d'un point de vue de la perception, disposer déjà d'un peu de visibilité est plus significatif qu'avoir une vision très large.

Complexité Nous calculons le facteur de complexité grâce à la formule suivante :

$$C = \frac{C_{x+} + C_{x-} + C_{z+} + C_{z-}}{W * 4}$$

Où C_{x+}, C_{x-}, C_{z+} et C_{z-} représentent la racine carrée de la complexité, calculée dans chaque direction à partir de la cellule actuelle, et où W est la taille de notre niveau en nombre de cellules.

Contrairement au calcul de la visibilité, nous utilisons cette fois-ci trois rayons, par direction, voir Figure 5.20. Le premier suit le même chemin que celui de la visibilité. Les deux autres rayons sont, quant à eux, parallèles au premier rayon d'une case, tout en gardant la même longueur. Par la suite, nous suivons tous ces rayons cellule par cellule, et nous calculons la complexité en comparant l'identifiant du module de la cellule courante et celui du module de la cellule précédente. Si les ID des modules divergent, cela signifie que l'agent voit quelque chose de « différent », nous ajoutons ainsi 1 de score à la complexité perçue dans cette direction. La somme est ensuite divisée par trois, car nous utilisons trois rayons.

Nous pouvons décrire la mesure de complexité dans une des directions, par la formule suivante :

$$C_{x+} = \sqrt{\frac{R_1 + R_2 + R_3}{3}}$$

Où R_1, R_2, R_3 correspondent chacun au nombre de cases perçues comme différentes, dans un rayon.

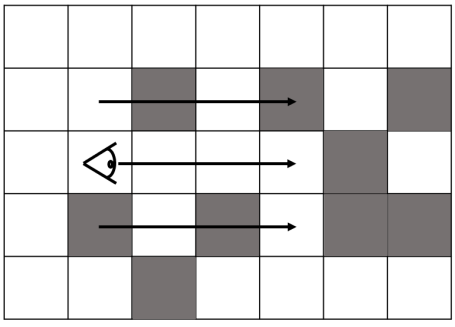


FIGURE 5.20 – Illustration des trois rayons de la complexité. La flèche au centre représente le trajet du rayon de la visibilité. Les deux autres rayons sont positionnés parallèlement et ont la même longueur.

5.6 Conlusion

Nous avons présenté et détaillé dans ce chapitre, notre méthode appelée *Genetic-WFC*. Il s'agit d'un pipeline de génération procédurale qui combine une approche *Search-Based*, à savoir un algorithme génétique et une évaluation par simulation, avec une méthode constructive, le *Wave Function Collapse*, afin de générer des niveaux ciblant des expériences de jeu spécifiques.

Le WFC agit, dans notre proposition, comme opérateur de réparation dans la méthode itérative, et permet de manipuler uniquement les niveaux respectant des contraintes d'adjacence sur le placement des assets. Ainsi, l'algorithme génétique se concentre plus particulièrement sur l'évaluation des niveaux grâce à une simulation de l'expérience de jeu. Dans le but de piloter le WFC, nous biaisons localement les probabilités de sélection des modules et leurs rotations grâce à des zones de probabilités, manipulées directement par l'algorithme génétique.

Dans le cadre de notre algorithme d'optimisation, nous utilisons un opérateur de croisement 2D spécifique pour diviser le niveau verticalement ou horizontalement. Nous réencodons également les gènes en utilisant les modules et rotations effectivement choisis par le WFC dans le niveau généré, afin de maximiser l'efficacité du croisement. Nous employons de même une sélection par tournoi, une réinsertion élitiste et une mutation uniforme.

L'évaluation est, quant à elle, effectuée par la simulation du parcours d'un agent autonome dans le niveau généré par le WFC. Ce joueur synthétique note son exploration du niveau selon des heuristiques de perception, à savoir la nouveauté, la sécurité et la complexité. Afin de guider la méthode itérative vers une certaine expérience de jeu, nous appliquons des coefficients de pondération aux heuristiques de l'agent.

Nous proposons également un principe de multicouche, voir Figure 5.21 page suivante, avec l'utilisation d'une première étape employant uniquement des *greyblocks* dans la méthode itérative, pour limiter l'explosion combinatoire du WFC. Puis, nous remplaçons simplement le résultat final obtenu par l'algorithme génétique par des assets plus visuels.

Dans le prochain chapitre, nous aborderons l'évaluation et l'analyse de notre méthode de génération au travers de plusieurs expérimentations et la présentation des résultats obtenus.

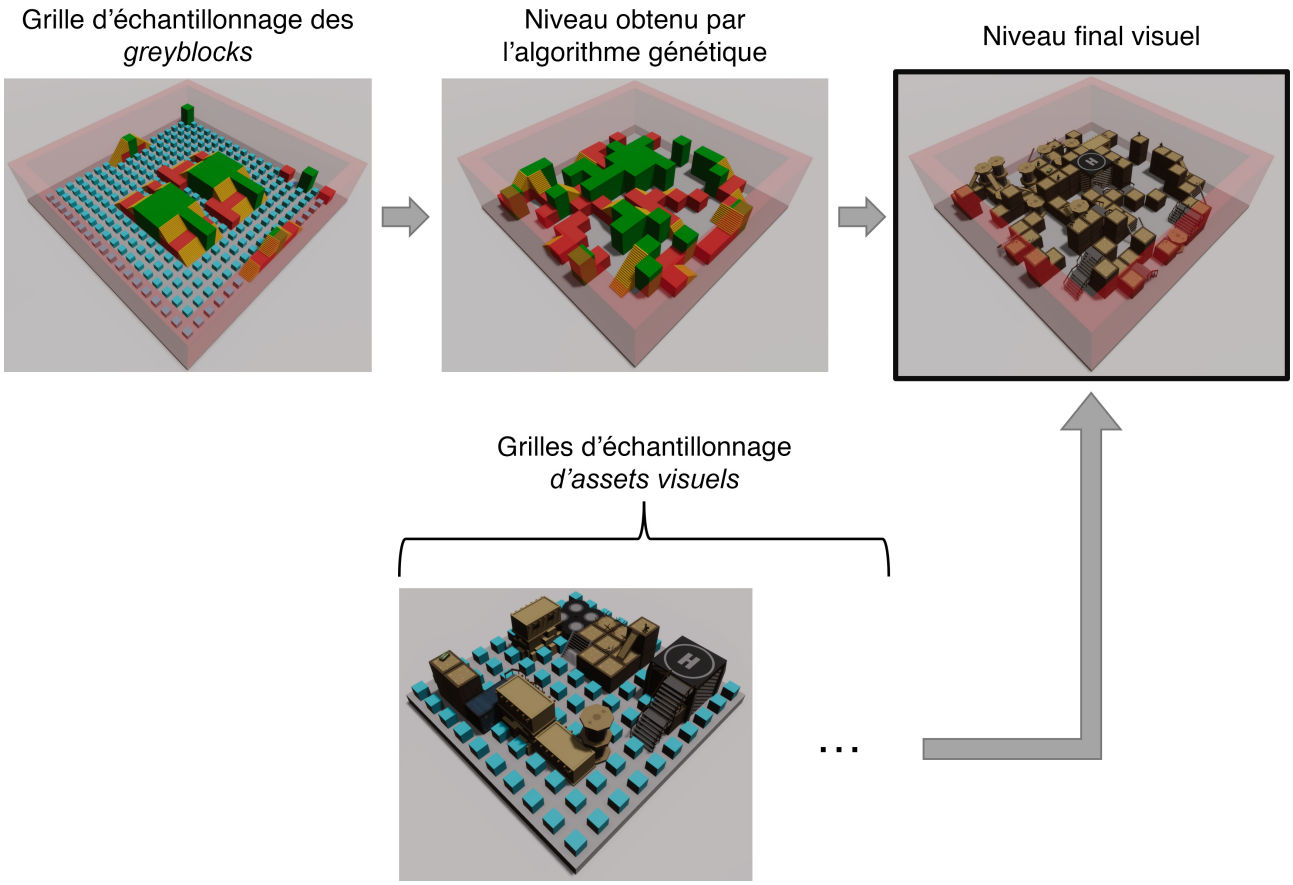


FIGURE 5.21 – Résumé illustré de notre pipeline de génération, appelé *Genetic-WFC*.

Chapitre 6

Évaluation de la méthode, expérimentations et résultats

Contenu

6.1 Introduction

Dans le chapitre précédent, nous avons détaillé le fonctionnement de notre méthode appelée *Genetic-WFC*. Nous nous sommes penchés sur ses différentes fonctionnalités. Nous allons à présent chercher à valider l'intérêt et l'utilité d'une telle proposition d'approche et analyser ses capacités grâce à plusieurs expériences. Les différentes expérimentations qui seront décrites, ci-après, vont nous permettre de confirmer nos hypothèses de développement et de répondre à nos questionnements face à notre problématique.

Pour cela, nous allons exposer trois expériences en liaison avec nos algorithmes utilisés. Dans la première expérimentation, nous chercherons à établir le temps de calcul nécessaire au WFC. Puis, nous expliquerons le choix de composition de notre niveau, avec la présentation des différents modules et d'une grille d'échantillonnage spécifique. Ensuite, à l'aide d'une comparaison entre diverses méthodes similaires, nous chercherons à démontrer l'utilité de notre *Genetic-WFC*. Une troisième et dernière expérience nous permettra d'explorer l'espace de génération de niveau et d'analyser la diversité des expériences de jeu que peut proposer notre méthode.

Les résultats qui seront présentés dans ce chapitre reprendront donc notre implémentation du *Genetic-WFC*, introduite précédemment. Notre méthode, comme l'avons déjà souligné, est écrite en C# et s'exécute dans le moteur de jeu *Unity*. Les diverses expérimentations et évaluations ont été réalisées avec un processeur *Intel Core i7-9700K*.

6.2 Wave Function Collapse

Dans cette partie, nous nous concentrons sur la performance de notre implémentation de l'algorithme du *Wave Function Collapse*.

6.2.1 Expérience n°1 - Temps de calcul

6.2.1.1 Objectif de l'expérience

L'objectif de notre première expérimentation est de visualiser l'influence du *Wave Function Collapse* sur le temps de calcul dans la génération de niveaux. Nous avons également l'intention de trouver le ratio temps/module/taille acceptable pour une recherche évolutionnaire. De plus, nous espérons estimer l'impact des modules plus contraints, tels que les *BigTiles* et leurs sous-modules.

6.2.1.2 Méthode et protocole d'expérimentation

Afin d'évaluer l'impact du WFC, nous avons réalisé une évaluation comparative de la croissance du temps de calcul en fonction de la taille de la grille et du nombre de modules disponibles.

Nous avons effectué 1000 WFC consécutifs pour chaque résultat, en *single thread* pour le processeur, dans le but d'obtenir une moyenne de temps en milliseconde (ms). Nous avons testé diverses tailles de la grille allant de 10 à 30 cellules, et fait varier le nombre de modules avec des valeurs entre 3 à 18.

Pour le passage de 12 à 18 modules, nous avons donc ajouté 6 assets, sur lesquels nous avons appliqué deux variations, nommées 18a et 18b. Dans la première, les modules rajoutés ne sont pas contraints de se placer l'un à côté de l'autre. Cependant, dans la seconde version, ces modules doivent obligatoirement être voisins, placés côte à côte tel un rectangle. Cela nous a permis d'émuler la représentation d'un *BigTile* avec ses sous-modules.

Nous avons utilisé uniquement des assets ayant la taille d'une seule case, en incluant le module d'« air ». Toutefois, nous ne prenons pas en compte, ni ne calculons, le module de « bordure ».

6.2.1.3 Résultats et discussion

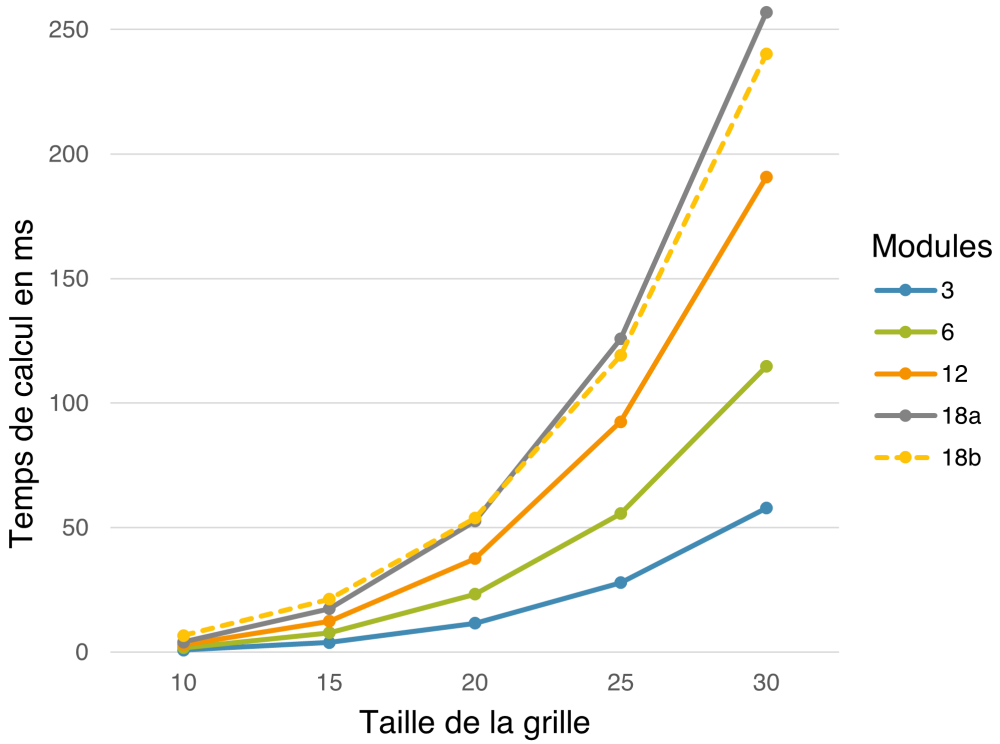


FIGURE 6.1 – Impact du nombre de modules et de la taille de la grille sur le temps de génération du WFC, temps de calcul moyen (en ms) pour 1000 exécutions. La variation 18a possède 6 modules non contraints entre eux, contrairement à la 18b qui simule un *BigTile* rectangulaire.

Taille Grille	Nombre de modules				
	3	6	12	18a	18b
10×10	1	2	3	4	7
15×15	4	8	12	17	21
20×20	12	23	38	53	54
25×25	28	56	93	126	119
30×30	58	115	191	257	240

TABLE 6.1 – Temps de calcul moyen du WFC (en ms), pour 1000 exécutions, en fonction de la taille de la grille et du nombre de modules disponibles. La variation 18a possède 6 modules non contraints entre eux, contrairement à la 18b qui simule un *BigTile* rectangulaire. Les résultats correspondent à la Figure 6.1.

Le Tableau 6.1 page précédente illustre les résultats obtenus concernant les temps de calcul pour le WFC. La Figure 6.1 page précédente reprend ces mêmes données.

Nous pouvons constater que le temps de génération croît rapidement en fonction du nombre d'assets et de la taille de la grille. Le temps nécessaire peut aller jusqu'à plus de 257 ms pour une grille de 30×30 cases, avec 18 modules différents.

Nous pouvons également noter que les résultats pour 18a et 18b ne divergent qu'à partir d'une certaine taille de grille. Ces chiffres suggèrent que contraindre des modules est avantageux sur de plus grandes tailles de niveaux, car davantage de modules seront supprimés rapidement. Le temps de calcul passe de 257 ms à 240 ms, soit une diminution de 6,6%. En effet, plus nous contraignons nos modules, plus le nombre de possibilités semble décroître grâce à la propagation, et par conséquent, la grille est générée plus rapidement. Nous pouvons donc en déduire que le nombre de relations possibles entre modules a un impact sur le temps de calcul final.

6.2.1.4 Synthèse des observations

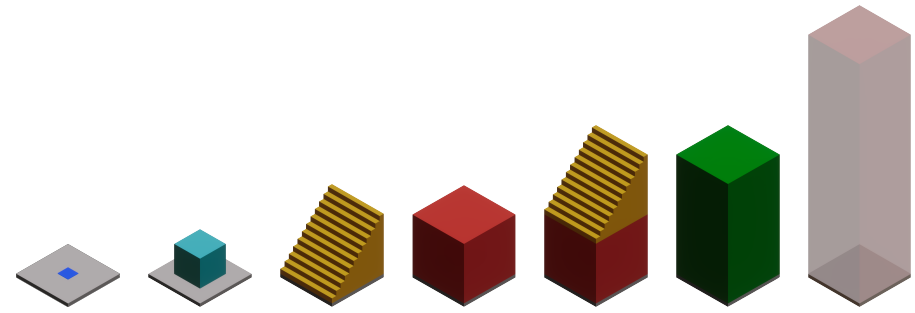
Cette expérience a permis de mettre en avant l'impact du nombre de modules, des contraintes et de la taille du résultat sur le temps de calcul du WFC.

Au terme de cette analyse, nous pouvons conclure que limiter le nombre d'assets, avec l'utilisation potentielle de *greyblocks*, représenterait un gain de temps significatif. Grâce aux résultats de cette expérimentation, nous pourrons ainsi définir un nombre d'assets adéquat et une taille de la grille associée qui permettra de contenir l'impact du WFC sur le temps de calcul dans le cadre de la méthode itérative.

6.3 Genetic-WFC

Dans cette partie, nous allons décrire les expériences réalisées afin d'évaluer les capacités de notre méthode à proprement parler. Nous allons nous intéresser à son efficacité par rapport à d'autres approches similaires, ainsi qu'à sa performance concernant la diversité de résultats atteignable.

6.3.1 Choix de la composition de notre niveau



(a) Point de départ (b) Air (c) Escalier H1 (d) Cube H1 (e) Escalier H2 (f) Cube H2 (g) Bordure

FIGURE 6.2 – Modules de *greyblock* employés.

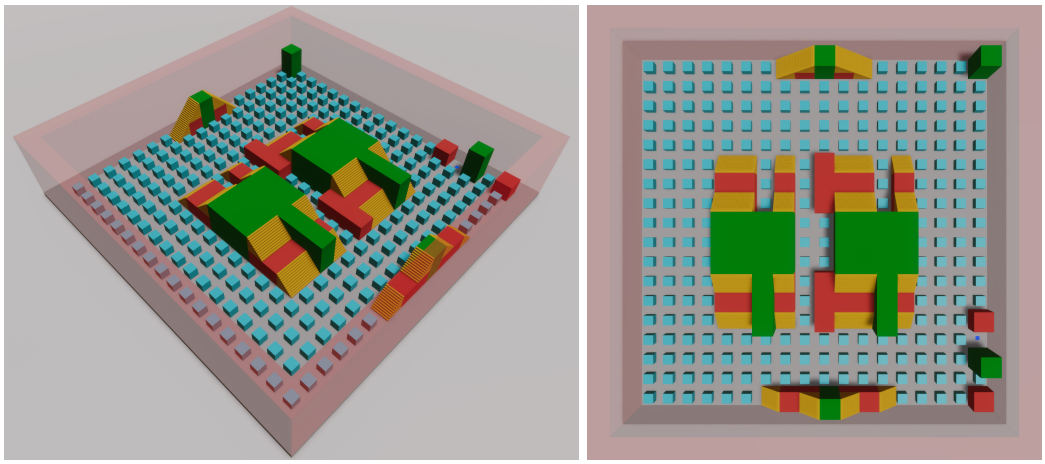


FIGURE 6.3 – Grille d'échantillonnage pour le WFC, avec les modules Figure 6.2. Le point de départ est réglé pour n'être présent au minimum et au maximum qu'une fois dans le résultat.

Pour les prochaines expériences, nous avons choisi, pour la génération, de viser des niveaux de 15 sur 15 cases et d'employer les modules de *greyblock* présentés dans la Figure 6.2. Pour le WFC, nous avons utilisé la grille d'exemple Figure 6.3. En effet, ces assets nous paraissent adaptés à la création de

niveaux ouverts non-linéaires en 3D, avec de la verticalité. En effet, ces modules entravent la visibilité afin de simuler la géométrie typique du genre FPS pour, par exemple, se mettre à couvert. Nous pensons également que ces 7 *greyblocks* combinés sont suffisants pour créer une abstraction jouable de ces niveaux et nous permettre d'évaluer de manière pertinente notre méthode de génération. De plus, nous pensons que ce nombre défini de modules conviendra à la création d'environnements variés avec différents parcours.

Il serait envisageable, par la suite, d'aboutir à une version visuelle avec des objets 3D de campement grâce à l'emploi du multicouche de notre WFC. En effet, nous avons préféré privilégier, en premier lieu, la validation de la structure navigable d'un fortin avec ces quelques *greyblocks*, avant le rajout d'éléments de gameplay et de l'aspect visuel.

Nos 7 *greyblocks*, Figure 6.2 page précédente, représentent chacun un module de la taille d'une seule case sur la grille. Ils sont également associés chacun à une catégorie précise d'asset, à savoir : point de départ du joueur (a), volume d'espace vide (b), volume praticable bloquant bas (d), volume praticable bloquant haut (f), transition du sol au volume praticable bloquant bas (c), transition du volume praticable bloquant bas au volume praticable bloquant haut (d), volume délimitant l'espace de jeu (g). Ce dernier volume (g), c'est-à-dire l'asset « bordure », a également comme fonction de contrôler l'orientation des escaliers aux abords du niveau, voir Figure 6.3 page précédente.

Le choix d'utiliser ces 7 assets précis est basé, entre autres, sur les résultats de l'expérience 1, voir Sous-Section 6.2.1 page 135. En effet, en appliquant le même protocole de calcul sur une grille de 15x15 avec la grille d'échantillonnage choisie, nous arrivons à 4 ms de temps de calcul. Ce critère nous a paru satisfaisant pour nous permettre d'effectuer une recherche évolutionnaire.

6.3.2 Expérience n°2 - Évaluation de la méthode

6.3.2.1 Objectif de l'expérience

Le but de cette expérience est de comparer notre méthode *Genetic-WFC* à d'autres approches similaires afin d'évaluer son potentiel. L'objectif est également de valider notre implémentation et nos différentes hypothèses vis-à-vis du pilotage et de la combinaison du WFC avec un algorithme génétique.

6.3.2.2 Méthode et protocole d'expérimentation

Afin de réaliser cette expérimentation, nous avons fait le choix d'exécuter plusieurs méthodes similaires à notre approche pour la génération des niveaux. En conséquence, nous avons exécuté 10 fois chaque méthode dans le but d'obtenir des valeurs moyennes. Tous les résultats des approches suivantes ont été obtenus en *multithread* pour le processeur.

Nous avons utilisé, pour chacune de ces méthodes, la simulation de notre joueur synthétique comme évaluation des niveaux produits par la génération, avec les heuristiques de nouveauté, de sécurité et de complexité, sur 1125 pas. Le calcul de la fitness par « pas de l'agent » dans la simulation a été fixée à $F_{Pas} = 1 * N + 1 * S - 1 * C$. Nous avons donc cherché à maximiser la nouveauté et la sécurité, et à minimiser la complexité. De plus, un résultat peut être évalué uniquement s'il est considéré comme valide, c'est-à-dire jouable, dès lors qu'un point de départ est disponible sur le terrain. Dans le cas contraire, la valeur de l'évaluation du niveau est fixée à $-\infty$.

Le fait que le parcours soit évalué sur une base de 1125 pas est une valeur expérimentale que nous avons choisie arbitrairement et qui nous semblait suffisamment représentative d'un parcours dans un niveau. Cette valeur correspond à 5 fois le nombre de cellules dans la grille de résultat. Notre calcul peut être détaillé comme suit : la taille de 15x15 donne 225 cellules, donc 1125 pas. Nous sommes conscients que cette durée de parcours pourrait être soit réduite, soit augmentée. L'impact de cette valeur sur l'évaluation du niveau mériterait d'être davantage approfondi à l'aide d'expériences ultérieures.

Dans cette expérience, nous avons ainsi mis en place les approches suivantes :

Genetic-WFC Notre méthode consiste à combiner un algorithme génétique et le WFC comme opérateur de réparation.

Genetic-WFC Non Réencodé Notre méthode *Genetic-WFC* est cette fois exécutée sans l'étape de réencodage des résultats dans les gènes. Cette variation permet d'illustrer le bénéfice du réencodage des gènes sur le processus d'optimisation.

Algorithme génétique sans WFC Il s'agit ici d'une implémentation d'un algorithme génétique basique, c'est-à-dire que nous désactivons le WFC comme opérateur de réparation, et que chaque gène du chromosome est directement traduit par un module et sa rotation dans la grille.

Algorithme génétique sans WFC pénalisé Dans le cadre de cette méthode, des pénalisations sont rajoutées à l'*algorithme génétique sans WFC*, mentionné ci-dessus. L'inclusion de malus sur le placement des assets incite la méthode itérative à corriger ses erreurs. Ce procédé de correction est le plus similaire à notre *Genetic-WFC*. Cependant, rectifier les erreurs par l'optimisation n'est pas la même démarche que d'avoir un algorithme qui corrige ces fautes pour nous.

Dans le cas de cette approche pénalisée, l'algorithme génétique travaille avec une valeur pénalisée, $F_{\text{Pénalisée}}$, tout en conservant la valeur de la simulation du niveau F_{Niveau}, permettant ainsi la comparaison avec les autres méthodes similaires.

Nous avons décidé d'employer des malus simplifiés qui ne reprennent pas toutes les règles d'adjacences du WFC. De ce fait, nos pénalisations sont les suivantes :

- Pour les points de départ : il y a pénalité quand le niveau ne possède aucun ou plus d'un point de départ ;
- Pour les escaliers : il y a pénalité quand le début ou la fin d'un escalier ne mène pas à une surface navigable ;
- Pour les bordures : il y a pénalité quand une bordure est placée à l'intérieur du niveau.

Dans le cas spécifique de cette méthode, seuls les niveaux disposant d'un point de départ unique sont simulés par la suite, les autres niveaux sont évincés de l'évaluation par l'agent. La valeur de fitness de la simulation est alors fixée à $F_{Niveau} = -\infty$. L'algorithme génétique se voit infligé, quant à lui, une valeur pénalisée hautement négative, proportionnelle à la pénalité des points de départ.

Dans le cas où le niveau a bien été simulé, la formule de pénalisation, concernant les bordures et les escaliers, peut être exprimée de la façon suivante :

$$F_{\text{Pénalisée}} = F_{Niveau} - (M_{Bordure} + M_{Escalier}) * 0,01$$

Où F_{Niveau} est le résultat de l'évaluation du niveau par l'agent, $M_{Bordure}$ et $M_{Escalier}$ correspondent au nombre de bordures et d'escaliers mal positionnés.

WFC uniquement Pour comprendre l'utilité réelle de l'optimisation génétique et le gain de fitness qu'elle procure, nous avons généré des niveaux en utilisant uniquement le WFC. Cette méthode est donc un algorithme de « recherche » par force brute, sans opérateurs génétiques, qui utilise uniquement le WFC pour créer les niveaux, tout en gardant le meilleur résultat par rapport à la simulation de l'agent.

Le WFC, en lui-même, est un algorithme de génération de niveaux procédurale qui peut donner des résultats très intéressants, même lorsqu'il n'est pas piloté par un processus d'optimisation. En effet, la génération de notre implémentation du WFC utilise les fréquences d'apparition de modules dans les grilles d'échantillonnage.

Les paramètres généraux concernant ces cinq approches, sont repris dans le Tableau 6.2 page suivante. Cependant, chaque méthode possède des spécificités propres, que nous nous proposons de détailler ci-après.

Les paramètres de génération qui ont servi pour les méthodes employant des algorithmes génétiques, c'est-à-dire le *Genetic-WFC*, le *Genetic-WFC N.R.*, l'*algorithme génétique sans WFC* et l'*algorithme génétique sans WFC pénalisé*, sont : 10k itérations, avec une taille de population définie à $Pop_{min} = 50$ et $Pop_{max} = 60$. La probabilité de mutation a été fixée à $p_{mut} = 0,04$ et la probabilité de croisement à $p_{cross} = 0,75$.

D'un autre côté, concernant la génération du niveau à proprement parler, la taille de la grille a

été fixée à 15 sur 15 avec les 7 modules disponibles et les bordures forcées, que ce soit avec ou sans l'inclusion du WFC.

Comme la taille de la population minimale des algorithmes génétiques, présentés ci-dessus, est définie à 50, et que nous exécutons 10k itérations, nous avons choisi de calculer ici $10k * 50$ soit $500k$ WFC aléatoires pour la méthode *WFC uniquement* et de ne garder que le meilleur résultat. Cependant, cette approche a été incluse strictement à titre indicatif, car elle n'incorpore pas de processus d'optimisation, et n'est basée donc que sur la génération aléatoire des WFC. Elle est toutefois comparable aux autres méthodes en termes de valeur de fitness obtenue.

Paramètres de l'algorithme génétique	
Population (min/max)	50/60
Itérations	10000
Croisement	75%
Mutation (par gène)	4%
Paramètres du WFC	
Taille	15x15
Modules	7
Bordures	Oui
Paramètres de la simulation	
Nouveauté	1.0
Sécurité	1.0
Complexité	-1.0
Nombre de pas	1125

TABLE 6.2 – Paramètres propres à chaque partie des méthodes utilisées.

Dans les sections suivantes, nous allons donc comparer notre *Genetic-WFC* aux approches qu'elle devrait améliorer, à l'aide des différents résultats obtenus. Nous allons tout d'abord analyser sa performance au travers de graphiques de fitness et de temps de calcul. Puis, nous ajusterons l'*algorithme génétique sans WFC pénalisé* sur la durée, en reprenant les mêmes paramètres présentés ci-dessus. Pour cela, nous modifierons uniquement le nombre d'itérations afin d'obtenir un temps de calcul similaire au *Genetic-WFC*. Nous terminerons par l'illustration et la comparaison des niveaux obtenus avec chaque approche, et par la description plus en détail de notre meilleur résultat concernant notre *Genetic-WFC*.

6.3.2.3 Fitness entre méthodes - Résultats et discussion

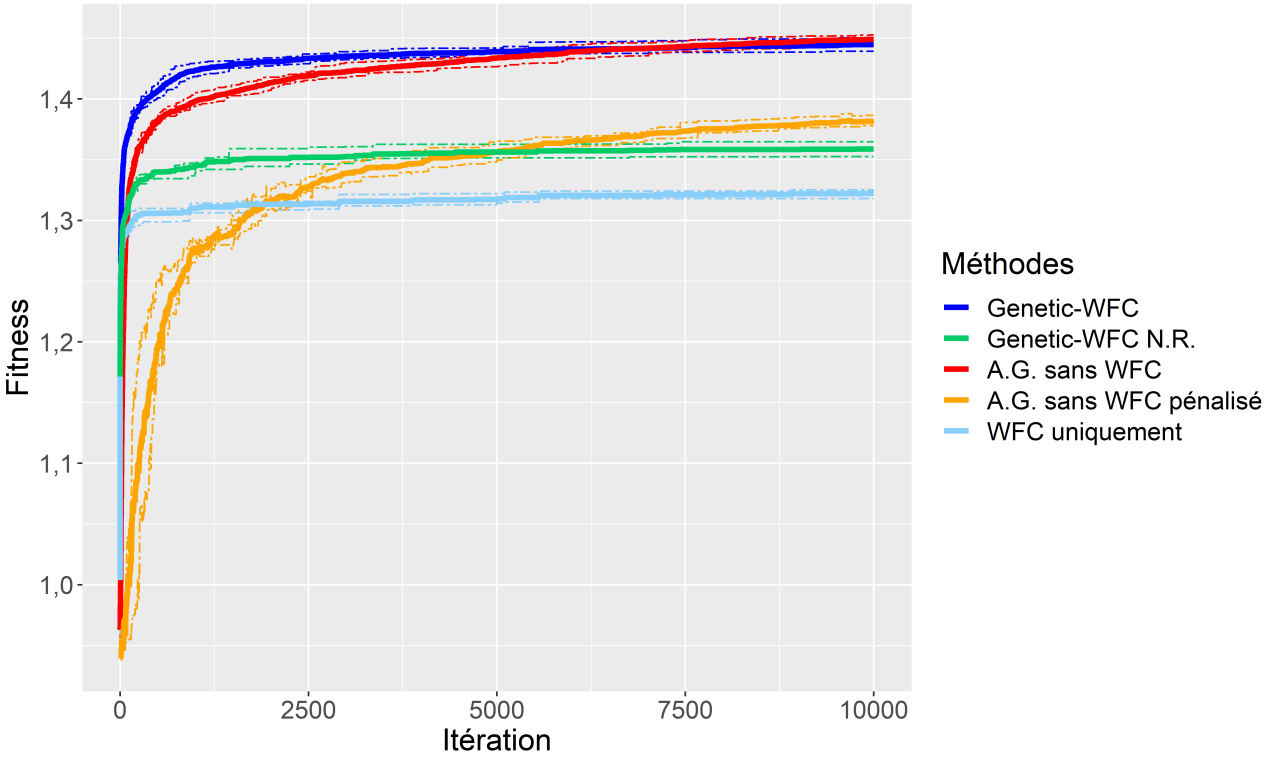


FIGURE 6.4 – Moyenne de l'évolution de la meilleure fitness pour 10 exécutions de chaque méthode, sur 10k itérations. Les lignes en pointillées indiquent les premiers et troisièmes quartiles. NR signifie Non Réencodé, et A.G., Algorithme Génétique. La fonction d'évaluation du parcours de l'agent correspond à $F_{Pas} = N + S - C$. La fitness de l'*algorithme génétique sans WFC pénalisé* est rapportée sans pénalité, ce qui permet de comparer les diverses approches.

Méthode	Total	Nouveauté	Sécurité	Complexité
Genetic-WFC	1,445 (0,009)	0,578 (0,009)	0,913 (0,002)	-0,046 (0,002)
Genetic-WFC N.R.	1,359 (0,008)	0,493 (0,009)	0,916 (0,002)	-0,051 (0,002)
A.G. sans WFC	1,449 (0,007)	0,571 (0,006)	0,922 (0,003)	-0,044 (0,002)
A.G. sans WFC pénalisé	1,382 (0,007)	0,514 (0,008)	0,915 (0,003)	-0,047 (0,004)
WFC uniquement	1,322 (0,005)	0,452 (0,009)	0,917 (0,003)	-0,047 (0,004)

TABLE 6.3 – Moyenne et (écart-type) des meilleurs scores atteints après 10 exécutions de chaque méthode, sur 10k itérations, arrondie à la troisième décimale pour la moyenne et pour l'écart-type. La fonction d'évaluation du parcours de l'agent correspond à $F_{Pas} = N + S - C$. La fitness de l'*algorithme génétique sans WFC pénalisé* est rapportée sans pénalité pour permettre la comparaison des diverses approches. Les résultats correspondent aux valeurs finales des courbes de la Figure 6.4. Une version plus détaillée des résultats de ce tableau peut être consultée en Annexe B page 223.

En premier lieu, nous pouvons examiner les courbes des moyennes de l'évolution de la meilleure fitness, correspondantes à chaque méthode et illustrées dans la Figure 6.4 page précédente.

La courbe de la méthode, impliquant uniquement le WFC, montre que, dans les toutes premières itérations, une valeur d'optimisation maximale est rapidement atteinte et que son évolution reste relativement stable. Les fréquences d'apparition initiales du WFC semblent restreindre la possibilité d'obtenir un niveau correspondant aux paramètres voulus de la simulation. Le résultat obtenu pour cette méthode, *WFC uniquement*, indique que l'implication d'un algorithme génétique aide à obtenir de meilleurs résultats. Nous constatons, en effet, que les quatre autres approches présentent des valeurs finales supérieures et dépassent même à partir de 1800 itérations environ la méthode *WFC uniquement*. Nous pouvons en déduire que nous avons donc réussi à piloter l'algorithme WFC avec un algorithme génétique pour obtenir de meilleurs résultats, voir la courbe *Genetic-WFC*. De plus, le réencodage des gènes aide clairement le processus d'optimisation, qui est plus faible lorsqu'il est désactivé, voir la courbe du *Genetic-WFC Non Réencodé*.

En observant les courbes du *Genetic-WFC* et de l'*algorithme génétique sans WFC*, nous pouvons voir que le *Genetic-WFC* atteint rapidement une meilleure valeur de fitness durant les premières itérations. En tant qu'opérateur de réparation, le WFC semble permettre à l'algorithme génétique d'obtenir directement de meilleurs niveaux, structurellement parlant. Le WFC corrige, en effet, les erreurs de placements liés aux zones de probabilité proposées par l'algorithme génétique. Par exemple, il oblige les escaliers à être placés contre les murs de manière navigable, alors que l'*algorithme génétique sans WFC* doit déduire le placement optimal pour chaque type d'asset.

La courbe de l'*algorithme génétique sans WFC pénalisé*, quant à elle, montre une progression plus constante sur la durée, mais pour le même nombre d'itérations, la valeur de fitness atteinte est plus faible que celle de notre méthode. Le résultat final devrait toutefois montrer une qualité structurelle plus élevée que l'algorithme génétique seul, sans malus.

Il est important de préciser que la valeur de fitness rapportée pour l'*algorithme génétique sans WFC pénalisé* ne correspond pas à la valeur utilisée directement par l'algorithme génétique. Il s'agit seulement de la partie correspondante à la simulation du joueur synthétique avant l'application des malus. Cela nous permet de comparer les différents résultats entre eux.

En dernière analyse, nous constatons que la plupart des méthodes stagnent plus ou moins rapide-

ment. Seule la courbe pénalisée continue sa montée. Notre méthode ne fait pas exception et semble atteindre une valeur quasi finale entre 5000 et 6000 itérations environ. En effet, si nous nous arrêtons à 5000 itérations, notre méthode atteint approximativement 98% de sa valeur de fitness finale. Cette estimation a été réalisée en normalisant les valeurs de fitness, entre la valeur minimale et maximale de la courbe.

À présent, nous allons nous pencher sur les données du Tableau 6.3 page 144, afin de mieux comprendre les performances de chaque méthode.

Nous pouvons constater que notre approche *Genetic-WFC* atteint la meilleure valeur de nouveauté en moyenne, mais obtient la plus faible en sécurité. En prenant en compte les différentes méthodes, la nouveauté présente le plus grand écart-type de nos trois heuristiques, face aux plus faibles variations de la complexité et de la sécurité.

D'autre part, l'approche sans restriction, c'est-à-dire l'*algorithme génétique sans WFC*, atteint une valeur maximale moyenne légèrement supérieure à notre *Genetic-WFC*. Cependant, comme nous le verrons par la suite dans la Sous-Section 6.3.2.6 page 151, les niveaux générés sans aucune pénalisation obtiennent certes une meilleure valeur de fitness, mais surexploitent certains modules et fournissent des niveaux malheureusement inexploitables.

6.3.2.4 Temps de calcul entre méthodes - Résultats et discussion

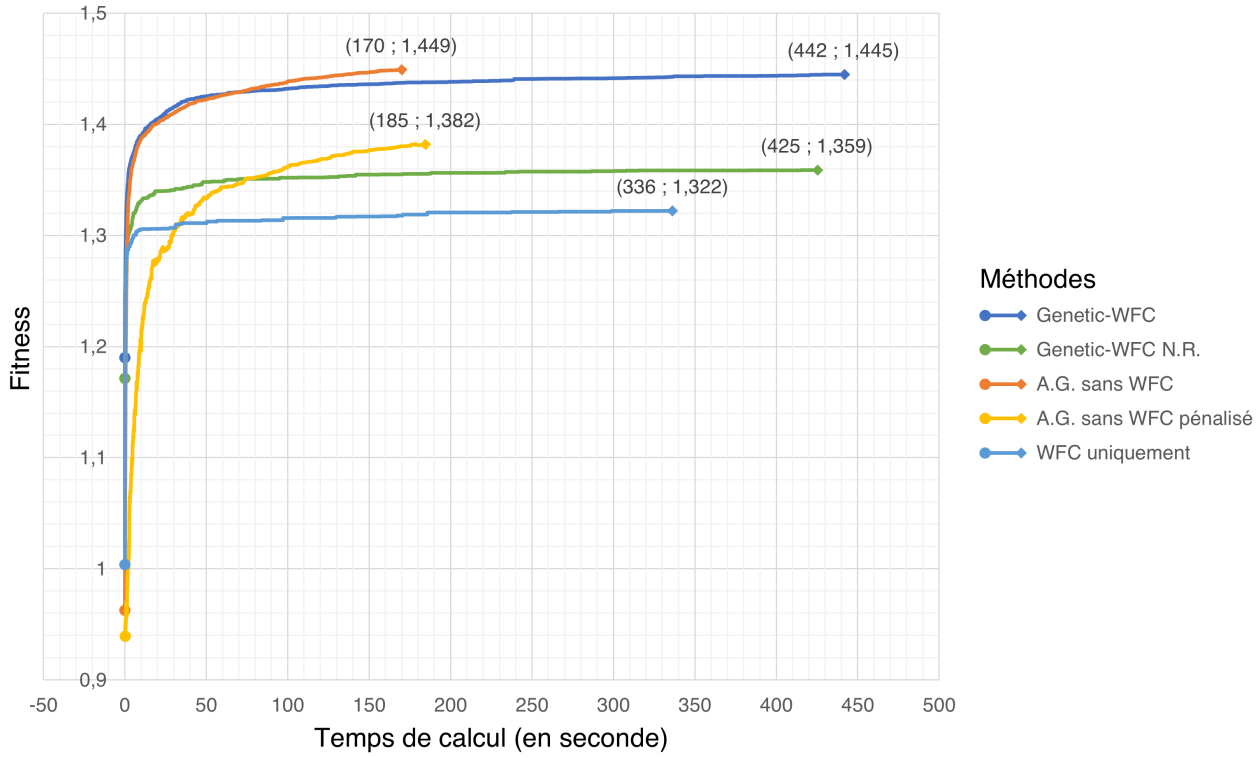


FIGURE 6.5 – Moyenne de l'évolution de la meilleure fitness pour 10 exécutions de chaque méthode, selon le temps de calcul en seconde. L'approche *WFC uniquement*, également en *multithread*, figure strictement à titre indicatif, car l'implémentation n'est pas comparable aux algorithmes génétiques. Les résultats correspondent à ceux mentionnés dans la Sous-Section 6.3.2.3 page 144.

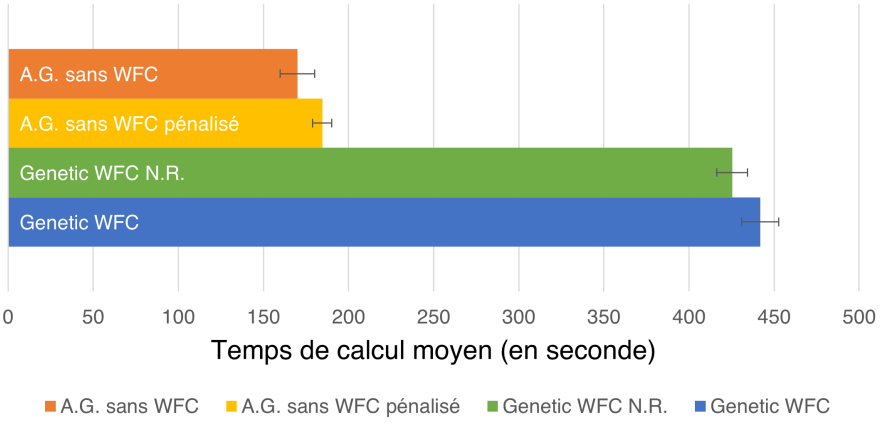


FIGURE 6.6 – Temps de calcul moyen, en seconde, des différents algorithmes génétiques, présents dans la Figure 6.5. Les barres d'erreur illustrent l'écart-type.

Méthode	Temps moyen(s)	σ(s)	Min.(s)	Max.(s)
Genetic-WFC	442 / 7 min 22 s	11	431	467
Genetic-WFC N.R.	425 / 7 min 05 s	9	415	447
A.G. sans WFC	170 / 2 min 50 s	10	162	195
A.G. sans WFC pénalisé	185 / 3 min 05 s	6	181	200
WFC uniquement	336 / 5 min 36 s	5	332	351

TABLE 6.4 – Temps de calcul moyen, écart-type, temps minimum et temps maximum, en seconde, des méthodes après 10 exécutions de 10k itérations. Les résultats correspondent à la Figure 6.5 page précédente.

Nous pouvons nous concentrer maintenant, plus particulièrement, sur le temps de calcul lié à chaque méthode, voir Figure 6.5 page précédente, Figure 6.6 page précédente et Tableau 6.4.

Grâce à ces résultats, nous pouvons remarquer que notre méthode, à savoir le *Genetic-WFC*, nécessite environ 7 min 20 s pour 10k itérations dans notre configuration. Par contre, l'*algorithme génétique sans WFC* et l'*algorithme génétique sans WFC pénalisé* ne prennent qu'environ 2 min 50 s et 3 min 5 s respectivement, soit sont 2,6 fois et 2,4 fois plus rapide. Nous en déduisons que le WFC, comme initialement envisagé, a un impact non négligeable sur le temps de calcul.

Il apparaît également que l'évolution de la valeur de fitness en fonction du temps de calcul entre le *Genetic-WFC* et l'*algorithme génétique sans WFC* est relativement comparable durant les 100 premières secondes environ. Si nous nous arrêtons sur le point de divergence vers 70 s, notre méthode atteint approximativement 94% de sa valeur finale, ce qui correspond à peu près à 1600 itérations. Cette estimation a été réalisée en normalisant les valeurs de fitness, entre la valeur minimale et maximale de la courbe.

Finalement, nous pouvons observer que les règles de pénalisation requièrent environ 15 s de temps supplémentaire et que le réencodage nécessite quant à lui 17 s environ. Cela représente un taux d'augmentation respectivement de 8% et de 4%.

6.3.2.5 Pénalisation ajustée en durée - Résultats et discussion

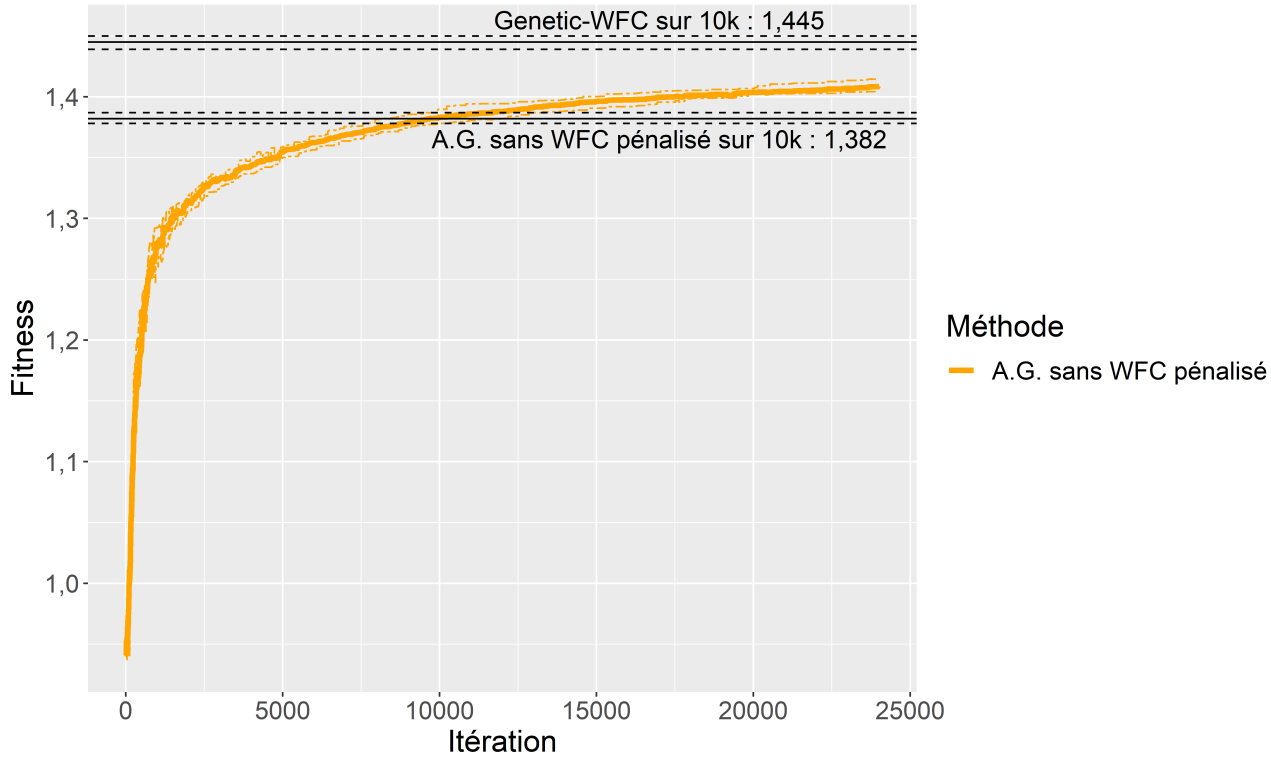


FIGURE 6.7 – Moyenne de l'évolution de la meilleure fitness pour 10 exécutions de l'*algorithme génétique sans WFC pénalisé*, sur 24k itérations. Les valeurs précédemment obtenues sur 10k itérations, dans la Sous-Section 6.3.2.3 page 144, pour le *Genetic-WFC* et l'*algorithme génétique sans WFC pénalisé* sont représentées par les lignes noires à l'horizontale, les pointillés indiquent les premiers et troisièmes quartiles. La fonction d'évaluation du parcours de l'agent correspond à $F_{Pas} = N + S - C$. La fitness de l'*algorithme génétique sans WFC pénalisé*, est rapportée sans pénalité, ce qui permet de comparer les diverses approches.

Méthode	Total	Nouveauté	Sécurité	Complexité
A.G. sans WFC pénalisé	1,408 (0,007)	0,543 (0,008)	0,911 (0,002)	-0,046 (0,003)

TABLE 6.5 – Moyenne et (écart-type) des meilleurs scores atteints après 10 exécutions de chaque méthode, sur 24k itérations, arrondie à la troisième décimale pour la moyenne et pour l'écart-type. La fonction d'évaluation du parcours de l'agent correspond à $F_{Pas} = N + S - C$. Les résultats correspondent à la valeur finale de la courbe de la Figure 6.7.

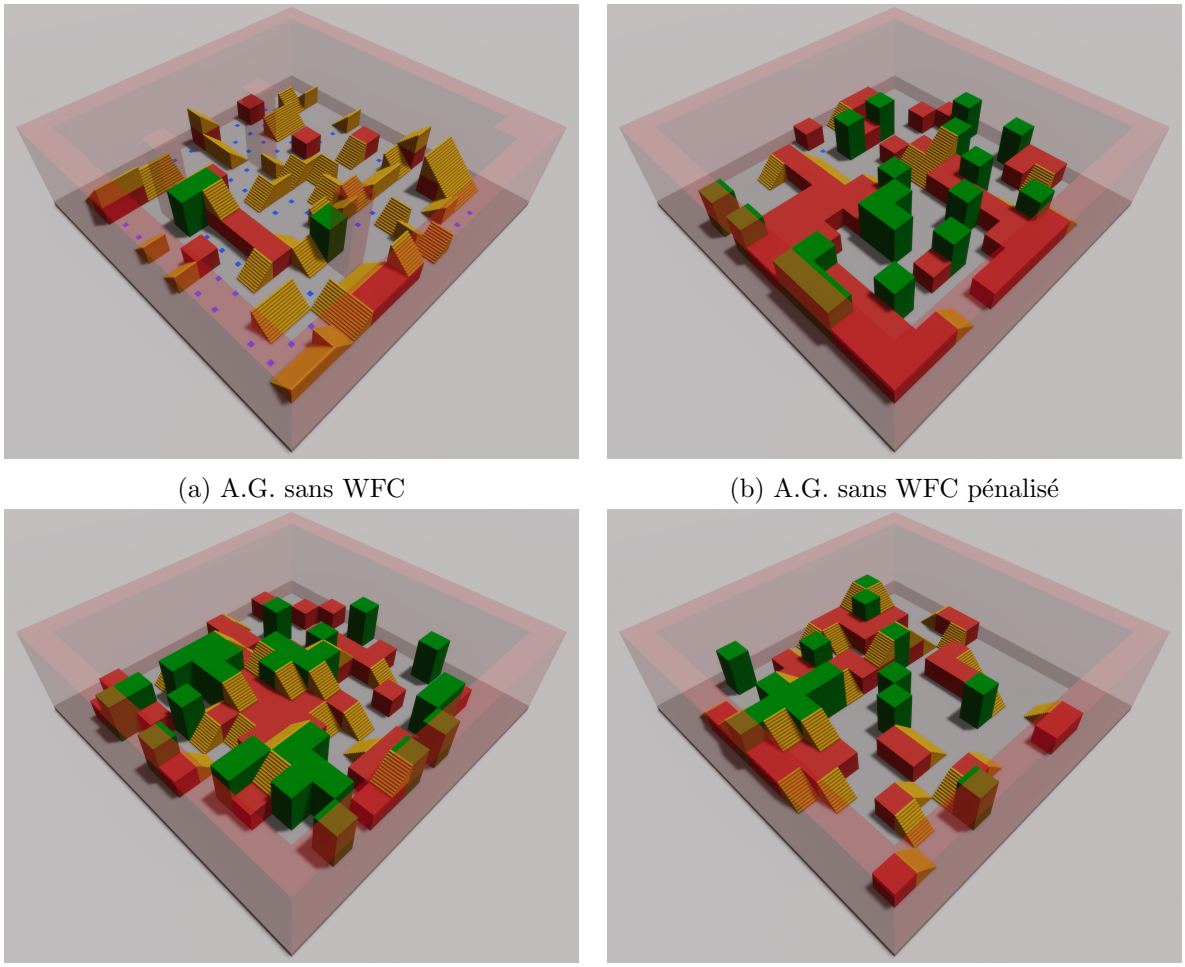
Méthode	Temps moyen(s)	σ(s)	Min.(s)	Max.(s)
A.G. sans WFC pénalisé	436 / 7 min 16 s	23	416	489

TABLE 6.6 – Temps de calcul moyen, écart-type, temps minimum et temps maximum, en seconde, de l'*algorithme génétique sans WFC pénalisé* après 10 exécutions de 24k itérations. Ces valeurs correspondent à cette méthode exécutée Figure 6.7 page précédente.

Nous avons relevé précédemment dans la Section 6.3.2.3 page 144, que la courbe de l'*algorithme génétique sans WFC pénalisé*, était inférieure à notre méthode *Genetic-WFC*, pour le même nombre d'itérations. Cependant, elle semblait continuer sa progression. Nous avons également remarqué, dans la Section 6.3.2.4 page 147, que cette méthode pénalisée était 2,4 fois plus rapide que la nôtre. Nous avons donc choisi d'ajuster le nombre d'itérations afin de correspondre au même temps de calcul en moyenne. Nous avons, de ce fait, exécuté l'*algorithme génétique sans WFC pénalisé* sur 24k itérations, afin d'atteindre un temps similaire de calcul à notre méthode. Le résultat obtenu est illustré par la Figure 6.7 page précédente et le Tableau 6.5 page précédente, et le temps de calcul par le Tableau 6.6.

Nous pouvons constater, que finalement, la moyenne atteint une meilleure valeur, mais qu'elle reste, néanmoins, inférieure à notre *Genetic-WFC* malgré un temps de calcul similaire. Malgré nos quelques règles de malus, le fait de pénaliser la valeur de fitness semble ralentir l'obtention d'une valeur égale. Il en résulte que l'algorithme génétique doit corriger en permanence les erreurs de placements, ce qui entraîne une progression plus lente.

6.3.2.6 Comparaison des niveaux - Résultats et discussion



(a) A.G. sans WFC

(b) A.G. sans WFC pénalisé

(c) WFC uniquement

(d) Genetic-WFC

FIGURE 6.8 – Illustration du meilleur niveau correspondant à la méthode associée, pour 10 exécutions sur 10k itérations, maximisant la nouveauté et la sécurité avec la plus faible complexité, d'après les calculs effectués dans la Sous-Section 6.3.2.3 page 144.

Nous allons à présent procéder à l'examen des niveaux de jeu, les mieux notés par l'agent, obtenus par les calculs de la Sous-Section 6.3.2.3 page 144. Ces niveaux vont nous permettre de mieux comprendre les performances relatives à chaque méthode.

La Figure 6.8a montre le meilleur résultat de l'*algorithme génétique sans WFC*. Nous pouvons voir que cette méthode a placé beaucoup d'escaliers. La moyenne obtenue sur ces métriques dans le Tableau 6.3 page 144 illustrent ces placements. Les escaliers sont très utiles pour optimiser à la

fois la nouveauté et la complexité. En effet, ils sont à la fois navigables, c'est-à-dire fournissent un retour positif pour la nouveauté, et bloquent la vision, ce qui aide à maximiser la sécurité. Néanmoins, cette stratégie a une limite, car le joueur doit revenir en arrière après avoir emprunté un escalier non connecté, ce qui pénalise le score de nouveauté. Nous pouvons également noter que l'algorithme a utilisé des modules de « bordure » à l'intérieur du niveau, car rien ne l'empêchait de le faire. Il a également placé une multitude de points de départ pour l'agent, augmentant ainsi la surface au sol globale. Il est important de noter que cela n'a pas d'influence sur la faisabilité de la simulation, car seul le premier *spawn* détecté est retenu comme point de départ du parcours. Cependant, même si ce résultat présente au final une valeur de fitness supérieure à notre méthode, il ne s'agit pas du type de niveau que nous voulons générer, du fait des aberrations du placement des escaliers, entre autres.

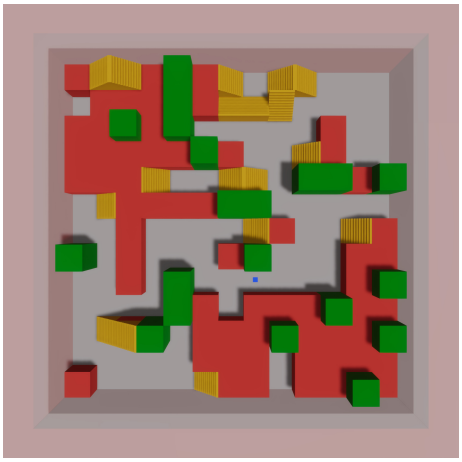


FIGURE 6.9 – Un autre niveau obtenu par l'*algorithme génétique sans WFC pénalisé*, parmi les 10 exécutions sur 10k itérations, maximisant la nouveauté et la sécurité avec la plus faible complexité, d'après les calculs effectués dans la Sous-Section 6.3.2.3 page 144. Des escaliers ne menant nulle part sont visibles en haut du niveau.

La Figure 6.8b page précédente, quant à elle, représente le niveau final de l'algorithme pénalisé. Nous constatons, que même si la disposition des éléments est sans erreurs, et ressemble à un résultat du WFC, le score reste inférieur à notre méthode, voir Tableau 6.3 page 144. De plus, il arrive que des fautes de placements surviennent encore en fin d'exécution, même avec 10k itérations, voir Figure 6.9. Cela s'explique par le fait que nous ne corrigeons pas les erreurs directement. Il est intéressant également de remarquer que des escaliers ont été disposés en pyramide, car nous avons seulement pénalisé la navigabilité du haut et du bas de cet élément.

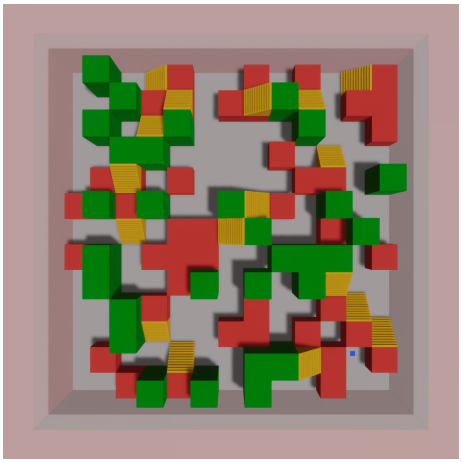


FIGURE 6.10 – Exemple de niveau obtenu aléatoirement par une exécution d'un WFC exclusivement.

La Figure 6.8c page 151 illustre le meilleur niveau pour le *WFC uniquement* et expose un résultat plus harmonieux que l'*algorithme génétique sans WFC*, malgré un score minimal selon toutes les méthodes évaluées, voir Tableau 6.3 page 144. Si nous regardons plus précisément les scores distincts des heuristiques, nous pouvons voir que le *WFC uniquement* a été nettement freiné par la nouveauté. Comme déjà mentionné, les fréquences d'apparition initiales du WFC semblent restreindre la possibilité d'obtenir un niveau correspondant aux paramètres voulus de la simulation. La Figure 6.10 illustre un niveau obtenu par l'exécution d'un seul WFC, à titre d'exemple. Nous pouvons y constater pourquoi la navigabilité est difficile à conserver pour cette méthode. En effet, nos règles d'adjacences permettent de créer des portions de niveaux localement navigables, les escaliers étant connectés correctement par exemple, mais réaliser un niveau globalement navigable est beaucoup plus compliqué.

Enfin, si nous nous intéressons au résultat de notre méthode *Genetic-WFC*, voir Figure 6.8d page 151, nous pouvons voir le niveau le mieux adapté aux contraintes, selon notre point de vue. En effet, le score de nouveauté en moyenne est le plus élevé par rapport aux autres méthodes, voir Tableau 6.3 page 144, et tous les *greyblocks* de hauteur 1 et les cases au sol peuvent être atteints. En ce qui concerne les assets de hauteur 2, certains sont navigables. D'autres sont dispersés dans le niveau bloquant la vue, assurant ainsi la sécurité et limitant la complexité perçue. Finalement, l'utilisation du WFC avec un algorithme génétique permet l'obtention d'une certaine qualité structurelle et d'un niveau potentiellement jouable.

6.3.2.7 Présentation du meilleur niveau Genetic-WFC - Résultats et discussion

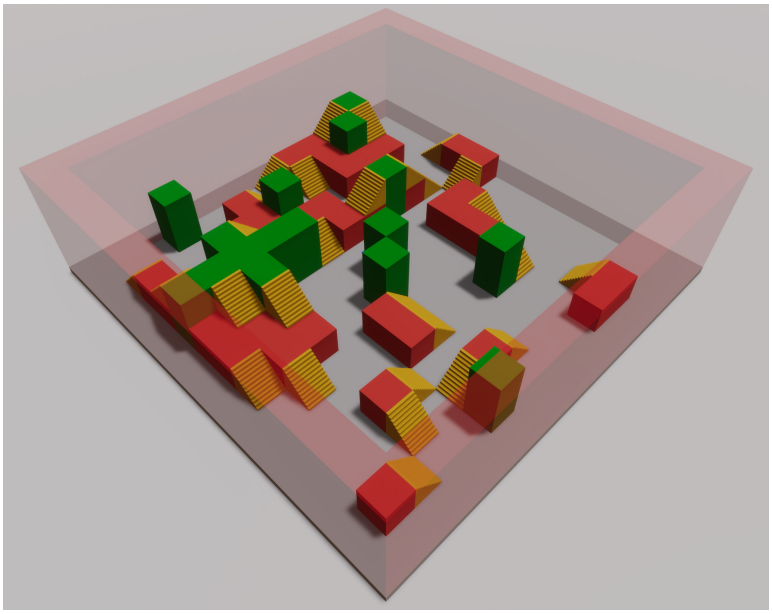


FIGURE 6.11 – Le meilleur niveau de notre méthode *Genetic-WFC*, pour 10 exécutions de 10k itérations, maximisant la nouveauté et la sécurité avec la plus faible complexité, d'après les calculs effectués dans la Sous-Section 6.3.2.3 page 144.

Pour terminer l'analyse des résultats de cette expérience, nous allons détailler à présent les données propres au meilleur niveau obtenu grâce à notre méthode *Genetic-WFC*, voir Figure 6.11, d'après les calculs de la Sous-Section 6.3.2.3 page 144.

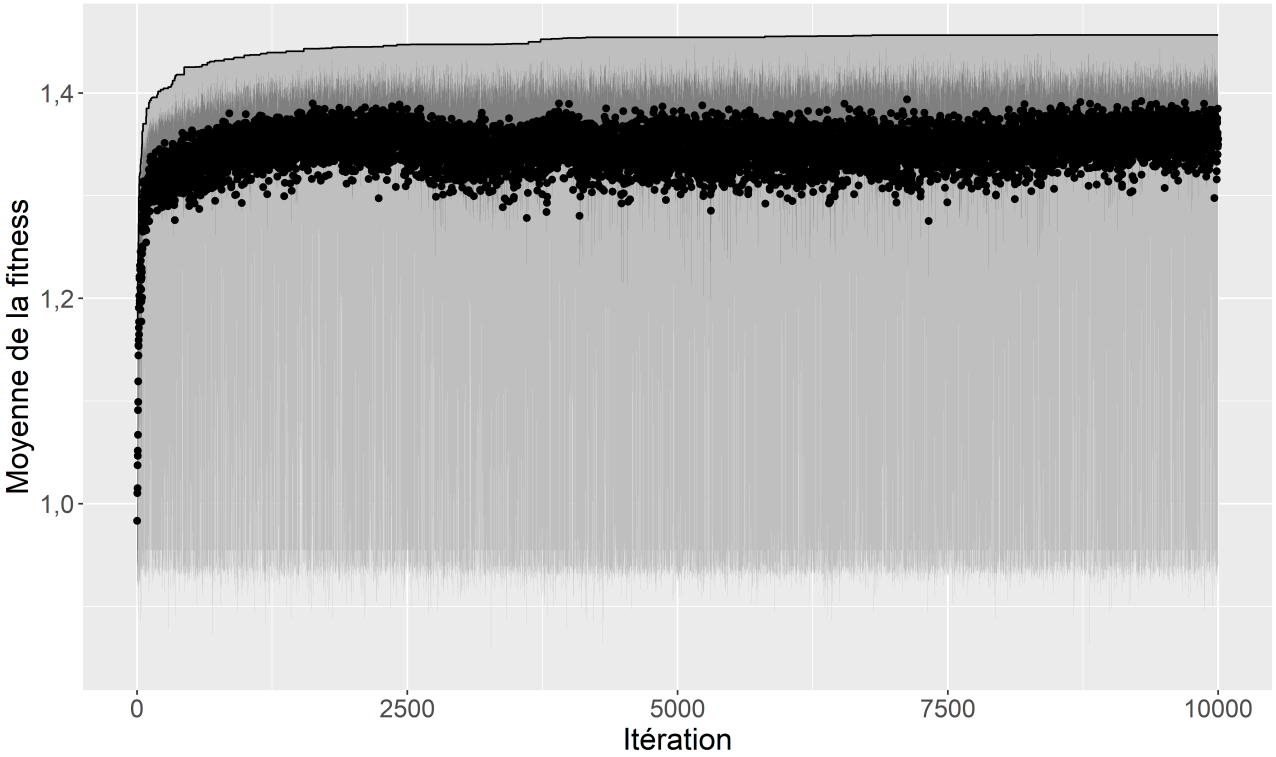
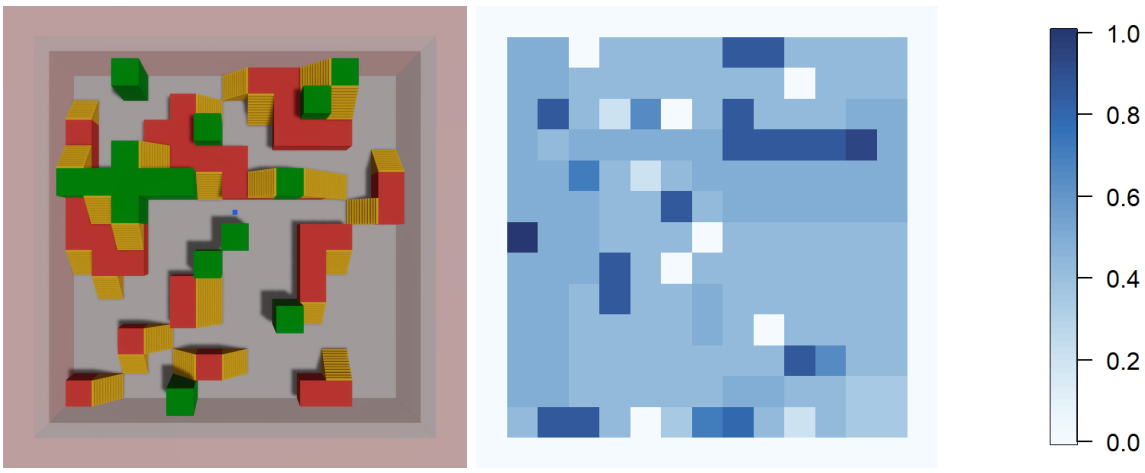


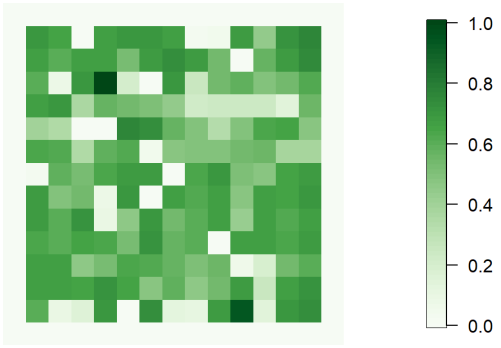
FIGURE 6.12 – Évolution de la moyenne de la valeur de fitness de la population, par itération. Il s'agit de l'exécution du *Genetic-WFC* ayant permis d'obtenir le niveau avec le meilleur score, illustré Figure 6.11 page précédente. La ligne noire en haut de la figure montre l'évolution de la valeur maximale de la fitness. Un point noir correspond à la moyenne de la fitness de la population pour une itération. La plage entre le seuil minimal et maximal de la valeur de fitness est représentée par le gris clair. La plage entre le premier et troisième quartile de la valeur de fitness est illustrée par le gris foncé. Les niveaux injouables, c'est-à-dire sans point de départ, et donc non simulés, ne sont pas pris en compte dans le calcul de la valeur de fitness moyenne.

Nous observons, sur la Figure 6.12, que la trajectoire de la moyenne n'évolue plus à partir de 2500 itérations, malgré un léger sursaut par la suite, vers 4000 itérations. Il semble que l'algorithme génétique ait des difficultés à optimiser l'ensemble de sa population, passé un certain stade. Ceci a pour conséquence d'entraîner une stabilité de la valeur de fitness maximale.

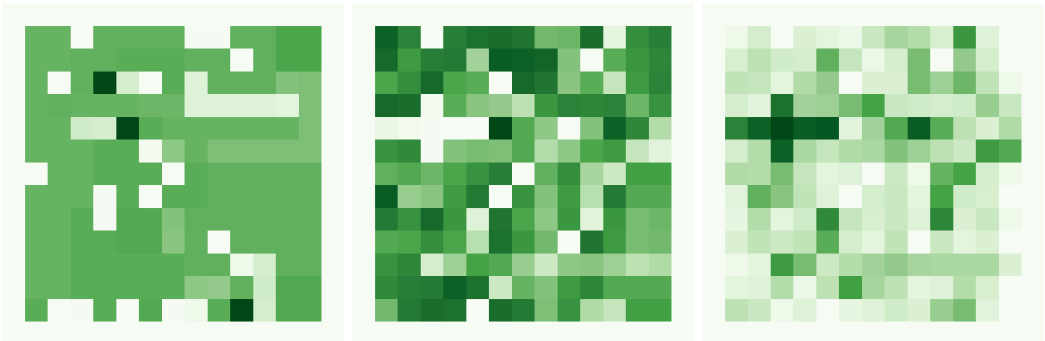


(a) Vue aérienne du niveau (b) Navigation de l'agent

FIGURE 6.13 – À gauche, vue aérienne du niveau de la Figure 6.11 page 154. À droite, répartition du nombre de fois où une case a été visitée lors du parcours de l'agent, sur la grille de ce même niveau. Les valeurs sont normalisées entre 0 et 1.



(a) Fitness du niveau



(b) Nouveauté (c) Sécurité (d) Complexité

FIGURE 6.14 – Au-dessus, représentation de la moyenne, sur chaque case, de la valeur de fitness du niveau (a). En dessous, distribution des différents scores associés aux trois heuristiques (b) (c) (d) et obtenus lors du parcours de l'agent, sur chaque case du niveau. Valeurs normalisées entre 0 et 1.

Nous allons à présent nous pencher sur les projections des données de l'agent sur la grille, obtenues grâce à son parcours du niveau, voir Figure 6.13 page précédente et Figure 6.14 page précédente. Les répartitions des différentes métriques permettent d'identifier le parcours de l'agent, motivé par une recherche de la nouveauté, mais également d'observer les endroits stratégiques concernant les scores des heuristiques d'évaluation.

Concernant la navigation, voir Figure 6.13b page précédente, l'illustration indique bien que l'agent navigue sur l'ensemble accessible du niveau. Nous remarquons les points de passage les plus fréquentés, à savoir une valeur supérieur à 0,7 environ. Par exemple, en haut à droite du niveau, il semble qu'un couloir ait été parcouru de nombreuses fois. Il est intéressant de relever, que ces cases les plus visitées, pourraient servir éventuellement d'emplacements d'objets ou d'ennemis pour rendre ces endroits plus stimulants.

Par ailleurs, nous constatons que la formule de notre fitness, $F_{Pas} = N + S - C$, est bien représentée au travers des trois illustrations, voir Figures 6.14b 6.14c 6.14d page précédente, à savoir maximiser la nouveauté et la sécurité et minimiser la complexité. Nous y discernons également les cases du niveau sur lesquelles les valeurs sont les plus concentrées.

Concernant la nouveauté, les cases $> 0,9$ indiquent des emplacements peu visités, mais qui ont une forte valeur associée, car l'agent a probablement parcouru suffisamment le niveau avant de retourner sur ces mêmes cases.

La sécurité, quant à elle, est intéressante pour exposer les endroits où l'agent s'est senti le plus en sécurité, c'est-à-dire là où sa vision est restreinte. La valeur est effectivement la plus faible dans les cases accessibles situées en hauteur 2, car la vue porte sur l'ensemble du niveau, voir les cases $< 0,3$ en haut à gauche. Les couloirs au sol, quant à eux, sont le plus représentés par une valeur élevée de sécurité. La structure globale du niveau au sol est, en effet, plutôt étriquée.

Pour finir, la complexité a bien été minimisée d'après les faibles valeurs visibles. Cependant, contrairement à la sécurité, elle est au plus haut sur les endroits accessibles en hauteur 2, voir les cases $> 0,7$ en haut à gauche. Ces rares cases offrent une vision large sur le niveau, seul endroit où la complexité peut émerger. Comme le niveau favorise la sécurité, l'impact de la complexité est limité, car la vision est fortement restreinte dans l'ensemble du niveau.

Grâce à ces résultats, nous pouvons en conclure que nos heuristiques influencent bien la structure

et le placement des objets dans le niveau, et ce, de manière pertinente.

6.3.2.8 Synthèse des observations

Cette expérience nous a permis de mettre en avant les capacités et les performances de notre méthode *Genetic-WFC* au travers de comparaisons avec des approches similaires et de l'analyse du meilleur niveau obtenu.

Tout d'abord, nous avons constaté que l'algorithme génétique de notre méthode *Genetic-WFC* pilote effectivement le WFC et influence la génération des niveaux. L'emploi du réencodage améliore également l'optimisation avec le WFC dans notre approche.

D'un autre côté, nous avons observé que la structure des niveaux reflète bien les paramètres des heuristiques de l'agent dans les optimisations par algorithme génétique. Nous pensons que les niveaux de jeu sont donc générés dans l'objectif de fournir une certaine expérience de jeu, grâce à l'évaluation par la simulation d'un parcours du point de vue du joueur.

Comme l'avions anticipé, le temps de calcul se trouve être fortement impacté par l'utilisation du WFC. Cependant, notre méthode reste, néanmoins, plus performante qu'une pénalisation classique malgré un temps ajusté.

D'autre part, nous avons pu voir qu'un algorithme génétique seul, sans opérateur de réparation et de malus, atteint une valeur maximale moyenne légèrement supérieure à la nôtre. Cependant, les niveaux générés sans aucune pénalisation obtiennent, certes, une meilleure valeur de fitness, mais surexploitent certains modules et fournissent des niveaux comportant des aberrations de placement d'escaliers, par exemple.

Finalement, nous pensons que l'utilisation du WFC se révèle être un choix judicieux pour obtenir des niveaux sans erreurs de placement d'objets, offrant ainsi une certaine qualité structurelle. Nous considérons également que nos heuristiques influencent de manière pertinente la génération du niveau afin de proposer une expérience de jeu spécifique.

6.3.3 Expérience n°3 - Exploration de l'espace de génération

6.3.3.1 Objectif de l'expérience

La finalité de cette dernière expérimentation est d'analyser la diversité des expériences de jeu que peut proposer notre méthode avec l'exploration de l'espace de génération de niveau, selon différents coefficients de pondération pour nos heuristiques. De ce fait, nous cherchons à nous assurer que nos mesures d'évaluation sont pertinentes et peuvent bien modifier la structure du niveau afin de proposer une certaine expérience de jeu, et ainsi aboutir à divers résultats. Nous espérons également identifier les corrélations entre l'expérience de jeu recherchée, c'est-à-dire selon nos coefficients de pondération, et la structure du niveau, c'est-à-dire les assets effectivement placés dans le niveau par la méthode itérative.

6.3.3.2 Méthode et protocole d'expérimentation

Afin d'évaluer la variété des expériences de jeu que peut proposer notre méthode, nous avons décidé d'échantillonner l'espace des niveaux possibles, à l'aide des résultats associés à différentes valeurs de coefficients de pondération pour la nouveauté P_N, la sécurité P_S et la complexité P_C. De ce fait, le calcul de la fitness par « pas de l'agent » peut être illustré de la manière suivante : $F_{Pas} = P_N * N + P_S * S + P_C * C$.

Dans cette expérimentation, nous avons choisi d'explorer des niveaux variés selon certains de ces coefficients de pondération. En effet, comme nous ne pouvons pas présenter ici toutes les dimensions de ces paramètres, nous nous sommes limités à certaines valeurs afin de proposer des résultats que nous avons jugés suffisamment pertinents dans la représentation des capacités d'expression de notre méthode et qui offrent également des niveaux intéressants et variés.

Les paramètres utilisés dans cette expérience ont ainsi été les suivants :

→ Pour chaque $P_N \in \{ 0 ; 0,125 ; 0,25 ; 0,5 ; 0,75 ; 1 \}$, nous effectuons chaque variation de P_S et P_C entre -1 et 1 par incrément de 0,5. Cela représente 25 niveaux par P_N, soit un total de 150 niveaux différents à échantillonner. Le Tableau 6.7 page suivante illustre les 25 variations des heuristiques P_S et P_C pour une valeur choisie de P_N.

Nous avons fait le choix de ne pas proposer une pondération de nouveauté en dessous de 0, car

nous pensons que cela inciterait fortement l'agent à ne pas explorer le niveau et à rester plutôt sur la case de départ. Ce cas de figure empêcherait toute navigation sur le terrain et donc l'évaluation avec l'obtention des différentes métriques, dans le but d'optimiser le niveau.

Nous avons également choisi d'explorer $P_N = 0,125$. En effet, nous pensons qu'avoir un P_N relativement faible, permettrait de limiter l'impact de la nouveauté sur l'expérience recherchée et donnerait ainsi aux deux autres heuristiques la possibilité de s'exprimer davantage dans les résultats. Nous avons pris cette décision dans l'idée de constater l'influence directe de la sécurité et de la complexité sur la structure des niveaux, et ainsi proposer des résultats intéressants à observer.

#	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
P_S	-1	-1	-1	-1	-1	-,5	-,5	-,5	-,5	-,5	0	0	0	0	0	,5	,5	,5	,5	,5	1	1	1	1	1
P_C	-1	-,5	0	,5	1	-1	-,5	0	,5	1	-1	-,5	0	,5	1	-1	-,5	0	,5	1	-1	-,5	0	,5	1

TABLE 6.7 – Les 25 variations des valeurs de pondération des heuristiques de sécurité et de complexité, selon une valeur choisie de nouveauté.

En définitive, nous avons effectué 150 exécutions de notre *Genetic-WFC* en *multithread*. Les paramètres de génération qui ont servi à produire toutes ces combinaisons de pondérations, sont illustrés Tableau 6.8 page suivante. En résumé, chaque niveau a résulté de 5000 itérations, avec une taille de population $Pop_{min} = 50$ et $Pop_{max} = 60$. La probabilité de mutation a été fixée à $p_{mut} = 0,04$, et la probabilité de croisement à $p_{cross} = 0,75$. La grille a été fixée à 15 sur 15 avec les 7 modules vus dans la Sous-Section 6.3.1 page 138 et avec l'inclusion des bordures. La simulation de l'agent, quant à elle, a été effectuée avec la variation des trois pondérations et sur 1125 pas.

Nous avons choisi de calculer notre optimisation sur 5000 itérations, car nous avons jugé que ce paramètre était suffisant pour échantillonner les 150 niveaux. En effet, nous avons vu dans l'expérience 2 et avec les résultats de la Sous-Section 6.3.2.3 page 144, que 5000 itérations permettaient d'atteindre 98% de la valeur maximale de fitness.

Paramètres de génération		
A.G.	Population (min/max)	50/60
	Itérations	5000
	Croisement	75%
	Mutation (par gène)	4%
WFC	Taille	15x15
	Modules	7
	Bordures	Oui
Agent	Nouveauté	P_N
	Sécurité	P_S
	Complexité	P_C
	Nombre de pas	1125

TABLE 6.8 – Paramètres généraux de génération pour le *Genetic-WFC* dans cette expérience.

Dans les sections suivantes, nous allons analyser les résultats obtenus, à l'aide de quelques mosaïques illustrant les niveaux générés. Puis, nous nous intéresserons à la mesure de la navigabilité des niveaux, à savoir le ratio de cases traversables par l'agent dans un résultat.

6.3.3.3 Mosaïques des niveaux - Résultats et discussion

Dans cette section, nous avons sélectionné trois mosaïques sur les six réalisées. L'ensemble des mosaïques, illustrant les 150 niveaux produits avec les six valeurs de P_N, peut être retrouvé en Annexe C page 225.

Les mosaïques retenues illustrent visuellement les niveaux générés selon les trois valeurs de P_N associées, respectivement $P_N = 0,5$, $P_N = 0,25$ et $P_N = 0,125$, voir Figure 6.9 page suivante, Figure 6.10 page 164 et Figure 6.11 page 165. Ces représentations nous permettent de distinguer les divergences selon la disposition et la quantité des différents modules. En effet, ces trois mosaïques, nous semblent particulièrement représentatives pour l'observation de la diversité des niveaux que peut proposer notre méthode.

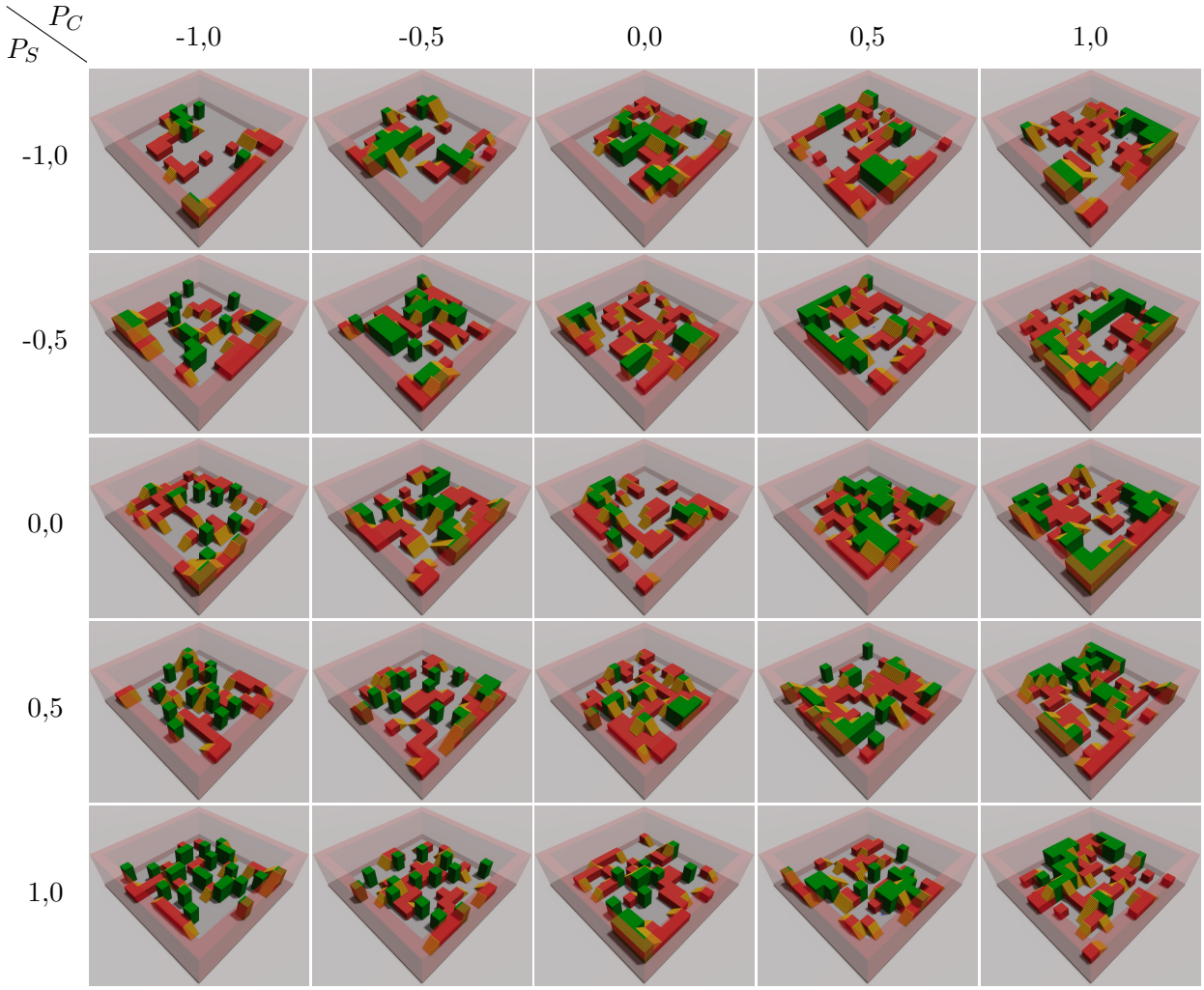


TABLE 6.9 – Mosaïque des résultats pour $P_N = 0,5$.

Nous allons, tout d'abord, examiner en détail les niveaux associés à $P_N = 0,5$. Ainsi, en observant la Figure 6.9, nous pouvons constater, l'influence de la complexité et de la sécurité sur la disposition des niveaux générés.

Dans la première ligne, $P_S = -1,0$, nous essayons de générer des niveaux avec un score de sécurité très bas. De tels niveaux peuvent être relativement vides, comme celui en haut à gauche. En effet, le joueur peut facilement s'y déplacer, mais il n'a pas la capacité de s'y dissimuler entièrement, par manque de densité de modules. Cependant, en allant vers la droite de cette première ligne, nous passons d'une complexité faible à une complexité élevée. De ce fait, nous observons que le générateur ajoute de plus en plus de modules, remplaçant les cases vides, créant des niveaux à la fois peu sûrs, mais

plus encombrés. Nous pouvons également constater que tous les modules de hauteur 2 sont navigables, offrant ainsi des positions à faible sécurité avec une vision plus étendue et favorisant également la valeur de complexité perçue par l'agent.

Par la suite, si nous observons la première colonne de la Figure 6.9 page précédente, $P_C = -1, 0$, nous constatons que le premier niveau, en haut à gauche, est relativement vide, non sécurisé et non complexe. Si nous regardons plutôt vers le bas de la colonne, la sécurité est accrue tout en gardant une faible complexité. Au fur et à mesure que la sécurité est récompensée, le générateur rajoute des modules limitant le champ de vision afin de maximiser cette valeur. De plus, cela lui permet également de pénaliser le moins possible la complexité perçue par l'agent. En effet, comme il ne peut plus distinguer un changement de géométrie, dû à une vision limitée, il ne peut donc plus être sanctionné sur cette heuristique qu'est la complexité. Il s'agit ici d'une synergie intéressante entre ces deux paramètres, que ce soit dans le cas $P_C = -1, 0$ et $P_S = -1, 0$ qui va favoriser l'émergence d'un niveau vide ou dans le cas $P_C = -1, 0$ et $P_S = 1.0$ qui va mener vers un niveau plus labyrinthique.

La colonne de droite de la Figure 6.9 page précédente, $P_C = 1, 0$, est plus difficile à interpréter. En effet, au fur et à mesure que nous descendons dans les niveaux, l'algorithme recherche quelque chose d'assez contradictoire. En effet, il s'efforce d'augmenter la sécurité tout en maintenant un haut niveau de complexité. Un résultat, qui maximise la complexité et la sécurité, a ainsi pour objectif : d'un côté de permettre au joueur de voir le plus loin possible afin de distinguer des structures complexes ; mais d'un autre côté de limiter la visibilité afin de maintenir un haut niveau de sécurité. Dans cette optique, la différence entre les niveaux du haut et du bas, réside dans le placement des assets de hauteur 1. En effet, en bas, pour le niveau le plus sûr, les modules de hauteur 1 créent des chemins d'une largeur en général d'une case, et donc proposent des parcours plus étroits au sol. En outre, les assets de hauteur 1 ne forment pas de plateformes de deux cases par deux cases ou plus. En revanche, pour le niveau du haut, celui considéré comme le plus « dangereux », nous pouvons voir de plus larges plateformes de hauteur 1. Éviter les plates-formes pourrait être un moyen pour le générateur de cumuler du score lié à la sécurité grâce aux chemins étroits, car la plupart d'entre eux offrent une vision plus restreinte. Cependant, la complexité, quant à elle, dans le niveau du bas, semble être présente dans la disposition plus chaotique des éléments. Au final, le résultat $P_C = 1, 0$ et $P_S = 1, 0$ est particulièrement intéressant, car il semble présenter un niveau plus varié. En effet, il possède des positions peu sûres en hauteur 2, avec une vue dégagée permettant de percevoir de la complexité sur l'ensemble du niveau. D'un autre

côté, le niveau présente des chemins plus étroits, donc sécurisés, à la hauteur 1 et au sol.

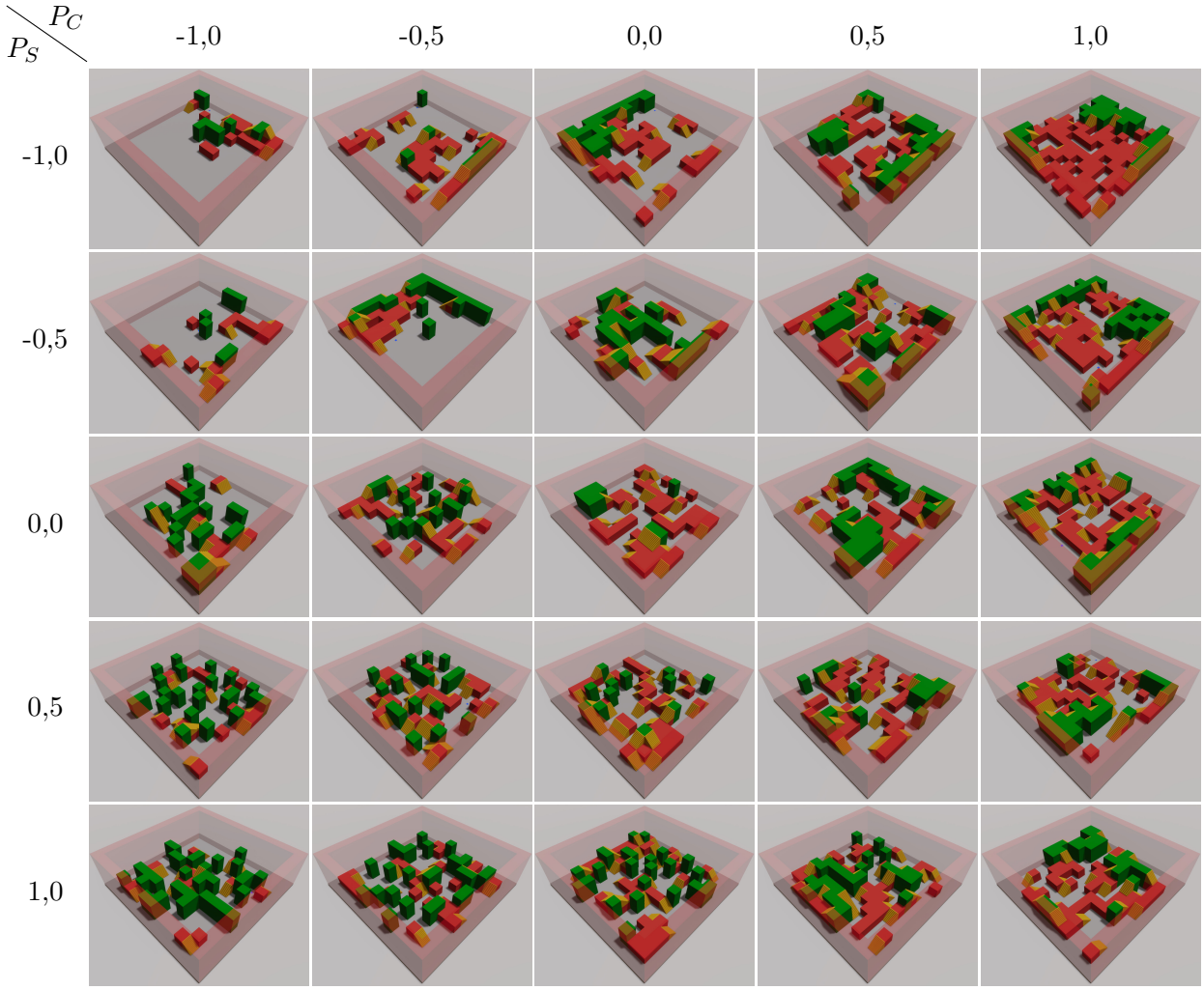


TABLE 6.10 – Mosaïque des résultats pour $P_N = 0,25$.

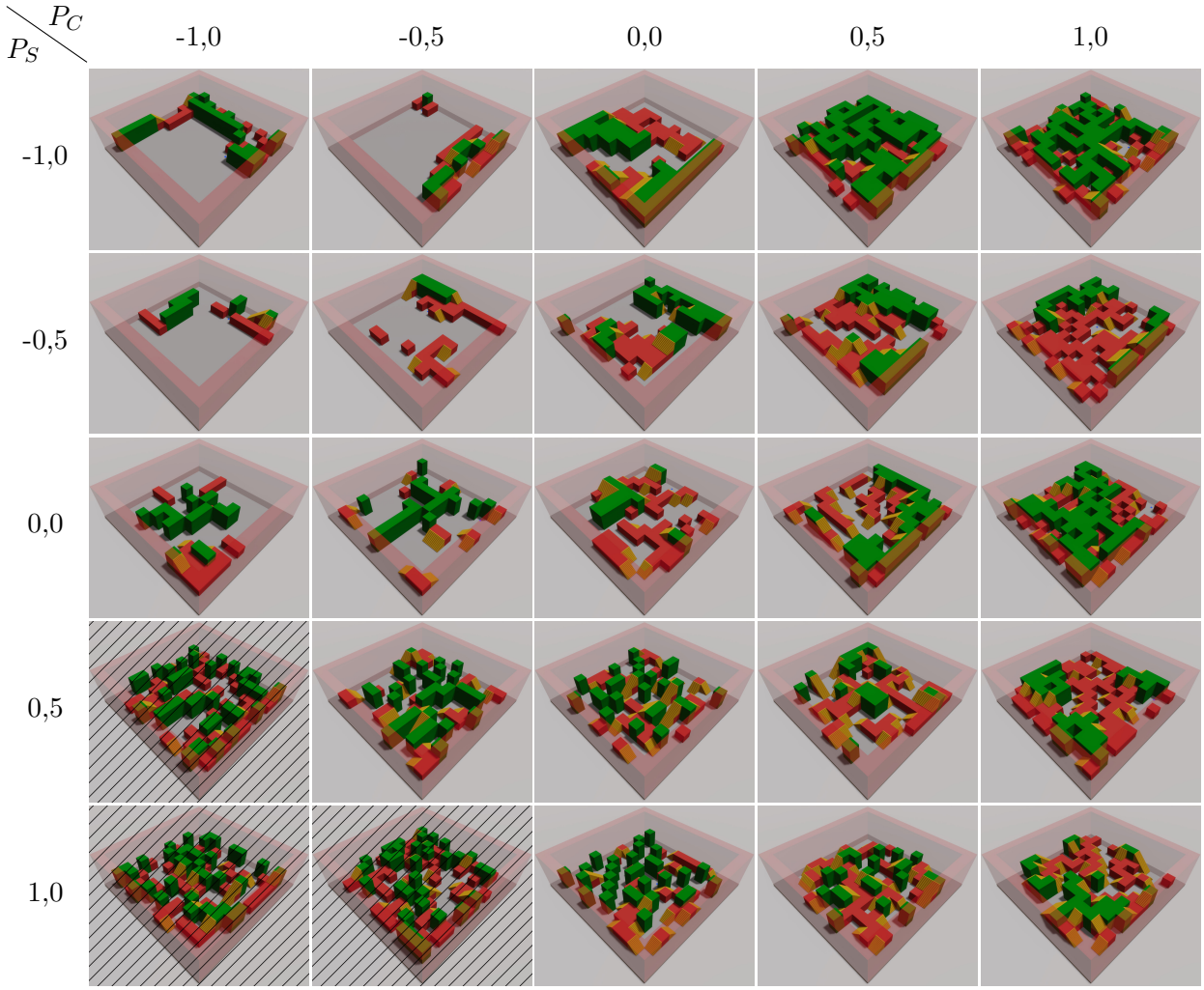


TABLE 6.11 – Mosaïque des résultats pour $P_N = 0,125$. Les niveaux hachurés représentent les niveaux non navigables, c'est-à-dire qui possèdent un point de départ bloqué.

Nous allons à présent nous pencher sur l'évolution des niveaux produits avec $P_N = 0,5$, $P_N = 0,25$ et $P_N = 0,125$, à l'aide des trois mosaïques correspondantes, voir Figure 6.9 page 162, Figure 6.10 page précédente et Figure 6.11.

Tout d'abord, nous remarquons rapidement que la baisse de la nouveauté entraîne l'apparition de résultats dont la particularité est accentuée, par exemple, que ce soit la suppression de géométrie dans le cas de coefficients négatifs, l'apparition de « labyrinthes » dans le cas d'un niveau très sécurisé, ou encore l'apparition de surfaces navigables en hauteur dans le cas de niveaux ayant une complexité élevée. Cette observation suggère que les heuristiques de sécurité et de complexité, s'expriment davantage

et sont plus mises en avant par l'aspect visuel du niveau. Nous pouvons ainsi en déduire que, lorsque la nouveauté est trop importante, elle prédomine les deux autres paramètres et atténue leur impact sur la structure du niveau.

D'un autre côté, nous pouvons observer que la nouveauté semble influencer la navigabilité du niveau. En effet, si le coefficient de nouveauté est trop faible, par exemple pour $P_N = 0,125$, nous observons l'apparition de départs bloqués, de création de « trous » dans le niveau ou encore la formation de surfaces « labyrinthiques » navigables en hauteur, voir $P_C = 1,0$ avec $P_S \in [-1;0]$ dans la Figure 6.11 page précédente. Ainsi, la nouveauté semble assurer la création d'un résultat optimisé pour sa navigabilité, mais au détriment des caractéristiques liées à la sécurité et à la complexité.

En regardant les trois mosaïques ici présentes, nous pouvons en déduire qu'un coefficient de la nouveauté, proche de $P_N = 0,25$, permettrait d'avoir des disparités prononcées dans le placement des modules tout en produisant des niveaux hautement navigables. D'un autre côté, $P_N = 0,125$ semble être le paramètre qui amène le plus de variabilité apparente au détriment de l'apparition de certains niveaux toutefois non navigables. Finalement, nous pouvons en conclure qu'employer un P_N proche de $0,25$ pour la nouveauté, paraît être le plus intéressant à observer pour générer des niveaux les plus disparates visuellement.

Il est aussi intéressant de noter que notre méthode semble être réticente à supprimer de la géométrie. En effet, seule l'association de la sécurité et de la complexité permet de créer des surfaces presque vides dans le niveau. Cependant, l'augmentation de P_N atténue également cette caractéristique.

Par ailleurs, dans les cas où $P_C = 0,0$ et $P_S = 0,0$, la nouveauté est alors le seul critère pris en compte. Les niveaux créés présentent malgré tout de la géométrie. Ce résultat est étonnant, il semblerait que l'optimisation cherche à rajouter des modules qui interrompent le parcours, probablement dans le but de former des chemins différents.

En dernier lieu, nous pouvons également observer l'évolution générale de P_N avec l'ensemble des mosaïques en Annexe C page 225. Nous constatons que la nouveauté semble uniformiser l'aspect des niveaux avec l'augmentation de son coefficient de pondération, comme mentionné précédemment. En effet, avec $P_N \geq 0,75$, les disparités visuelles entre les niveaux semblent fortement diminuer, de plus, le nombre d'assets paraît plus homogène.

6.3.3.4 Navigabilité - Résultats et discussion

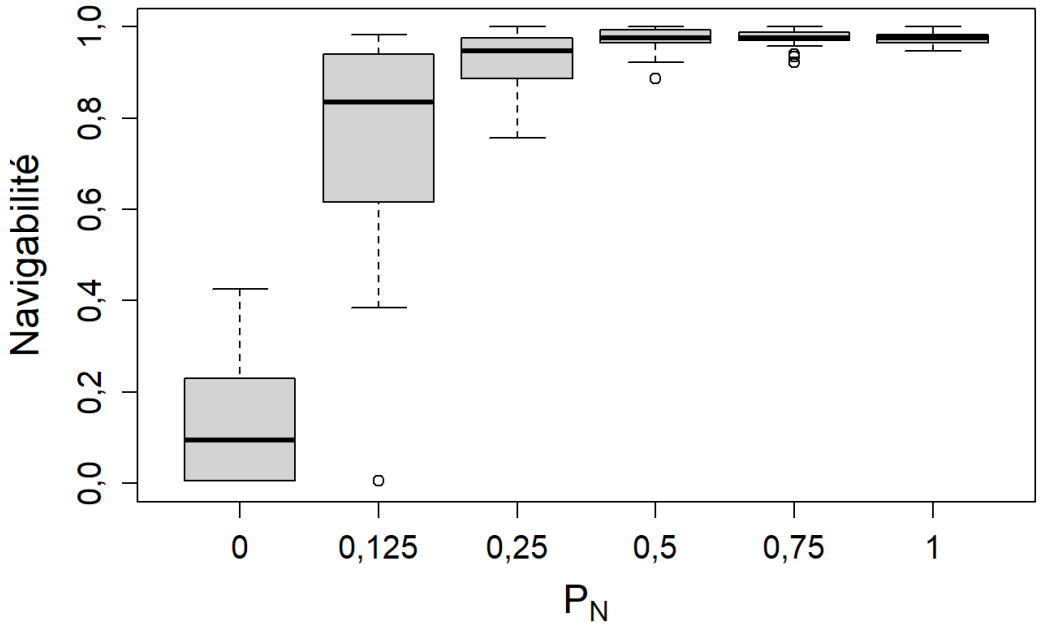


FIGURE 6.15 – Boîtes à moustaches représentant la navigabilité des 25 niveaux générés avec le *Genetic-WFC* selon chaque P_N.

Nous allons, à présent, nous intéresser plus spécialement à la navigabilité des niveaux illustrés dans la section précédente.

Pour ce faire, nous avons calculé une mesure de navigabilité par niveau, associée au pourcentage de cases traversables par l'agent, selon la formule suivante :

$$Navigabilité = \frac{1}{T}\sum_{i=1}^{T} n_i$$

Où T est le nombre total de cellules de la grille du niveau à l'intérieur des bordures, et où n est calculé de la façon suivante pour chaque cellule située également à l'intérieur des bordures :

$$n = \begin{cases} 1, & \text{si case navigable par l'agent, selon le mesh de navigation} \\ 0, & \text{sinon} \end{cases}$$

Dans le cadre de ce calcul, seul le nombre total de cellules de la grille à l'intérieur des bordures est pris en compte. En effet, nous ne considérons pas le contour du niveau dans cette mesure afin de nous concentrer, plus précisément, sur les éléments choisis par notre méthode itérative. D'autre part, le calcul de la navigabilité est uniformisé entre 0 et 1, ce qui équivaut à une valeur de pourcentage de navigabilité pour un niveau.

Le graphique Figure 6.15 page précédente représente la répartition des valeurs de navigabilité obtenues sous la forme de boîtes à moustaches.

En premier lieu, nous constatons que la navigabilité se resserre autour de 1 quand P_N augmente. En effet, plus la valeur de P_N est élevée, plus les résultats affichent une mesure de navigabilité élevée. À partir de $P_N \geq 0,25$, tous les niveaux sont navigables à plus de 75% du niveau, hors bordures. La navigabilité atteint même une valeur $\geq 90\%$ à partir de $P_N \geq 0,5$, nous pouvons ainsi en déduire que les niveaux sont presque entièrement explorables.

D'un autre côté, nous observons que $P_N = 0$ et $P_N = 0,125$ sont les seuls coefficients à proposer des niveaux non navigables, donc une navigabilité à 0. Comme remarqué sur les mosaïques précédemment, nous avons quelques niveaux non navigables, car le point de départ est bloqué entre plusieurs modules d'une certaine hauteur. En effet, notre agent est limité dans ses capacités de déplacement, il ne peut ni sauter ni traverser en diagonale. Comme l'agent ne peut bouger de la case de départ, ces niveaux ne sont donc pas simulables. Cette caractéristique semble ainsi apparaître quand la valeur de nouveauté est trop faible.

D'autre part, nous constatons qu'à partir de $P_N \geq 0,25$, tous les niveaux sont navigables au-delà du point de départ. Cependant, $P_N = 0,125$ est le seul à proposer une répartition de la navigabilité très large pour les niveaux, sans bloquer toutefois le point de départ, hors aberrations.

Finalement, avec l'observation de ce graphique, nous pouvons déduire une forte corrélation entre P_N et la navigabilité d'un niveau. À l'aide des mosaïques, vues précédemment dans la Sous-Section 6.3.3.3 page 161, nous avions déjà constaté cette relation. Il semblerait que la nouveauté incite bien l'agent à découvrir le niveau, ce qui correspond à notre attente concernant cette heuristique, à savoir la motivation d'explorer le niveau. Pouvoir proposer des résultats en partie navigables est essentiel pour évaluer le parcours de l'agent selon les heuristiques de perception. En effet, notre méthode itérative a besoin de simuler un parcours afin d'optimiser les niveaux.

6.3.3.5 Synthèse des observations

Dans cette expérimentation, nous avons analysé la diversité des expériences de jeu que peut proposer notre méthode avec l'exploration de l'espace de génération de niveau. Les résultats ont été présentés sous forme de mosaïques, selon différents coefficients de pondération pour les heuristiques de percep-

tion. Nous avons également établi la navigabilité des niveaux avec la présentation d'un graphique de boîtes à moustaches.

L'étude des mosaïques a montré que notre méthode propose bien une structure de niveau ouvert non-linéaire dans ses résultats et que nos valeurs de perception ont un impact visible sur les niveaux. En effet, nous avons observé que nos heuristiques influencent bien la disposition, le placement et le nombre de modules dans le niveau généré. Nous avons également observé l'impact de la nouveauté sur la navigabilité dans le niveau. La variation de ces valeurs de perception permet, de ce fait, l'obtention de niveaux distincts. Nous pouvons en déduire que notre méthode parvient ainsi à proposer une certaine diversité d'expériences de jeu.

Finalement, nous avons constaté quelques particularités de nos heuristiques sur la formation des niveaux, à savoir :

- La nouveauté influence directement la navigabilité du niveau ;
- La complexité et la sécurité impactent la disposition des modules et apportent une particularité au niveau ;
- L'augmentation de la nouveauté atténue les caractéristiques structurelles dues à la complexité et à la sécurité.

6.4 Conclusion

Pour conclure, nous nous sommes appuyés sur trois expérimentations afin de confronter nos hypothèses concernant nos divers questionnements.

La première expérience, basée sur le temps de calcul du WFC, nous a conforté dans l'idée d'utiliser un nombre limité de *greyblocks*.

La deuxième expérience a comparé notre *Genetic-WFC* à d'autres méthodes similaires. Ceci nous a permis de valider le fonctionnement de notre approche, à savoir le pilotage et l'inclusion du WFC dans un algorithme génétique en tant qu'opérateur de réparation. Nous avons observé que le WFC était un choix judicieux pour produire des niveaux avec une certaine qualité structurelle. Cependant, nous avons constaté l'impact de l'utilisation de cet algorithme sur le temps de calcul. De surcroît, nous avons remarqué que le réencodage était nécessaire lors de l'utilisation du WFC. De plus, les résultats

obtenus ont permis d'analyser l'influence de nos heuristiques de perception sur la formation du niveau généré. En tenant compte des résultats, nous avons établi que notre méthode était plus performante pour proposer des niveaux jouables de qualité structurelle, qu'une approche classique de pénalisation.

La troisième et dernière expérience, axée sur l'exploration de l'espace de génération de niveau de jeu que peut proposer notre méthode, a mis en avant l'impact de nos valeurs de perception sur la composition et la disposition des assets dans les niveaux. Les résultats ont indiqué que les variations des coefficients des heuristiques produisent, en effet, une certaine diversité d'expériences de jeu.

Dans le chapitre suivant, nous allons aborder plus particulièrement la mise en perspective de tous ces résultats face à notre problématique. Nous pourrons nous interroger sur les limites de notre proposition et sur les diverses pistes à explorer éventuellement. Cette analyse nous permettra de réfléchir à l'aspect motivation et intérêt pour le joueur, avec l'inclusion d'éléments de gameplay FPS propres à un campement, afin de proposer une véritable jouabilité dans nos niveaux.

Chapitre 7

Discussion et perspectives

Contenu

7.1 Introduction

L'intention de ce chapitre est de revenir sur le travail effectué dans cette thèse et de se pencher sur les résultats, grâce aux diverses expérimentations exposées dans le chapitre précédent. Nous allons confronter notre proposition avec la problématique et ses axes de recherche. Nous allons évaluer de façon critique notre étude, en présentant les aspects positifs et négatifs. Nous discuterons également des pistes envisageables pour la poursuite des recherches sur ce sujet.

7.2 Mise en perspective des travaux de la thèse

Le fil conducteur des travaux mis en œuvre dans cette thèse, a été la question de la diversité et de la qualité dans le domaine de la PCG. Nous nous sommes penchés, plus précisément, sur la problématique suivante : « Comment générer procéduralement une forte diversité d'expériences de jeu tout en conservant une certaine qualité structurelle ? ». En d'autres termes, nous avons cherché à atteindre le plus grand espace possible de diversité d'expériences de jeu tout en limitant le plus possible les erreurs de *level design*. Nous avons ainsi centré notre étude sur la conception des niveaux de jeu et le placement d'objets. Nous nous sommes focalisés, plus particulièrement, sur la génération de niveaux 3D ouverts non-linéaires de type jeu de tir à la première personne, avec une infrastructure de campement dont le contenu est positionné sur une grille 2D.

Les recherches menées ont permis d'étudier les différents aspects de la PCG, que ce soit ses concepts ou ses méthodes. Cette documentation préalable nous a incités à proposer une approche innovante afin de répondre à notre sujet. C'est pourquoi nous avons présenté, durant cette étude, les capacités et l'évaluation de notre nouvelle méthode appelée *Genetic-WFC*. Il s'agit d'un algorithme procédural de génération de niveaux combinant un algorithme génétique avec l'algorithme de *Wave Function Collapse*, pour générer des niveaux ciblant des expériences de jeu spécifiques. Il s'agit, plus particulièrement, d'une approche *Search-Based* qui emploie l'algorithme génétique comme algorithme de recherche avec une simulation pour l'évaluation des niveaux selon l'expérience de jeu. Le WFC s'y intègre alors comme opérateur de réparation. Notre réflexion s'est articulée, par conséquent, autour de deux aspects distincts de la PCG, à savoir :

- Le *Search-Based*, associant recherche évolutionnaire et évaluation par simulation d'un joueur

synthétique, pour atteindre une expérience de jeu spécifique ;

- Le *Wave Function Collapse*, en tant que méthode constructive générique, pour créer des niveaux avec des contraintes d'adjacence ;

Au travers de différentes expériences, nous avons démontré la faisabilité et l'efficacité de notre méthode. En effet, au cours de nos investigations, nous avons validé et répondu à divers questionnements et hypothèses, en nous articulant autour de trois expérimentations. Nous nous sommes tout d'abord intéressés au temps de calcul inhérent au WFC, ce qui nous a conforté dans l'idée de n'utiliser qu'un nombre limité de *greyblocks* pour abstraire la structure du niveau. La deuxième démonstration s'est plutôt axée sur la comparaison directe de notre approche avec d'autres méthodes similaires. Les résultats obtenus ont montré que l'algorithme génétique pilote bien le WFC avec les zones de probabilités et que le réencodage est efficace et nécessaire. Toutefois, l'impact du WFC reste conséquent sur le temps de calcul. Cependant, nous avons constaté que notre *Genetic-WFC* est plus performant qu'une recherche génétique basique avec des pénalisations, tout en proposant un résultat toujours considéré comme « valide » en termes de contraintes d'adjacence. Nous avons aussi observé que le parcours de l'agent autonome, dans notre simulation, a permis d'évaluer les niveaux, et que les préférences du joueur synthétique sont bien reflétées dans la composition des assets du résultat. Les préférences correspondent aux coefficients de pondération des valeurs de perception, c'est-à-dire les heuristiques de nouveauté, de sécurité et de complexité. La dernière étude a permis d'explorer l'espace de génération de niveau dans le but d'observer la diversité des expériences de jeu que peut proposer notre méthode, en fonction des coefficients liés aux heuristiques de l'agent. Les niveaux illustrés et la mesure de navigabilité ont indiqué que la variation des coefficients pour les trois heuristiques influencent bien les divers aspects du niveau, c'est-à-dire la disposition, le placement et le nombre d'assets. Finalement, nous obtenons une certaine diversité de résultats, que nous considérons comme digne d'intérêt.

Les résultats, détaillés précédemment, ont montré que notre méthode fonctionne de manière satisfaisante et que nous obtenons des niveaux diversifiés et intéressants selon notre point de vue. En effet, la combinaison du *Search-Based* et d'une méthode constructive, à savoir le WFC, permet d'obtenir des résultats prometteurs. De plus, comme nous l'espérions, les niveaux ont été obtenus avec un minimum de contraintes et avec l'application de nos trois heuristiques. Le WFC permet avec un minimum de restrictions d'atteindre une certaine qualité structurelle. La recherche, quant à elle, avec différents co-

efficients pour les heuristiques de perception, produit des résultats distincts. Finalement, l'algorithme génétique se concentre bien sur l'évaluation de l'expérience de jeu avec l'agent pour son optimisation et délaisse la correction des erreurs structurelles du niveau au WFC. Nous présumons avoir proposé une solution efficace, répondant à notre problématique de la façon suivante :

- l'utilisation du *Search-Based* avec une simulation pour atteindre une diversité d'expériences de jeu,
- et l'emploi du WFC dans le but d'obtenir des niveaux d'une certaine qualité structurelle.

Toute réflexion faite, nous pensons que l'élaboration de l'approche *Genetic-WFC* apporte deux contributions spécifiques à la recherche dans le domaine de la génération procédurale de contenu :

- L'intégration et le pilotage du WFC dans un algorithme génétique représente un nouvel opérateur de réparation dans une optimisation sous contrainte.
- Nos trois heuristiques de perception permettent d'aboutir à une diversité d'expériences de jeu. Ces trois métriques ont été conçues pour être pertinentes du point de vue du joueur.

Les travaux spécifiés dans cette thèse, ont mené notamment à la publication d'un article dans le journal *IEEE Transactions on Games*, *Genetic-WFC : Extending Wave Function Collapse with Genetic Search*, présentant l'ensemble de la méthode [189]. Nous avons également eu l'occasion de participer à un consortium doctoral lors de la conférence *IFIP ICEC* de 2021, qui a eu lieu à Coimbra au Portugal, et d'y proposer un article [190].

Cependant, bien que nous ayons répondu à notre problématique avec des résultats positifs sur la faisabilité de notre approche, nous avons constaté que nous n'avons pas eu l'opportunité d'approfondir toutes les recherches et les pistes prévues qui auraient pu éventuellement améliorer notre méthode.

Nous avons proposé des heuristiques pour évaluer l'expérience de jeu en prenant en compte le point de vue du joueur, car cela nous semblait pertinent pour simuler des niveaux ouverts non-linéaires avec du gameplay émergent. Néanmoins, nous n'avons pas démontré l'impact réel sur le ressenti d'un véritable utilisateur, à savoir la présence d'une variabilité et d'une diversité intéressante des niveaux obtenus. Il s'agit d'une piste particulièrement importante à explorer, permettant de prouver la pertinence de ces heuristiques. Sont-elles significatives pour le joueur ? Faut-il envisager d'autres valeurs

d'évaluation ? De plus, comme nous avons choisi de nous concentrer uniquement sur l'aspect abstrait de la structure des niveaux afin d'appuyer notre combinaison d'algorithmes, le développement du gameplay d'un campement serait la suite logique à notre réflexion, à savoir l'inclusion d'ennemis, d'objets et d'objectifs. Nous pourrions ainsi mettre en pratique la multicouche du WFC pour proposer des niveaux jouables. Le but serait de nous rapprocher davantage de la véritable expérience de jeu dans un campement et de proposer une diversité plus pertinente avec une simulation plus complète. Finalement, le résultat rendrait alors possible la validation de nos heuristiques d'évaluation avec une expérience utilisateur, par exemple en réalisant des *playtests*[1].

D'autre part, avec le développement du principe de fortin, nous pourrions notamment constater si notre proposition peut s'adapter à divers genres de jeux avec des concepts similaires de design.

Nous avons observé une certaine diversité apparente avec l'exploration de l'espace de génération. Cependant, cette expérience a été basée principalement sur les illustrations de 150 niveaux obtenus et pourrait donc faire l'objet d'une étude plus approfondie pour mieux évaluer les capacités et les limites de notre approche. Afin d'évaluer plus précisément la variété des expériences de jeu, nous pourrions envisager le concept de spectre d'expressivité, ou *expressive range* en anglais, d'un générateur de niveaux. Ce concept a pour but de comprendre la diversité et l'unicité d'un générateur de niveaux et a été introduit par Gillian Smith et Jim Whitehead [75]. En effet, il s'agit d'une approche permettant de visualiser et de mesurer la variation du contenu généré en fonction d'une ou plusieurs métriques représentatives. Cette méthode d'évaluation, pour les outils de génération procédurale de contenu, permet ainsi de connaître la capacité de l'algorithme à produire un contenu diversifié.

Par ailleurs, nous pourrions comparer également les performances de notre *Genetic-WFC* à d'autres méthodes d'optimisation sous contrainte comme le *FI-2Pop* [125], et observer les résultats générés en se basant sur des paramètres similaires, notamment pour les assets.

Nous sommes également conscients des inconvénients de notre méthode et des lacunes concernant son fonctionnement. Premièrement, le temps de calcul reste conséquent et empêche une utilisation en temps réel. Une optimisation de la programmation reste possible, par exemple, en employant plutôt le C++ au lieu du C# et de se détacher du moteur *Unity* pour les calculs. De plus, la construction de nos niveaux reste limitée par rapport aux modules de *greyblocks* choisis. Avons-nous implémenté suf-

1. Un *playtest* est le processus par lequel un concepteur propose à des individus de tester un nouveau jeu ou un nouveau concept afin d'obtenir des remarques et des impressions de la part des joueurs, pour améliorer l'expérience générale du jeu, détecter les bogues ou encore les défauts de conception [191, Chapitre 9].

fisamment d'assets pour couvrir un maximum de niveaux pertinents ? D'autre part, pour l'algorithme génétique, il serait tout à fait possible de limiter les choix inutiles, par exemple le calcul des bordures dans l'évolution des gènes.

Nous avons également constaté que l'optimisation des individus stagne au bout d'un certain temps, en effet, elle semble atteindre un maxima local. Les paramètres génétiques sont-ils adaptés ou limitent-ils l'algorithme ? Faut-il modifier la mutation ou le croisement ? Il serait ainsi envisageable de s'inspirer des travaux sur les algorithmes génétiques et de leurs performances, notamment sur la question de la diversité des individus dans la population, afin d'éviter de converger vers un optimum local [192].

Afin de répondre à nos divers questionnements évoqués ci-dessus, nous allons détailler, dans la section suivante, des pistes qui ont été parfois amorcées et qui restent à explorer, dont par exemple, diverses possibilités complémentaires pour notre méthode et une expérience utilisateur envisageable.

En définitive, nous pensons, malgré tout, qu'en prenant en compte les fonctionnalités et les inconvénients au point actuel, notre méthode présente une utilité pratique. En effet, elle pourrait servir à générer des niveaux dans le cadre d'un outil de design pour les développeurs grâce aux nombreuses fonctionnalités et réglages déjà introduits. Nous nous positionnons ainsi dans le cadre d'une méthode de génération dite « hors ligne » et d'un outil *mixed-initiative*. Par exemple, un *level designer* peut tout à fait créer entièrement un niveau ou compléter un niveau déjà existant grâce au WFC. Il est possible de spécifier des zones et de poser des assets initiaux pour créer un niveau manuellement et laisser l'algorithme trouver la solution optimale. Avec le WFC et l'extraction automatique de contraintes d'adjacence, il n'est pas nécessaire de spécifier les règles à la main, facilitant ainsi le développement en général. Avec un minimum de règles pour le placement des assets, à savoir par exemple, les contraintes d'adjacence ou les limites du nombre de modules à poser, il est possible d'obtenir des niveaux corrects ne présentant pas d'erreurs.

D'un autre côté, il serait également envisageable d'employer notre *Genetic-WFC* conjointement avec un algorithme de *machine learning* dont le temps d'apprentissage est certes important, mais qui permet, toutefois, par la suite de générer des résultats plus rapidement. Notre méthode pourrait produire une multitude de niveaux en amont, créant ainsi une base de données d'exemples, pour entraîner, notamment, un réseau neuronal.

7.3 Pistes amorcées et pistes à explorer

Nous allons à présent proposer des suggestions de développement par rapport aux pistes et questionnements évoqués précédemment pour approfondir notre méthode et élargir son champ d'action. Au cours de notre recherche, nous avions déjà envisagé et amorcé quelques idées d'avancement. Le fil conducteur du développement était d'atteindre l'expérience utilisateur en proposant des campements jouables afin de valider nos heuristiques d'évaluation.

7.3.1 WFC

Concernant tout d'abord l'optimisation du temps de calcul nécessaire au WFC, la solution serait de fractionner la taille du niveau souhaité et de réaliser plusieurs exécutions de l'algorithme avec des tailles réduites, voir exemple Figure 7.1 page suivante. En effet, le coût de génération pourrait être réduit en optant pour une approche hiérarchique. Le WFC pourrait prendre en compte des contraintes d'adjacence aux bords d'un niveau, ce qui permettrait de générer successivement plusieurs sous-niveaux qui se connecteraient l'un à l'autre. Dans cette perspective, en observant les résultats du Tableau 6.1 page 136 de l'expérience 1, produire un niveau de 30 sur 30 avec 12 assets nécessiterait seulement 48 ms, c'est-à-dire quatre fois la génération d'un niveau de 15x15 sur 12 ms, au lieu des 191 ms obtenus sans fractionnement. Optimiser ainsi de manière significative le temps de génération permettrait également d'enrichir le contenu avec davantage de modules et d'agrandir éventuellement la surface du terrain. Cependant, une telle application pourrait amener des questionnements supplémentaires, concernant notamment les mininum/maximum à respecter pour les assets, ou encore les connexions entre les sous-niveaux vis-à-vis des contraintes d'adjacence.

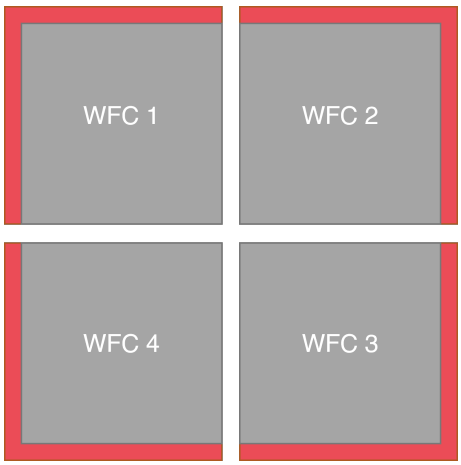


FIGURE 7.1 – Exemple de découpage d'un niveau en plusieurs exécutions du WFC. La bordure globale du niveau est affichée en rouge.

Il serait également possible de diminuer les coûts de calcul liés aux 4 rotations d'un module, par exemple s'il est symétrique ou encore si la rotation n'est pas pertinente.

Nous pourrions aussi envisager l'inclusion de nouvelles contraintes non spatiales, en plus des fréquences et du minimum/maximum pour l'apparition d'un module, par exemple avec l'ajout d'une règle basée sur une distance entre plusieurs assets [151].

7.3.2 Simulation des agents : joueur et ennemis

Afin d'intégrer des ennemis et des objets dans le niveau de type campement, nous avions commencé à étendre les capacités de notre simulation d'agents autonomes, avec :

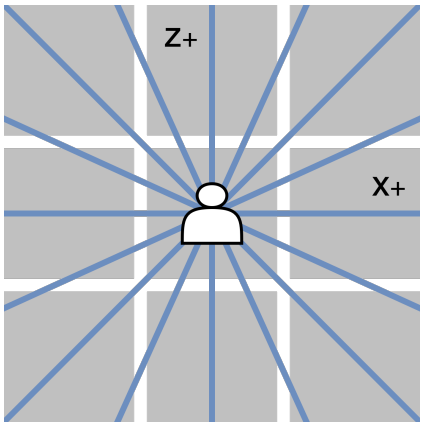
- Inclusion d'ennemis statiques avec un comportement fixe, à savoir tirer sur le joueur s'il est visible, voir Figure 7.3 page 181 ;
- Rajout d'un cône de vision sur les agents simulés, pour détecter, en outre, les autres personnages ;
- Machine à états finis, ou *finite state machine* en anglais, pour l'agent joueur, avec un état pour le combat, pour la recherche d'objectif et pour la découverte du niveau ;
- Ajout d'une case objectif à atteindre pour le joueur, synonyme de victoire ;
- Navigation sur 8 directions pour l'agent joueur, avec la possibilité de sauter, voir Figure 7.2b page 180 ;

- Changement du calcul des formules pour les trois heuristiques, à savoir la nouveauté, la sécurité et la complexité, voir Figure 7.2a page suivante et Figure 7.2c page suivante ;
- Prise en compte des dynamiques de jeu pour l'évaluation du niveau, simplifiées en événements survenus aux agents, par exemple tirer, tuer, être détecté, voir Figure 7.4 page 181.

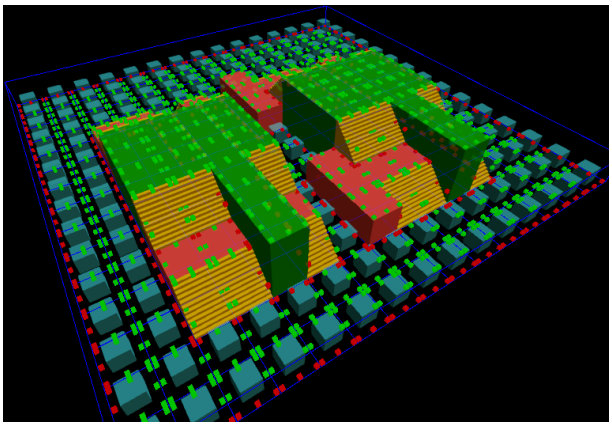
Cependant, nous nous sommes rendus compte que ces ajouts n'étaient pas suffisants pour proposer des niveaux à de véritables joueurs. En effet, il reste de nombreuses fonctionnalités à développer, par exemple :

- Ajout d'états de patrouilles et de poursuites pour les agents ennemis ;
- Inclusion d'autres objectifs réalisables par l'agent joueur, par exemple éliminer tous les ennemis, récupérer certains objets ;
- Développement d'objets utilisables par les agents, par exemple les trousses de secours, les munitions ;
- Permettre aux agents simulés de comprendre la mécanique de couverture ;
- Établir davantage d'heuristiques en liaison avec les événements survenus, par exemple un moyen de mesurer la dangerosité d'un niveau par le nombre de confrontations et de tirs effectués ou subis.

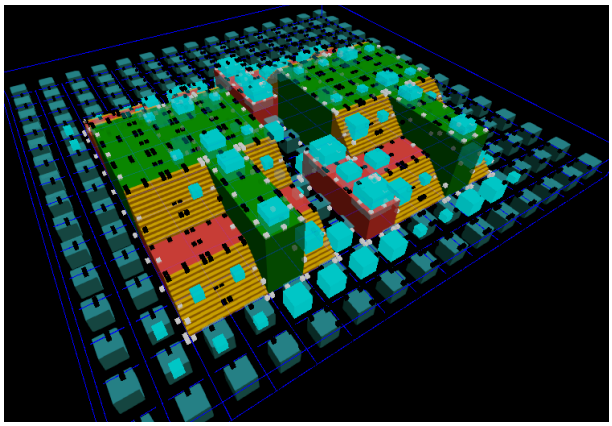
Toutes ces pistes soulèvent également des interrogations concernant la notation du niveau, notamment le parcours qui détermine les actions réalisées. Comment évaluer les possibilités d'un niveau et obtenir une note représentative, avec un seul trajet ? Est-il plutôt préférable de réaliser une multitude de chemins possibles à l'aide de nombreuses simulations sur un même niveau ? Résoudre ces divers questionnements permettrait d'atteindre une simulation plus complète avec du gameplay de type campement afin de réaliser l'expérience utilisateur.



(a) Amélioration du calcul de la visibilité et de la complexité avec de nombreux rayons supplémentaires en diagonale pour affiner le résultat.



(b) Amélioration de la navigation de l'agent avec l'intégration des diagonales pour 8 directions possibles de déplacement.



(c) La valeur de complexité d'une case se base maintenant sur les angles des assets visibles.

FIGURE 7.2 – Exemples d'améliorations réalisées pour la navigation de l'agent joueur et le calcul des heuristiques.

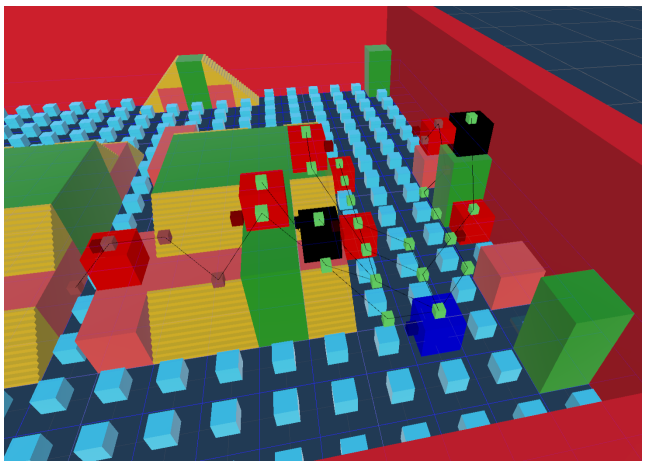


FIGURE 7.3 – Exemple d'une simulation de combat, avec les ennemis en rouge, le joueur en bleu et les personnages décédés en noir. Le joueur perçoit les adversaires selon son cône de vision.

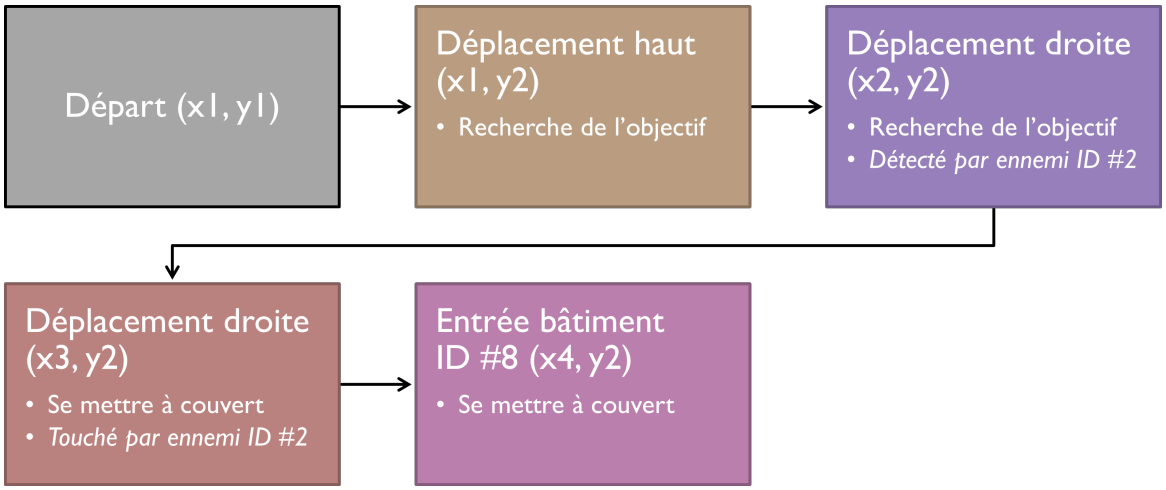


FIGURE 7.4 – Exemple de représentation potentielle de la chronologie des événements survenus à l'agent joueur.

Une idée supplémentaire a été explorée lors des travaux de la thèse. Il s'agissait de simuler de manière encore plus abstraite le parcours du joueur, pour éviter que la simulation plus complexe de plusieurs agents ait un impact trop important sur le temps de calcul. Nous avons cherché à retranscrire la navigation du niveau sous la forme d'un arbre de cases accessibles selon le point de départ et de visualiser des chemins potentiels, voir Figure 7.5 page suivante. Il aurait été possible de calculer une simulation des cases traversées pour tous les chemins accessibles depuis le point de départ, en simplifiant les événements survenus sur les cases parcourues. L'idée aurait été d'attribuer un événement

par case, par exemple un affrontement avec un ennemi ou une récupération de munitions. Nous aurions ainsi pu obtenir une moyenne globale pour le niveau entier. Cependant, la découverte de chemins dans un graphe soulève également des interrogations, notamment sur la complexité algorithmique.

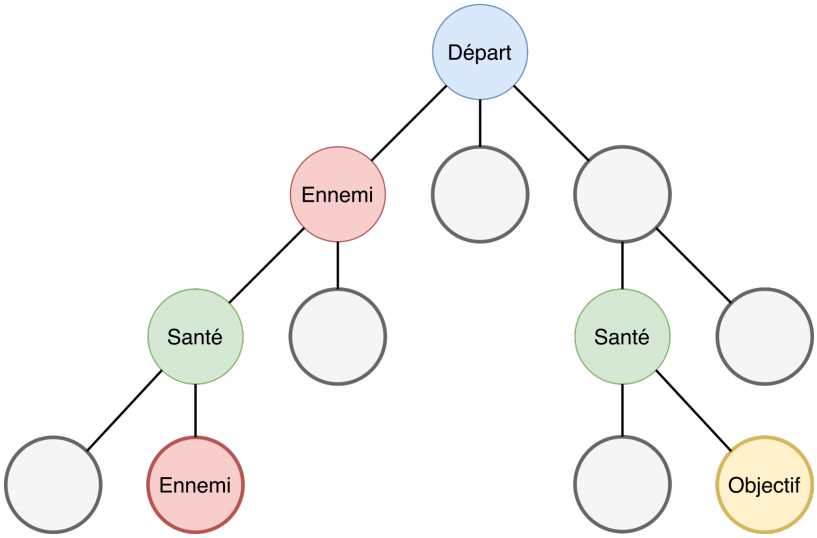


FIGURE 7.5 – Retranscription du niveau sous la forme d'un arbre. Les feuilles représentent les cases accessibles depuis le point de départ et les sommets indiquent un chemin possible pour y accéder. Les couleurs des cases représentent : en gris les emplacements vides, en bleu le point de départ, en rouge les ennemis, en vert la santé, en jaune un objectif.

7.3.3 Éléments du niveau

Afin de proposer une véritable structure de campement, nous avions l'intention de rajouter des *greyblocks* de type bâtiment, par exemple des murs, des portes et des fenêtres, et des *greyblocks* de type gameplay, à savoir des emplacements pour des trousses de secours, des munitions, des objectifs, ou encore des positions pour des personnages ennemis. Nous pourrions également nous inspirer des ingrédients de gameplay mentionnés dans notre formalisation des fortins, voir Annexe A page 213.

Dans ce cas de figure, nous avions également projeté de produire les niveaux avec une étape supplémentaire, voir Figure 7.6 page suivante. En effet, il s'agissait tout d'abord de générer la structure du niveau avec nos trois heuristiques habituelles, puis, dans une couche intermédiaire, de rajouter les éléments de gameplay et de simuler les comportements appropriés pour les agents, puis, de terminer avec le remplacement des *greyblocks* par des assets visuels.

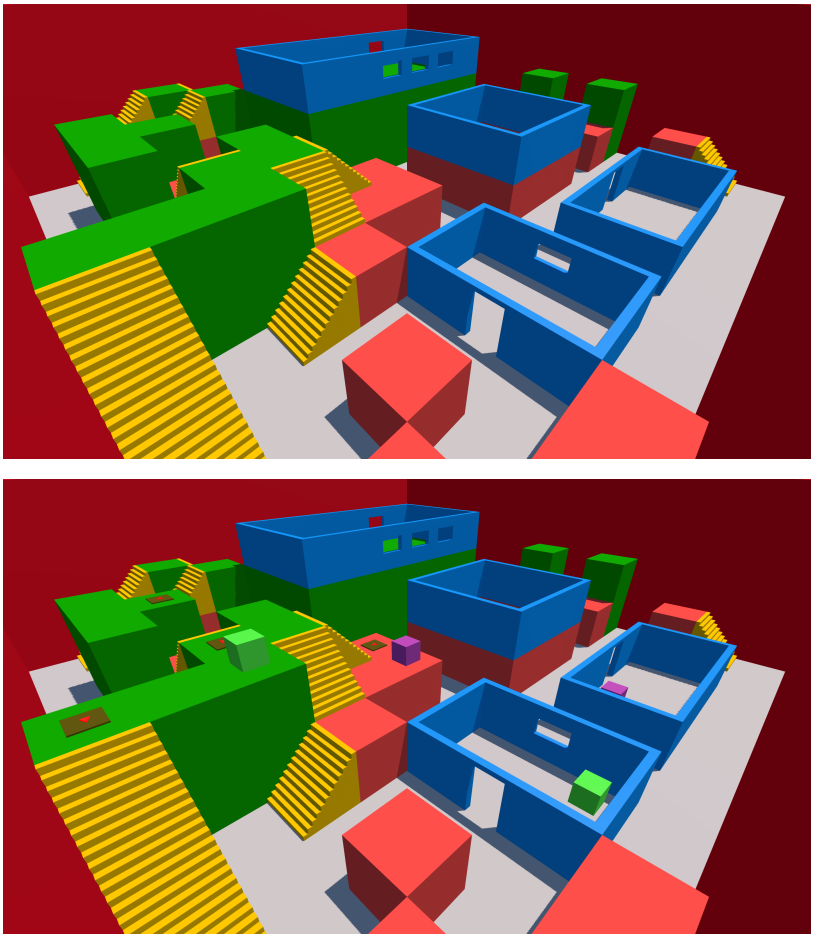


FIGURE 7.6 – Suggestion de représentation de l'implémentation de *greyblocks* supplémentaires. En haut, génération d'un niveau avec des bâtiments. En bas, utilisation du résultat précédent avec le rajout des éléments de gameplay.

7.4 Expérience utilisateur envisagée

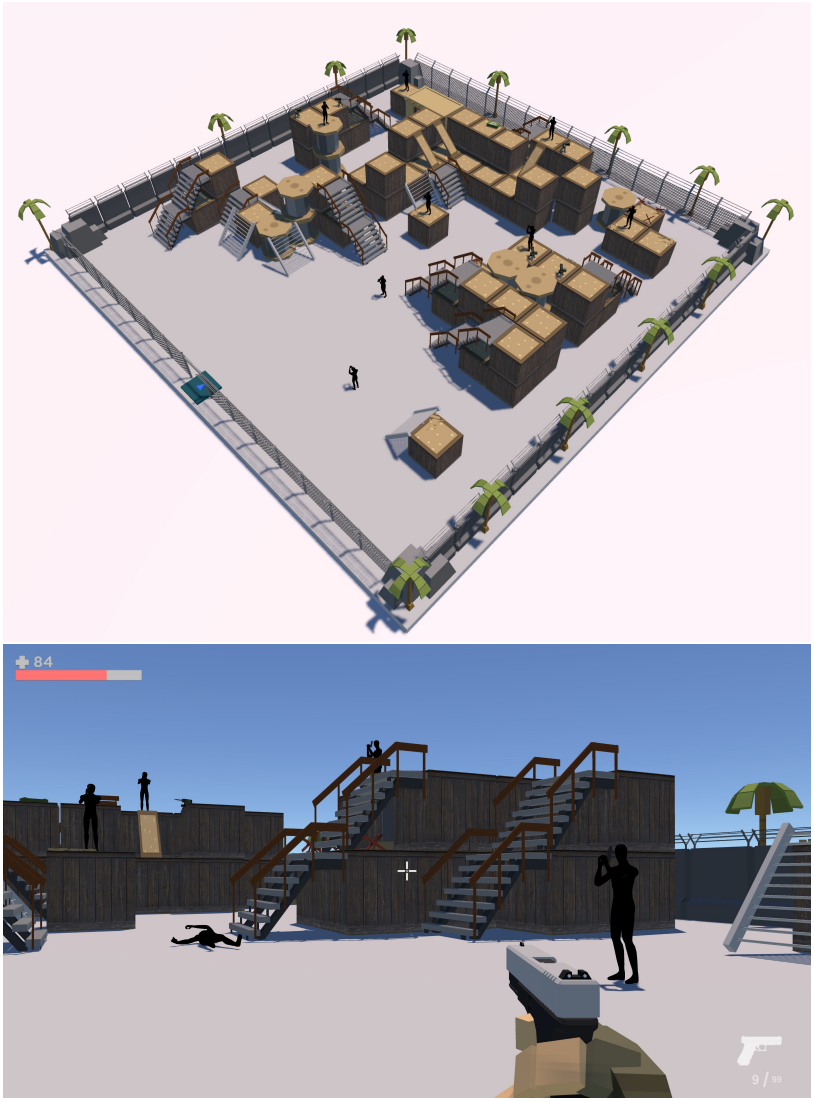


FIGURE 7.7 – Exemple d'un niveau pouvant être proposé lors de l'expérience utilisateur.

7.4.1 Objectif

Dans le but d'approfondir l'utilité de notre méthode de génération, nous pourrions envisager comme suite logique, la mise en place d'une expérience utilisateur, avec la création de campements en niveau ouvert non-linéaire, avec du gameplay en vue FPS. Cette expérimentation représenterait le prolongement de nos travaux avec la validation de la pertinence des heuristiques d'évaluation. Le ressenti de véritables joueurs permettrait, en outre, de vérifier si la diversité des expériences de jeu que nous

proposons éveille suffisamment l'intérêt.

Afin de présenter aux joueurs des niveaux jouables, il conviendrait de reprendre les améliorations présentées précédemment avec les différentes pistes concernant les agents autonomes, c'est-à-dire le joueur et les ennemis, et leurs simulations, ainsi que l'inclusion d'éléments de gameplay supplémentaires dans la génération. La création de différents niveaux avec un prototype fonctionnel permettrait l'élaboration d'un *playtest* comme expérience utilisateur.

Cette étude pourrait répondre à des questions, telles que :

- Obtenons-nous des niveaux intéressants pour les joueurs ?
- Les niveaux sont-ils vraiment ressentis comme différents ?
- La variation de nos heuristiques est-elle perçue ?

Au terme de cette analyse, nous pourrions poursuivre dans cette voie et examiner plus particulièrement l'aspect d'une des problématiques de la génération procédurale de contenu, à savoir, proposer une variabilité intéressante et pertinente qui motive le joueur et stimule sa curiosité.

7.4.2 Protocole

Mettre en place une telle expérimentation demanderait une organisation matérielle, la production du contenu à tester, ainsi que la préparation d'un questionnaire à soumettre à nos joueurs.

Organisation matérielle Il s'agirait avant tout de disposer d'un nombre suffisant de joueurs participant à l'expérience et permettant ainsi d'obtenir assez de résultats. Il serait alors envisageable de faire appel aux locaux et aux élèves de l'établissement Cnam-Enjmin à Angoulême. Cela permettrait également d'avoir des avis pertinents, car les participants étudient justement dans le domaine du jeu vidéo. Il faudrait réserver dans ce cas une salle pour un certain temps, une semaine par exemple, avec le matériel adéquat, à savoir entre autres, un ordinateur, une caméra, un microphone. De plus, l'expérimentation nécessiterait de disposer d'un prototype final à soumettre aux joueurs, avec les différents niveaux, et d'un questionnaire établi par nos soins.

Réflexion nécessaire sur le déroulement Concernant l'exécution des différentes étapes de cette expérience, nous pourrions procéder avec une personne à la fois, en lui expliquant les règles et les objectifs

à atteindre dans le jeu. Il serait concevable de fixer un temps maximal pour l'expérimentation, par exemple 20 min au total. Il serait également judicieux de limiter un temps de jeu par niveau, dans le cas où une personne se retrouverait bloquée ou s'ennuierait. Nous pourrions interroger, à l'aide d'un questionnaire, le participant entre certains niveaux et/ou après l'expérience, sur son ressenti et noter ses commentaires.

Enregistrement durant la partie Un enregistrement de la partie du joueur serait approprié afin de récolter le plus d'informations possibles sur sa façon de jouer. Nous pourrions prendre en compte son parcours et ses actions, son temps de jeu, son nombre d'essais, ses échecs et ses victoires. Il serait également judicieux d'utiliser une caméra afin de sauvegarder les réactions physiques du participant.

L'établissement du contenu à expérimenter Le prototype proposé pourrait être composé d'un certain nombre de scénarios répartis dans plusieurs niveaux. Un scénario à tester pourrait être composé, par exemple, d'une succession de trois niveaux, avec un même objectif, mais avec toutefois des variations d'éléments de gameplay. Les mêmes objectifs devraient être proposés à chaque participant. Un objectif pourrait être, par exemple, d'atteindre un certain point dans le niveau. Chaque scénario pourrait avoir un aspect visuel différent pour être différenciable facilement. Dans chaque scénario, il faudrait également définir un niveau neutre parmi les trois, afin de disposer d'une base de notation. Les deux autres niveaux pourraient proposer des expériences de jeu spécifiques, par exemple avec des situations d'attaques frontales ou d'infiltrations. Nous devrions alors prévoir les paramètres et les heuristiques que nous voudrions valider dans cette expérimentation. Le but recherché par ces scénarios serait, par exemple, de faire ressentir aux joueurs de légères variations pour vérifier la subtilité des paramètres choisis dans notre génération de niveaux.

Au final, il nous faudrait définir clairement les scénarios, les objectifs des niveaux et les valeurs ajustables que nous désirerions tester, mais également détailler les conditions de victoire et d'échec pour le joueur.

7.4.3 Analyse des résultats

Grâce aux différentes données recueillies durant l'expérience utilisateur, nous disposerions alors de résultats conséquents. Nous pourrions ensuite les regrouper et les analyser afin de comprendre les

connexions entre les heuristiques associées aux niveaux et les réponses du questionnaire. Est-ce que le joueur a ressenti le gameplay favorisé par notre génération ? A-t-il identifié des différences ? Il serait également possible de confronter le parcours du joueur dans le niveau au parcours de notre agent initial simulé. Les conclusions que nous pourrions en tirer, au final, nous permettraient de valider la pertinence ou non de nos heuristiques d'évaluation.

7.5 Conclusion

Ce chapitre a présenté une discussion sur les divers travaux réalisés dans cette thèse. Nous nous sommes efforcés d'apporter une analyse de ces résultats face à notre problématique et à nos objectifs initiaux.

Nous avons pu ainsi exposer nos apports dans le domaine de la recherche avec notre nouvelle méthode fonctionnelle, à savoir :

- l'intégration et le pilotage du WFC dans un algorithme génétique, en tant qu'opérateur de réparation dans une optimisation sous contrainte,
- et nos heuristiques de perception conçues pour être pertinentes du point de vue du joueur permettant d'aboutir à une diversité d'expériences de jeu.

Au final, nous pensons avoir réalisé un outil intéressant de génération procédurale de niveau, axé sur la question de la diversité tout en conservant une certaine qualité structurelle, afin de générer la plus grande variété de niveaux jouables. Pour ce faire, nous avons développé notre algorithme *Genetic-WFC*, qui se compose ainsi :

- d'une approche *Search-Based* avec une évaluation par simulation, pour atteindre une diversité d'expériences de jeu,
- et d'une méthode constructive, à savoir le WFC, dans le but d'obtenir des niveaux ayant une certaine qualité structurelle avec l'apport d'un minimum de contraintes.

En ce qui concerne la simulation, elle a été implémentée avec le parcours d'un joueur synthétique et de différentes heuristiques de perception, dans le but de guider la recherche vers des expériences de jeu spécifiques.

Cependant, nous avons soulevé quelques points pouvant être approfondis, à propos de certains inconvénients d'utilisation et de certaines suggestions d'améliorations importantes à prendre en considération. En effet, nous sommes conscients du temps de calcul conséquent que nécessite l'exécution de notre méthode, et des interrogations qui restent ouvertes concernant l'usage de nos paramètres et des éléments employés dans la conception du niveau. Nous tenons également à souligner que nous n'avons pas eu l'occasion de démontrer l'impact réel sur le ressenti d'un véritable utilisateur de nos heuristiques d'évaluation propres à l'agent. Il serait tout indiqué, dans ce cas, de poursuivre avec la mise en place d'une expérience utilisateur.

En définitive, dans ce chapitre, nous avons présenté différentes pistes qui avaient été amorcées pendant notre thèse, mais également celles envisageables pour améliorer notre approche dans le but de nous rapprocher d'une véritable expérience utilisateur et ainsi de mieux mesurer la diversité obtenue, plus particulièrement dans le cadre d'un campement. Pour ce faire, nous avons, entre autres, réfléchi à l'avancement possible de la simulation avec l'inclusion d'ennemis et l'amélioration de notre agent joueur. Nous avons également envisagé d'autres pistes concernant le rajout de contenu dans le niveau, c'est-à-dire des assets liés au gameplay, et des heuristiques complémentaires pour l'évaluation des actions réalisées dans le niveau selon les agents présents. Finalement, nous avons présenté l'ébauche d'une expérience utilisateur potentiellement réalisable, afin d'évaluer notre méthode et ses heuristiques de façon plus pertinente, dans une infrastructure jouable de campement. Cette expérience représenterait, comme déjà mentionné, la suite logique à nos travaux.

Chapitre 8

Conclusion

Nos travaux de recherche nous ont permis d'étudier le domaine de la génération procédurale de contenu dans le jeu vidéo. Nous nous sommes intéressés plus spécifiquement aux questionnements liés à la diversité et à la qualité du contenu généré. Il s'agissait, pour nous, d'un sujet important à approfondir et a donc représenté le fil conducteur de cette thèse.

Grâce à une revue de littérature, l'étude des concepts, des méthodes et des travaux déjà existants, nous a permis de découvrir les recherches pouvant nous servir à appréhender notre sujet. Nous avons distingué les algorithmes constructifs, qui produisent le contenu en une seule fois, et les algorithmes essai-erreur et itératifs, plutôt basés sur l'évaluation jusqu'à l'obtention d'un résultat satisfaisant. Nous avons également exploré la programmation par contraintes et les grammaires génératives. Finalement, nous avons constaté que la génération procédurale de contenu est un domaine de recherche particulièrement vaste. Les problématiques liées à son utilisation demeurent toujours au cœur de nouvelles études.

Dans le cadre de cette thèse, nous avons décidé d'aborder notre sujet à l'aide de la problématique suivante : « Comment générer procéduralement une forte diversité d'expériences de jeu, tout en conservant une certaine qualité structurelle ? ». En d'autres termes, nous avons cherché à atteindre le plus grand espace possible de diversité d'expériences de jeu tout en limitant au maximum les erreurs de *level design*. De ce fait, nous avons centré notre étude sur la conception des niveaux de jeu et le placement d'objets. Nous nous sommes focalisés, plus particulièrement, sur la génération de niveaux 3D ouverts non-linéaires de type jeu de tir à la première personne, avec une infrastructure de campement dont le contenu est positionné sur une grille 2D. Afin de répondre à cette problématique, nous avons orienté notre recherche vers la proposition d'une méthode innovante de génération de niveaux, selon des formalismes existants, mentionnés dans notre état de l'art.

Nos travaux ont abouti à la proposition d'une nouvelle méthode, appelée *Genetic-WFC*, dans le but d'offrir une diversité d'expériences de jeu avec des niveaux possédant une certaine qualité structurelle. Il s'agit d'un pipeline de génération procédurale qui permet de générer des niveaux ciblant des expériences de jeu spécifiques. Notre méthode combine une approche *Search-Based*, composée d'un algorithme génétique et d'une évaluation par simulation d'un joueur synthétique, avec la méthode constructive, le *Wave Function Collapse*, un algorithme de propagation de contraintes locales d'adjacence. Le WFC s'intègre ainsi comme opérateur de réparation dans une optimisation sous contrainte. L'emploi du *Search-Based* avec une évaluation par simulation nous permet d'atteindre une diversité d'expériences

de jeu. Nous avons utilisé le WFC, quant à lui, dans le but d'obtenir des niveaux ayant une certaine qualité structurelle avec l'apport d'un minimum de contraintes.

Dans notre méthode développée, le WFC extrait les contraintes d'adjacence à partir d'exemples de niveau, et nous permet d'effectuer la recherche génétique sur des résultats ne présentant pas d'erreurs de placement d'objets. Il agit, en effet, en tant qu'opérateur de réparation pour les individus de la population de l'algorithme génétique. Le pilotage du WFC, par l'algorithme de recherche, est rendu possible en influençant la probabilité de sélection de ses éléments, grâce à des zones de probabilités manipulées par l'algorithme génétique. Nous employons une solution de réencodage des niveaux qui nous permet d'améliorer le processus d'optimisation de notre algorithme évolutionnaire. Nous utilisons également un joueur synthétique pour évaluer l'expérience de jeu à l'aide de trois heuristiques de perception, à savoir, la nouveauté, la sécurité et la complexité, lors d'une simulation d'un parcours. Afin de guider la méthode itérative vers une certaine expérience de jeu, nous appliquons des coefficients de pondération à ces heuristiques. Nous proposons également un principe de multicouche avec l'utilisation d'une première étape employant uniquement des catégories d'assets dans la méthode itérative, pour limiter l'explosion combinatoire du WFC. Puis, nous remplaçons simplement le résultat final obtenu par l'algorithme génétique par des assets plus visuels.

Afin de démontrer la faisabilité et l'efficacité de notre méthode, nous avons réalisé différentes expériences. Nous avons, tout d'abord, réalisé une première expérimentation basée sur le temps de calcul du WFC. Le résultat obtenu nous a confortés dans l'idée d'utiliser un nombre limité de *greyblocks*. Une deuxième démonstration s'est plutôt concentrée sur la comparaison directe de notre approche avec d'autres méthodes similaires. Les résultats obtenus ont permis de valider le fonctionnement de notre *Genetic-WFC*, à savoir le pilotage et l'inclusion du WFC dans un algorithme génétique en tant qu'opérateur de réparation, grâce aux zones de probabilités. De plus, le réencodage des gènes s'est révélé être efficace et nécessaire. Nous avons également observé que le WFC était un choix judicieux pour produire des niveaux avec une certaine qualité structurelle. Toutefois, son utilisation a impacté le temps de calcul. En tenant compte des résultats, nous avons constaté que notre méthode était plus performante qu'une approche classique de pénalisation, pour proposer des niveaux jouables de qualité structurelle. Nous avons aussi observé que le parcours de l'agent autonome, dans notre simulation, a permis d'évaluer les niveaux. Les préférences du joueur synthétique, liées aux heuristiques de perception de nouveauté, de sécurité et de complexité, se sont bien reflétées dans la composition des assets du

résultat. La troisième et dernière expérimentation, axée sur l'exploration de l'espace de génération, nous a donné l'occasion d'observer la diversité des expériences de jeu que peut proposer notre méthode. Les résultats ont indiqué que les variations des coefficients des heuristiques produisent bien une certaine diversité d'expérience de jeu, en influençant les divers aspects du niveau, c'est-à-dire la disposition, le placement et le nombre d'assets.

Les résultats de nos expérimentations nous paraissent concluants. Néanmoins, quelques points méritent d'être approfondis lors de recherches ultérieures. En effet, nous sommes conscients du temps de calcul conséquent que nécessite l'exécution de notre méthode. Certaines interrogations restent également encore ouvertes concernant l'usage de nos paramètres et des éléments employés dans la conception du niveau. De plus, intégrer des éléments de gameplay et compléter la simulation avec des agents ennemis dans notre méthode de génération, permettrait de nous rapprocher d'une expérience utilisateur plus représentative, et ainsi de mieux mesurer la diversité obtenue, plus particulièrement dans le cadre d'un campement. Nous tenons également à souligner que nous n'avons pas eu l'occasion de démontrer l'impact réel sur le ressenti d'un véritable utilisateur de nos heuristiques d'évaluation propres à l'agent. Il serait tout indiqué, dans ce cas, de poursuivre avec la mise en place d'une expérience utilisateur, qui représenterait alors la suite logique de nos travaux.

Pour conclure, les travaux que nous avons menés tout au long de cette thèse, nous ont permis d'appréhender certains aspects particuliers de la PCG. Le développement de notre étude a conduit à la création d'un générateur innovant, à savoir une approche proposant une diversité d'expériences de jeu, tout en conservant une certaine qualité structurelle avec l'apport d'un minimum de contraintes. Notre *Genetic-WFC* dans sa composition distinctive, à savoir l'intégration d'une méthode constructive, le WFC, dans une optimisation par approche *Search-Based*, propose des résultats prometteurs aux questionnements liés à la diversité et à la qualité du contenu généré. Pour autant, notre contribution peut être prolongée par l'exploration de nouvelles directions et susciter des études complémentaires.

Bibliographie

[1] S. Garces, “Steam in graphs in 2019,” 2020, [En ligne]. Disponible : https://www.gamedeveloper.com/business/steam-in-graphs-in-2019/, consulté le 10/09/2022.

[2] W. Commons, “Screenshot of rogue (1980),” 2008, [En ligne]. Disponible : https://commons.wikimedia.org/wiki/File:Rogue_Screen_Shot_CAR.PNG, consulté le 28/09/2022.

[3] M. Toy et G. Wichman, “Rogue,” 1980.

[4] G. N. Yannakakis et J. Togelius, *Artificial Intelligence and Games*, 1er éd. Springer Publishing Company, Incorporated, 2018.

[5] HelloGames, “No man’s sky,” 2016.

[6] T. Hely, “How to use bsp trees to generate game maps,” 2013, [En ligne]. Disponible : https://gamedevelopment.tutsplus.com/tutorials/how-to-use-bsp-trees-to-generate-game-maps--gamedev-12268, consulté le 25/10/2019.

[7] W. Commons, “Coloured voronoi 2d,” 2006, [En ligne]. Disponible : https://commons.wikimedia.org/wiki/File:Coloured_Voronoi_2D.png, consulté le 25/10/2019.

[8] A. Patel, “Polygonal map generation for games,” 2010, [En ligne]. Disponible : http://www-cs-students.stanford.edu/~amitp/game-programming/polygon-map-generation/, consulté le 03/10/2019.

[9] M. Cook, “Generate random cave levels using cellular automata,” 2013, [En ligne]. Disponible : https://gamedevelopment.tutsplus.com/tutorials/generate-random-cave-levels-using-cellular-automata--gamedev-9664, consulté le 25/10/2019.

[10] W. Commons, “Random walk in two dimensions,” 2013, [En ligne]. Disponible : https://commons.wikimedia.org/wiki/File:Random_walk_2500.svg, consulté le 03/10/2019.

[11] D. Kazemi, “Spelunky generator lessons,” [En ligne]. Disponible : http://tinysubversions.com/spelunkyGen/, consulté le 10/09/2019.

[12] J. Togelius, G. N. Yannakakis, K. O. Stanley et C. Browne, “Search-based procedural content generation : A taxonomy and survey,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, n°. 3, p. 172–186, 2011.

[13] J. Dieskau, “Multi-objective procedural level generation for general video game playing,” Mémoire de maîtrise, 2016.

[14] J. Togelius, M. Preuss, N. Beume, S. Wessing, J. Hagelbäck et G. N. Yannakakis, “Multiobjective exploration of the starcraft map space,” dans *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*. IEEE, 2010, p. 265–272.

[15] M. Gumin, “Wave function collapse,” 2016, [En ligne]. Disponible : https://github.com/mxgmn/WaveFunctionCollapse, consulté le 03/10/2019.

[16] C. Sun, “Implementation of wave function collapse algorithm in houdini for 3d content generation,” 2020, [En ligne]. Disponible : https://chloesun.medium.com/implementation-of-wave-function-collapse-algorithm-in-houdini-for-3d-content-generation-76f8eec573b1, consulté le 21/06/2022.

[17] O. Stålberg, “Townscaper,” 2020.

[18] Q. E. Morris, “Modifying wave function collapse for more complex use in game generation and design,” Thèse de doctorat, 2021.

[19] T. Møller et J. Billeskov, “Expanding wave function collapse with growing grids for procedural content generation.” Thèse de doctorat, 05 2019.

[20] T. Veldhuis, “Graph-defined dungeons : Exploring constraint solving for the generation of action-adventure levels,” Mémoire de maîtrise, Utrecht University, 2018.

[21] N. Shaker, J. Togelius et M. J. Nelson, *Procedural Content Generation in Games : A Textbook and an Overview of Current Research.* Springer, 2016, website : http://pcgbook.com/. Last visited 30 Jun. 2021.

[22] J. Dormans et S. Bakkes, "Generating missions and spaces for adaptable play experiences," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, n°. 3, p. 216–228, 2011.

[23] J. Dormans, "Epc2016 - joris dormans - cyclic dungeon generation," 2016, [En ligne]. Disponible : https://www.youtube.com/watch?v=mA6PacEZX9M, consulté le 16/01/2020.

[24] Ubisoft, "Tom clancy's ghost recon : Wildlands," 2017.

[25] Whitebox, "An analysis of ghost recon wildlands," 2018, [En ligne]. Disponible : https://theconstruct.wixsite.com/whitebox/single-post/2018/02/14/An-Analysis-of-Ghost-Recon-Wildlands, consulté le 27/06/2022.

[26] Ubisoft, "Tom clancy's ghost recon : Breakpoint," 2019.

[27] S. Aleem, L. F. Capretz et F. Ahmed, "Critical success factors to improve the game development process from a developer's perspective," *Journal of Computer Science and Technology*, vol. 31, n°. 5, p. 925–950, 2016.

[28] B. Elad, "25+ steam statistics 2022 users, most played games, market share and demographics," 2022, [En ligne]. Disponible : https://www.enterpriseappstoday.com/stats/steam-statistics.html, consulté le 28/09/2022.

[29] Y. Liang, "Analysis of the video gaming industry," dans *2022 2nd International Conference on Enterprise Management and Economic Development (ICEMED 2022).* Atlantis Press, 2022, p. 1146–1150.

[30] U. Wilhelmsson, W. Wang, R. Zhang et M. Toftedahl, "Shift from game-as-a-product to game-as-a-service research trends," p. 1–3, 2022.

[31] Ubisoft, "Tom clancy's rainbow six : Siege," 2015.

[32] W. W. Maxis, "Spore," 2008.

[33] J. Togelius, E. Kastbjerg, D. Schedl et G. N. Yannakakis, "What is procedural content generation ? : Mario on the borderline," dans *Proceedings of the 2nd international workshop on procedural content generation in games.* ACM, 2011, p. 3.

[34] R. Rouse et S. Ogden, *Game Design : Theory and Practice, Second Edition*, ser. Wordware game developer's library. Jones & Bartlett Learning, 2004.

[35] K. S. Tekinbas et E. Zimmerman, *Rules of play : Game design fundamentals.* MIT press, 2003.

[36] Mojang, "Minecraft," 2009.

[37] M. Albinet, *Concevoir un jeu vidéo : les méthodes et les outils des professionnels expliqués à tous !* Fyp éditions, 2015.

[38] G. Smith, "An analog history of procedural content generation." dans *FDG*, 2015.

[39] A. Lindenmayer, "Mathematical models for cellular interactions in development i. filaments with one-sided inputs," *Journal of theoretical biology*, vol. 18, n°. 3, p. 280–299, 1968.

[40] K. Perlin, "An image synthesizer," *ACM Siggraph Computer Graphics*, vol. 19, n°. 3, p. 287–296, 1985.

[41] D. Braben et I. Bell, "Elite," 1984.

[42] N. Brewer, "Going rogue : A brief history of the computerized dungeon crawl," 2016, [En ligne]. Disponible : https://insight.ieeeusa.org/articles/going-rogue-a-brief-history-of-the-computerized-dungeon-crawl/, consulté le 24/10/2022.

[43] V. Cerny et F. Dechterenko, "Rogue-like games as a playground for artificial intelligence–evolutionary approach," dans *International Conference on Entertainment Computing.* Springer, 2015, p. 261–271.

[44] G. N. Yannakakis, A. Liapis et C. Alexopoulos, "Mixed-initiative co-creativity." dans *FDG*, 2014.

[45] G. Smith, J. Whitehead et M. Mateas, "Tanagra : A mixed-initiative level design tool," dans *Proceedings of the Fifth International Conference on the Foundations of Digital Games.* ACM, 2010, p. 209–216.

[46] E. Olsson et E. Grevillius, “Mixed-initiative quest generation,” Thèse de doctorat, 2020.

[47] G. Smith, M. Treanor, J. Whitehead et M. Mateas, “Rhythm-based level generation for 2d platformers,” dans *Proceedings of the 4th International Conference on Foundations of Digital Games*. ACM, 2009, p. 175–182.

[48] GearboxSoftware, “Borderlands,” 2009.

[49] I-NovaeStudios, “Infinity,” 2004.

[50] Valve, “Left 4 dead,” 2008.

[51] M. Booth, “The ai systems of left 4 dead,” *Artificial Intelligence and Interactive Digital Entertainment Conference at Stanford, 2009*, 2009. [En ligne]. Disponible : https://ci.nii.ac.jp/naid/10027130028/en/

[52] B. M. Guillaume Werle, “[gdc] ghost recon wildlands terrain tools and technology,” 2017, [En ligne]. Disponible : https://www.youtube.com/watch?v=kzthHcbG9IM, consulté le 03/10/2019.

[53] G. Smith, “Understanding procedural content generation : a design-centric analysis of the role of pcg in games,” dans *Proceedings of the 32nd annual ACM conference on Human factors in computing systems*. ACM, 2014, p. 917–926.

[54] Farbrausch, “.kkrieger,” 2004.

[55] T. Tutenel, R. Bidarra, R. M. Smelik et K. J. D. Kraker, “The role of semantics in games and simulations,” *Computers in Entertainment (CIE)*, vol. 6, n°. 4, p. 57, 2008.

[56] Bill, “Chasing an industry : The economics of videogames turned hollywood,” 2013, [En ligne]. Disponible : https://monstervine.com/2013/06/chasing-an-industry-the-economics-of-videogames-turned-hollywood/, consulté le 15/01/2020.

[57] E. J. Hastings, R. K. Guha et K. O. Stanley, “Automatic content generation in the galactic arms race video game,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 1, n°. 4, p. 245–263, 2009.

[58] J. Togelius, A. J. Champandard, P. L. Lanzi, M. Mateas, A. Paiva, M. Preuss et K. O. Stanley, "Procedural content generation : Goals, challenges and actionable steps," dans *Dagstuhl Follow-Ups*, vol. 6. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

[59] D. Santiago, "[gdc] procedurally crafting manhattan for marvel's spider-man," 2019, [En ligne]. Disponible : https://www.youtube.com/watch?v=4aw9uyj9MAE, consulté le 03/10/2019.

[60] I. I. Interactive Data Visualization, "Speedtree," 2002.

[61] NaturalMotion, "Euphoria," 2008.

[62] E. R. C. Zurich, "Cityengine," 2008.

[63] D. Isla, "[gdc] forging the river in the flame in the flood," 2016, [En ligne]. Disponible : https://www.youtube.com/watch?v=6N56YpHCHBM, consulté le 03/10/2019.

[64] Tarn et Z. Adams, "Dwarf fortress," 2006.

[65] R. Moss, "7 uses of procedural generation that all developers should study," 2016, [En ligne]. Disponible : https://www.gamasutra.com/view/news/262869/7_uses_of_procedural_generation_that_all_developers_should_study.php, consulté le 03/10/2019.

[66] S. Chauvin, G. Levieux, J.-Y. Donnart et S. Natkin, "Making sense of emergent narratives : An architecture supporting player-triggered narrative processes," dans *2015 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2015, p. 91–98.

[67] G. Smith, E. Gan, A. Othenin-Girard et J. Whitehead, "Pcg-based game design : enabling new play experiences through procedural content generation," dans *Proceedings of the 2nd international workshop on procedural content generation in games*. ACM, 2011, p. 7.

[68] R. Lopes et R. Bidarra, "Adaptivity challenges in games and simulations : a survey," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, n°. 2, p. 85–99, 2011.

[69] G. N. Yannakakis et J. Togelius, "Experience-driven procedural content generation," p. 519–525, 2015.

[70] N. Shaker, G. N. Yannakakis et J. Togelius, "Towards automatic personalized content generation for platform games." dans *AIIDE*, 2010.

[71] ——, "Towards player-driven procedural content generation," dans *Proceedings of the 9th conference on Computing Frontiers*. ACM, 2012, p. 237–240.

[72] R. Hunicke, "The case for dynamic difficulty adjustment in games," dans *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*. ACM, 2005, p. 429–433.

[73] M. Zohaib, "Dynamic difficulty adjustment (dda) in computer games : A review," *Advances in Human-Computer Interaction*, vol. 2018, 2018.

[74] W. R. Fernandes et G. Levieux, "delta-logit : Dynamic difficulty adjustment using few data points," dans *Joint International Conference on Entertainment Computing and Serious Games*. Springer, 2019, p. 158–171.

[75] G. Smith et J. Whitehead, "Analyzing the expressive range of a level generator," dans *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. ACM, 2010, p. 4.

[76] B. Horn, S. Dahlskog, N. Shaker, G. Smith et J. Togelius, "A comparative evaluation of procedural level generators in the mario ai framework," dans *Foundations of Digital Games 2014, Ft. Lauderdale, Florida, USA (2014)*. Society for the Advancement of the Science of Digital Games, 2014, p. 1–8.

[77] M. Preuss, A. Liapis et J. Togelius, "Searching for good and diverse game levels," dans *2014 IEEE Conference on Computational Intelligence and Games*. IEEE, 2014, p. 1–8.

[78] A. Liapis, G. N. Yannakakis et J. Togelius, "Constrained novelty search : A study on game content generation," *Evolutionary computation*, vol. 23, n°. 1, p. 101–129, 2015.

[79] A. Summerville, "Expanding expressive range : Evaluation methodologies for procedural content generation," dans *Fourteenth artificial intelligence and interactive digital entertainment conference*, 2018.

[80] D. Gravina, A. Khalifa, A. Liapis, J. Togelius et G. N. Yannakakis, "Procedural content generation through quality diversity," dans *2019 IEEE Conference on Games (CoG)*. IEEE, 2019, p. 1–8.

[81] M. Cook, S. Colton et J. Gow, "The angelina videogame design system—part i," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 9, n°. 2, p. 192–203, 2016.

[82] M. J. Nelson et M. Mateas, "Towards automated game design," dans *Congress of the Italian Association for Artificial Intelligence*. Springer, 2007, p. 626–637.

[83] Nintendo, "Warioware, inc. : Mega mini-jeux," 2003.

[84] A. Zook et M. O. Riedl, "Automatic game design via mechanic generation." dans *AAAI*, 2014, p. 530–537.

[85] T. Schaul, "A video game description language for model-based or interactive learning," dans *2013 IEEE Conference on Computational Inteligence in Games (CIG)*. IEEE, 2013, p. 1–8.

[86] R. Mall, *Fundamentals of software engineering*. PHI Learning Pvt. Ltd., 2018.

[87] K. Hullett et J. Whitehead, "Design patterns in fps levels," dans *proceedings of the Fifth International Conference on the Foundations of Digital Games*. ACM, 2010, p. 78–85.

[88] K. Hullett, "The science of level design : Design patterns and analysis of player behavior in first-person shooter levels," Thèse de doctorat, UC Santa Cruz, 2012.

[89] S. Dahlskog, "Patterns and procedural content generation in digital games : automatic level generation for digital games using game design patterns," Thèse de doctorat, 2016.

[90] M. Hendrikx, S. Meijer, J. Van Der Velden et A. Iosup, "Procedural content generation for games : A survey," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 9, n°. 1, p. 1, 2013.

[91] C. W. Totten, *Level design : Processes and experiences*. CRC Press, 2017.

[92] J. v. Neumann, "Theory of self-reproducing automata," *Edited by Arthur W. Burks*, 1966.

[93] M. Gardner, "Mathematical games the fantastic combinations of john conway's new solitaire game" life," *Scientific American*, vol. 223, p. 120–123, 1970.

[94] L. Johnson, G. N. Yannakakis et J. Togelius, "Cellular automata for real-time generation of infinite cave levels," dans *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. ACM, 2010, p. 10.

[95] R. Van der Linden, R. Lopes et R. Bidarra, "Procedural generation of dungeons," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, n°. 1, p. 78–89, 2014.

[96] D. Ashlock, "Evolvable fashion-based cellular automata for generating cavern systems," dans *2015 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2015, p. 306–313.

[97] E. Garza, "Algorithmic generation of cities using cellular automata in a dynamically generated world," *Computer Science Honors Theses*, p. 7, 2005.

[98] J. Olsen, "Realtime procedural terrain generation," 2004.

[99] T. Bacher, "Procedural level generation algorithms," 2018.

[100] S. Benard, "Building the level design of a procedurally generated metroidvania : a hybrid approach," 2017, [En ligne]. Disponible : https://www.gamasutra.com/blogs/SebastienBENARD/20170329/294642/Building_the_Level_Design_of_a_procedurally_generated_Metroidvania_a_hybrid_approach.php, consulté le 16/01/2020.

[101] D. Yu, "Spelunky," 2008.

[102] M. Twin, "Dead cells," 2017.

[103] AmplitudeStudios, "Dungeon of the endless," 2014.

[104] F. H. Edmund McMillen, "The binding of isaac," 2011.

[105] D. Yu, *Spelunky : Boss Fight Books# 11*. Boss Fight Books, 2016, vol. 11.

[106] P. Mawhorter et M. Mateas, "Procedural level generation using occupancy-regulated extension," dans *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*. IEEE, 2010, p. 351–358.

[107] A. Eiben et J. Smith, "Introduction to evolutionary computing," 2015.

[108] D. Ashlock, *Evolutionary computation for modeling and optimization*. Springer, 2006, vol. 571.

[109] Nintendo, "Super mario bros," 1985.

[110] R. Bartle, "Hearts, clubs, diamonds, spades : Players who suit muds," *Journal of MUD research*, vol. 1, n°. 1, p. 19, 1996.

[111] P. T. Ølsted, B. Ma et S. Risi, "Interactive evolution of levels for a competitive multiplayer fps," dans *2015 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2015, p. 1527–1534.

[112] J. Secretan, N. Beato, D. B. D'Ambrosio, A. Rodriguez, A. Campbell, J. T. Folsom-Kovarik et K. O. Stanley, "Picbreeder : A case study in collaborative evolutionary exploration of design space," *Evolutionary Computation*, vol. 19, n°. 3, p. 373–403, 2011.

[113] S. Risi, J. Lehman, D. B. D'Ambrosio, R. Hall et K. O. Stanley, "Combining search-based procedural content generation and social gaming in the petalz video game," dans *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2012.

[114] A. Liapis, C. Holmgård, G. N. Yannakakis et J. Togelius, "Procedural personas as critics for dungeon generation," dans *European Conference on the Applications of Evolutionary Computation*. Springer, 2015, p. 331–343.

[115] C. Holmgård, A. Liapis, J. Togelius et G. N. Yannakakis, "Evolving personas for player decision modeling," dans *2014 IEEE Conference on Computational Intelligence and Games*. IEEE, 2014, p. 1–8.

[116] C. Holmgard, M. C. Green, A. Liapis et J. Togelius, "Automated playtesting with procedural personas with evolved heuristics," *IEEE Transactions on Games*, 2018.

[117] C. A. C. Coello, G. B. Lamont, D. A. Van Veldhuizen *et al.*, *Evolutionary algorithms for solving multi-objective problems*. Springer, 2007, vol. 5.

[118] S. Dahlskog et J. Togelius, "Procedural content generation using patterns as objectives," dans *European Conference on the Applications of Evolutionary Computation*. Springer, 2014, p. 325–336.

[119] V. Valtchanov et J. A. Brown, "Evolving dungeon crawler levels with relative placement," dans *Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering*. ACM, 2012, p. 27–35.

[120] D. Ashlock, C. Lee et C. McGuinness, "Search-based procedural generation of maze-like levels," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, n°. 3, p. 260–273, 2011.

[121] Blizzard, "Starcraft," 1998.

[122] P. D. Surry, N. J. Radcliffe *et al.*, "The comoga method : constrained optimisation by multi-objective genetic algorithms," *Control and Cybernetics*, vol. 26, p. 391–412, 1997.

[123] K. Wiese et S. D. Goodwin, "Keep-best reproduction : a selection strategy for genetic algorithms," dans *Proceedings of the 1998 ACM symposium on Applied Computing*, 1998, p. 343–348.

[124] E. Mezura-Montes, C. A. Coello Coello et E. I. Tun-Morales, "Simple feasibility rules and differential evolution for constrained optimization," dans *Mexican International Conference on Artificial Intelligence*. Springer, 2004, p. 707–716.

[125] S. O. Kimbrough, G. J. Koehler, M. Lu et D. H. Wood, "On a feasible–infeasible two-population (fi-2pop) genetic algorithm for constrained optimization : Distance tracing and no free lunch," *European Journal of Operational Research*, vol. 190, n°. 2, p. 310–327, 2008.

[126] Z. Michalewicz *et al.*, "A survey of constraint handling techniques in evolutionary computation methods." *Evolutionary programming*, vol. 4, p. 135–155, 1995.

[127] Z. Michalewicz, "Genetic algorithms, numerical optimization, and constraints," dans *Proceedings of the sixth international conference on genetic algorithms*, vol. 195. Citeseer, 1995, p. 151–158.

[128] E. Mezura-Montes et C. A. C. Coello, "Constraint-handling in nature-inspired numerical optimization : past, present and future," *Swarm and Evolutionary Computation*, vol. 1, n°. 4, p. 173–194, 2011.

[129] P. Merrell, "Example-based model synthesis," dans *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, 2007, p. 105–112.

[130] ——, "Model synthesis," [En ligne]. Disponible : https://paulmerrell.org/model-synthesis/, consulté le 21/06/2022.

[131] C. E. Shannon et W. Weaver, *The Mathematical Theory of Communication.* Urbana, IL : University of Illinois Press, 1949.

[132] J. Parker, "Generating worlds with wave function collapse," [En ligne]. Disponible : https://www.procjam.com/tutorials/wfc/, consulté le 21/06/2022.

[133] R. Heaton, "The wavefunction collapse algorithm explained very clearly," 2018, [En ligne]. Disponible : https://robertheaton.com/2018/12/17/wavefunction-collapse-algorithm/, consulté le 21/06/2022.

[134] B. T. Brave, "Wave function collapse explained," 2020, [En ligne]. Disponible : https://www.boristhebrave.com/2020/04/13/wave-function-collapse-explained/, consulté le 21/06/2022.

[135] A. Newgas, "Tessera : A practical system for extended wavefunctioncollapse," dans *The 16th International Conference on the Foundations of Digital Games (FDG) 2021*, 2021, p. 1–7.

[136] I. Karth et A. M. Smith, "Wavefunctioncollapse is constraint solving in the wild," dans *Proceedings of the 12th International Conference on the Foundations of Digital Games.* ACM, 2017, p. 68.

[137] M. Gumin, "Basic wave function collapse 3d," 2016, [En ligne]. Disponible : https://bitbucket.org/mxgmn/basic3dwfc/src/master/, consulté le 16/01/2020.

[138] J. Parker, "Unity wave function collapse," 2016, [En ligne]. Disponible : https://selfsame.itch.io/unitywfc, consulté le 25/10/2019.

[139] O. M. Joseph Parker, Ryan Jones, "Proc skater 2016," 2016, [En ligne]. Disponible : https://arcadia-clojure.itch.io/proc-skater-2016, consulté le 25/10/2019.

[140] R. Devaux, "Week 60 : Lots of things," 2017, [En ligne]. Disponible : https://trasevol.dog/2017/06/19/week60/, consulté le 16/01/2020.

[141] A. Wallace, "Maureen's chaotic dungeon," 2019, [En ligne]. Disponible : https://globalgamejam.org/2019/games/maureens-chaotic-dungeon, consulté le 25/10/2019.

[142] M. Kleineberg, "Infinite city," 2019, [En ligne]. Disponible : https://marian42.de/article/wfc/, consulté le 25/10/2019.

[143] O. Stalberg, “Hybrid between marching cubes and wave function collapse,” 2019, [En ligne]. Disponible : https://twitter.com/OskSta/status/1181464374839521280, consulté le 16/01/2020.

[144] ——, “Wave - by oskar stålberg,” 2017, [En ligne]. Disponible : http://oskarstalberg.com/game/wave/wave.html, consulté le 16/01/2020.

[145] PlausibleConcept, “Bad north,” 2018.

[146] FreeholdGames, “Caves of qud,” 2015.

[147] M. O’Leary, “Poetry generator inspired by wave function collapse,” 2016, [En ligne]. Disponible : https://twitter.com/mewo2/status/789167437518217216, consulté le 16/01/2020.

[148] W. Gaisbauer et H. Hlavacs, “Procedural attack ! procedural generation for populated virtual cities : a survey,” *International Journal of Serious Games*, vol. 4, n°. 2, p. 19–29, 2017.

[149] W. Gaisbauer, W. L. Raffe, J. A. Garcia et H. Hlavacs, “Procedural generation of video game cities for specific video game genres using wavefunctioncollapse (wfc),” dans *Extended Abstracts of the Annual Symposium on Computer-Human Interaction in Play Companion Extended Abstracts*, 2019, p. 397–404.

[150] B. Lin, W. Jabi et R. Diao, “Urban space simulation based on wave function collapse and convolutional neural network,” 2020.

[151] D. Cheng, H. Han et G. Fei, “Automatic generation of game levels based on controllable wave function collapse algorithm,” dans *International Conference on Entertainment Computing*. Springer, 2020, p. 37–50.

[152] A. Sandhu, Z. Chen et J. McCoy, “Enhancing wave function collapse with design-level constraints,” dans *Proceedings of the 14th International Conference on the Foundations of Digital Games*, 2019, p. 1–9.

[153] B. Fritzke, “Growing grid—a self-organizing network with constant neighborhood range and adaptation strength,” *Neural processing letters*, vol. 2, n°. 5, p. 9–13, 1995.

[154] N. Chomsky, “Three models for the description of language,” *IRE Transactions on information theory*, vol. 2, n°. 3, p. 113–124, 1956.

[155] P. Prusinkiewicz, "Graphical applications of l-systems," dans *Proceedings of graphics interface*, vol. 86, n°. 86, 1986, p. 247–253.

[156] J. Dormans, "Adventures in level design : generating missions and spaces for action adventure games," dans *Proceedings of the 2010 workshop on procedural content generation in games*. ACM, 2010, p. 1.

[157] G. Rozenberg, *Handbook of Graph Grammars and Comp.* World scientific, 1997, vol. 1.

[158] G. Stiny et J. Gips, "Shape grammars and the generative specification of painting and sculpture." dans *IFIP Congress (2)*, vol. 2, n°. 3, 1971.

[159] D. Adams *et al.*, "Automatic generation of dungeons for computer games," *Bachelor thesis, University of Sheffield, UK.*, 2002.

[160] Ludomotion, "Unexplored," 2017.

[161] R. Van der Linden, R. Lopes et R. Bidarra, "Designing procedurally generated levels," dans *Proceedings of the the second workshop on Artificial Intelligence in the Game Design Process*, 2013.

[162] Y. I. Parish et P. Müller, "Procedural modeling of cities," dans *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM, 2001, p. 301–308.

[163] P. Müller, P. Wonka, S. Haegler, A. Ulmer et L. Van Gool, "Procedural modeling of buildings," dans *Acm Transactions On Graphics (Tog)*, vol. 25, n°. 3. ACM, 2006, p. 614–623.

[164] P. Prusinkiewicz et A. Lindenmayer, *The Algorithmic Beauty of Plants*. Berlin, Heidelberg : Springer-Verlag, 1990.

[165] R. Abela, A. Liapis et G. N. Yannakakis, "A constructive approach for the generation of underwater environments," dans *Proceedings of the FDG workshop on Procedural Content Generation in Games*. Citeseer, 2015.

[166] B. Mark, T. Berechet, T. Mahlmann et J. Togelius, "Procedural generation of 3d caves for games on the gpu." dans *FDG*, 2015.

[167] R. L. DuBois, "Applications of generative string-substitution systems in computer music," *Columbia University*, 2003.

[168] N. S. Goel et I. Rozehnal, "Some non-biological applications of l-systems," *International Journal Of General System*, vol. 18, n°. 4, p. 321–405, 1991.

[169] M. Mateas et A. Stern, "Façade : An experiment in building a fully-realized interactive drama," dans *Game developers conference*, vol. 2, 2003, p. 4–8.

[170] J. M. Font, R. Izquierdo, D. Manrique et J. Togelius, "Constrained level generation through grammar-based evolutionary algorithms," dans *Applications of Evolutionary Computation*, G. Squillero et P. Burelli, édit. Cham : Springer International Publishing, 2016, p. 558–573.

[171] I. M. Dart, G. De Rossi et J. Togelius, "Speedrock : procedural rocks through grammars and evolution," dans *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*. ACM, 2011, p. 8.

[172] T. W. Malone, "Toward a theory of intrinsically motivating instruction," *Cognitive science*, vol. 5, n°. 4, p. 333–369, 1981.

[173] R. Koster, *Theory of fun for game design*. " O'Reilly Media, Inc.", 2013.

[174] Ubisoft, "Far cry."

[175] FromSoftware, "Elden ring," 2022.

[176] R. Hunicke, M. Leblanc et R. Zubek, "Mda : A formal approach to game design and game research," *AAAI Workshop - Technical Report*, vol. 1, 01 2004.

[177] L. Cardamone, G. N. Yannakakis, J. Togelius et P. L. Lanzi, "Evolving interesting maps for a first person shooter," dans *European Conference on the Applications of Evolutionary Computation*. Springer, 2011, p. 63–72.

[178] K. Hullett, "Cause-effect relationships between design patterns and designer intent in fps levels," dans *Proceedings of the Second Workshop on Design Patterns in Games (DPG 2013), ser. DPG*, vol. 13, 2013.

[179] J. Kruse, R. Sosa et A. M. Connor, "Procedural urban environments for fps games," dans *Proceedings of the Australasian computer science week multiconference*. ACM, 2016, p. 77.

[180] A. Liapis, G. N. Yannakakis et J. Togelius, "Towards a generic method of evaluating game levels." dans *AIIDE*, 2013.

[181] D. Yuret, "From genetic algorthms to efficient optimization," Thèse de doctorat, Massachusetts Institute of Technology, 1994.

[182] A. Kramarzewski et E. De Nucci, *Practical game design : learn the art of game design through applicable skills and cutting-edge insights*. Packt Publishing Ltd, 2018.

[183] D. Berlyne, *Conflict, Arousal, and Curiosity*, ser. Conflict, Arousal, and Curiosity. McGraw-Hill, 1960.

[184] D. E. Berlyne, "Novelty, complexity, and hedonic value," *Perception & psychophysics*, vol. 8, n°. 5, p. 279–286, 1970.

[185] T. T. Constant et G. Levieux, "Perception de la variabilité et facteurs de la curiosité dans les jeux vidéo," CNAM, Conservatoire national des arts et métiers ; CEDRIC laboratory, CNAM-Paris, France, Research Report, févr. 2022. [En ligne]. Disponible : https://hal.archives-ouvertes.fr/hal-03602170

[186] G. Snook, "Simplified 3d movement and pathfinding using navigation meshes," *Game programming gems*, vol. 1, n°. 1, p. 288–304, 2000.

[187] D. Giacomelli, "Geneticsharp," 2013, [En ligne]. Disponible : https://github.com/giacomelli/GeneticSharp, consulté le 02/07/2021.

[188] M.-W. Tsai, T.-P. Hong et W.-T. Lin, "A two-dimensional genetic algorithm and its application to aircraft scheduling problem," *mathematical Problems in Engineering*, vol. 2015, 2015.

[189] R. Bailly et G. Levieux, "Genetic-wfc : Extending wave function collapse with genetic search," *IEEE Transactions on Games*, p. 1–10, 2022.

[190] ——, "Genetic WFC : Procedural Level Generation Maximizing Perceived Diversity," dans *IFIP International Conference on Entertainment Computing 2021*, Coimbra, Portugal, nov.

2021, cet article a été réalisé dans le cadre d'un consortium doctoral. [En ligne]. Disponible : https://hal.archives-ouvertes.fr/hal-03807178

[191] T. Fullerton, *Game design workshop : a playcentric approach to creating innovative games.* CRC press, 2014.

[192] M.-y. Li, Z.-x. Cai et G.-y. Sun, "An adaptive genetic algorithm with diversity-guided mutation and its global convergence property," *Journal of Central South University of Technology*, vol. 11, p. 323–327, 09 2004.

[193] I.-C. Oniscu, "Outpost design in open world game - analysis part 01," 2020, [En ligne]. Disponible : https://iuliu-cosmin-oniscu.medium.com/outpost-design-in-open-world-game-analysis-part-01-3944beefda21, consulté le 27/06/2022.

[194] ——, "Outpost design in open world game - analysis part 02," 2020, [En ligne]. Disponible : https://iuliu-cosmin-oniscu.medium.com/outpost-design-in-open-world-game-analysis-part-02-dade2b615742, consulté le 27/06/2022.

[195] Whitebox, "An analysis of far cry 5 : Outposts," 2018, [En ligne]. Disponible : https://theconstruct.wixsite.com/whitebox/single-post/2018/03/31/an-analysis-of-far-cry-5-outposts, consulté le 27/06/2022.

[196] ——, "An analysis of assassin's creed origins : Strongholds," 2018, [En ligne]. Disponible : https://theconstruct.wixsite.com/whitebox/single-post/2018/06/11/an-analysis-of-assassins-creed-origins-strongholds, consulté le 27/06/2022.

[197] P. Bergeron, "[gdc] 360 approach for open world mission design in far cry and assassin's creed," 2016, [En ligne]. Disponible : https://www.youtube.com/watch?v=knWzlR8mGWA, consulté le 27/06/2022.

Annexes

Annexe A

Formalisation d'un fortin

Contenu

A.1 Introduction

FIGURE A.1 – Campement typique de *Ghost Recon : Wildlands* [24].

Dans le cadre de cette thèse, nous souhaitions découvrir quels sont les facteurs pouvant maximiser la perception de variabilité chez le joueur. Pour ce faire, nous nous sommes intéressés à l'étude des lieux appelés campements présents, entre autres, dans les jeux d'Ubisoft et plus particulièrement dans *Ghost Recon : Widlands*, voir Figure A.1. En effet, notre collaboration avec Ubisoft Bordeaux s'est axé sur cette licence, développée en partie dans leurs locaux. Nous espérions développer une solution de génération procédurale afin de construire ces fortins et d'y réaliser nos tests. Ainsi, nous procéderons, dans les prochaines parties, à la formalisation des différents composants d'un campement. Nous commencerons tout d'abord par une présentation du jeu *Ghost Recon : Wildlands* [24] avant de détailler les mécaniques propres aux fortins présents dans ce titre.

A.2 Ghost Recon : Wildlands

Ghost Recon Wildlands est le dixième volet de la très populaire franchise *Ghost Recon*. Le jeu se déroule en Bolivie, une nation d'Amérique du Sud connue pour ses paysages typiques et sa diversité écologique. Son objectif était de recréer ce pays de manière authentique, afin de l'explorer virtuellement. Ceci a permis, notamment, au monde ouvert de ce jeu d'être l'un des plus vastes d'Ubisoft, tout en conservant le gameplay tactique représentatif de la marque *Ghost Recon*.

FIGURE A.2 – Le monde ouvert de *Ghost Recon : Wildlands* [24].

Scénario Le pays de Bolivie est gangréné par la drogue. En effet, il s'agit du plus grand fournisseur au monde, notamment de cocaïne. La production de ces substances illicites est contrôlée par le cartel de la drogue de Santa Blanca, une organisation puissante dont l'influence a déstabilisé la région. Ces cartels qui ne cessent de monter en puissance, sont devenus une menace mondiale, et, de ce fait, préoccupent le gouvernement des États-Unis. Par conséquent, l'armée américaine envoie une unité d'opérations spéciales d'élites appelée *Ghosts*, « Fantômes » en français, ayant pour but de révéler et de démanteler les connexions qui existent entre le cartel de la drogue et le gouvernement local du pays.

Gameplay Concernant les mécaniques propres à ce titre, il s'agit avant tout d'un jeu de tir tactique à la troisième personne, se déroulant dans un monde ouvert, voir Figure A.2. Le joueur dispose de plusieurs gadgets améliorables tels que des drones et des grenades spéciales. Ces gadgets peuvent être débloqués et améliorés grâce à de nombreuses ressources parsemées dans le monde ouvert. Ces dernières peuvent être obtenues, par exemple, en explorant des installations ou en complétant des missions secondaires telles que détourner des convois, voir Figure A.3 page suivante, ou dérober des avions remplis de matériels.

FIGURE A.3 – Détournement d'hélicoptère et récupération de ressources médicales.

A.3 Présentation d'un campement

Le jeu *Ghost Recon : Wildlands* d'Ubisoft confronte le joueur à deux groupes d'ennemis fictionnels, disséminés sur la gigantesque carte du jeu. Ces opposants peuvent affronter le joueur, notamment dans des lieux spécifiques appelés campements. Il s'agit de zones fortifiées, aux mains des ennemis, dans lesquelles le joueur pourra être amené à utiliser ses diverses compétences. La structure même et l'agencement des campements influenceront alors le choix du joueur quant à sa méthode d'approche, tel que l'infiltration ou le combat direct, afin d'atteindre divers objectifs proposés par les lieux. Les motivations personnelles du joueur l'inciteront, par exemple, soit à réussir une mission du jeu, soit simplement à récupérer des points de compétences.

Nous nous proposons de présenter l'approche d'un campement selon les étapes suivantes :

1. Tout d'abord, le joueur va tenter d'obtenir des informations concernant le campement. Pour cela, il peut effectuer une reconnaissance des lieux en amont, avant de s'approcher. L'utilisation d'un drone pourra l'aider à repérer et à identifier des ennemis potentiels, ainsi que leurs patrouilles, et d'autres dangers. En effet, ce drone permet d'utiliser diverses compétences, notamment l'aide des rebelles, nos alliés, afin de localiser les ennemis ou de les attaquer directement. Le joueur aura alors le choix d'engager directement le combat à distance grâce au drone ou à son propre armement. En d'autres termes, l'utilisation du drone permet de se familiariser avec le campement avant d'engager le combat ou de procéder à une infiltration.

2. Après avoir procédé à la prospection des lieux, le joueur sera confronté au choix du point d'accès lui permettant de pénétrer dans l'enceinte du campement. En effet, les développeurs ont imaginé diverses routes d'accès au fortin. Le joueur va pouvoir appliquer sa stratégie d'approche, c'est-à-dire sélectionner une route d'infiltration à emprunter et engager le combat, que ce soit de manière volontaire ou par nécessité. Si nous prenons comme exemple le campement présenté Figure A.4 page suivante, il est possible d'approcher ce dernier par plusieurs voies terrestres, mais également par les airs ou par la voie maritime. Le joueur pourra accéder par la suite à l'intérieur du fortin, que ce soit par les entrées principales ou par des portes non gardées situées sur les flancs.

3. Une fois à l'intérieur de ces bases ennemies, il existe divers trajets d'exploration et d'itinéraires guidant le joueur vers ses objectifs. Afin de varier les situations de jeu, certains endroits se trouveront dans la ligne de mire des snipers, d'autres seront considérés comme des embuscades tant pour le joueur que pour les ennemis. La plupart du temps, l'architecture globale du campement incitera le joueur à suivre un itinéraire composé de couvertures afin qu'il puisse se faufiler sans alerter les adversaires, voir Figure A.5 page suivante.

D'autre part, plusieurs éléments peuvent également influencer les décisions du joueur, à savoir, par exemple, la météo ou un état d'alerte éventuel dans le campement.

En effet, les conditions météorologiques constituent un élément modificateur pouvant influencer la partie du joueur, que ce soit par sa vision ou ses actions. Sa vision peut se trouver réduite à cause de la pluie et de l'obscurité. Il semble intéressant de noter que les adversaires sont également impactés par la météo.

De plus, si le joueur est repéré par un ennemi, il se peut qu'une alarme soit déclenchée dans le campement. Ceci peut entrainer l'appel de renfort et un rassemblement d'ennemis sur la position du joueur. Ce dernier se verra alors contraint d'adapter sa stratégie à la nouvelle situation. Il peut dans ce cas, par exemple, tenter de s'échapper si la difficulté semble trop importante, ou choisir d'affronter les vagues d'ennemis tout en poursuivant ses objectifs.

FIGURE A.4 – Diverses voies d'accès possibles dans un campement, les flèches en bleu représentent les points d'entrée.

FIGURE A.5 – Divers chemins composés de couvertures, suggérés au joueur [25].

A.4 Composition d'un campement

Nous allons à présent nous pencher, plus en détail, sur les divers éléments pouvant composer un campement, liste non exhaustive. Pour ce faire, nous avons identifié plusieurs catégories d'agencement d'objets et d'interactions possibles dans un fortin.

A.4.1 Objets

- **Placement des bâtiments et des couvertures**

  Les couvertures vont suggérer plusieurs voies possibles au joueur et permettront de prendre l'ennemi à revers, voir Figure A.5 page précédente.

  Le placement des bâtiments va également influencer la visibilité du joueur et lui donner un aperçu des routes empruntables.

- **Les ennemis IA**

  Plusieurs archétypes d'ennemis sont positionnés stratégiquement dans le campement. Nous pouvons citer comme exemple des snipers embusqués dans des tours. Ces adversaires vont offrir au joueur, défi et réflexion quant à la stratégie à adopter. De plus, les ennemis peuvent patrouiller sur différentes routes, mais également rester statiques ou encore effectuer des actions « aléatoires », comme fumer une cigarette.

- **Quatre ressources différentes de matériaux**

  Elles permettent de débloquer les compétences du personnage grâce au balisage de :

  - Caisse de médicaments ;
  - Baril de pétrole ;
  - Nourritures ;
  - Transmission.

- **Les collectibles**

  Il s'agit d'objets à récupérer, par exemple :

  - Dossier d'enregistrement, offre des informations sur le « lore » du jeu ;
  - *Intel*, renseignement critique, permet de débloquer des missions ;
  - Caisse d'armement, permet de débloquer des armes ;
  - Caisse d'accessoires pour les armes ;
  - Caisse de munitions pour les armes ;

  - Médaille bonus, offre une compétence supplémentaire, par exemple, -5% sur les dégâts causés par les balles ;
  - Document offrant un point de compétence.

- **Les véhicules**

- **« Ingrédients » de *level design***

  Ils offrent diverses interactions supplémentaires. Nous pouvons citer comme exemples d'« ingrédients » :

  - Brouilleur de drone ;
  - Mortier ;
  - Alarme ;
  - Projecteur, représente une difficulté supplémentaire pour l'infiltration de nuit ;
  - Explosifs, par exemple, baril ou conteneur de gaz ;
  - Prison de rebelles ;
  - Générateur électrique.

  Ces « ingrédients » vont permettre une variabilité dans le parcours et les actions du joueur à travers le campement. Ils pourront également modifier la difficulté globale du fortin et proposer un défi au joueur.

A.4.2 Interactions

Le jeu propose également au joueur diverses actions possibles selon les éléments inclus dans le campement, à savoir par exemple :

- Libérer des prisonniers, permet une diversion, car les personnes libérées reviennent attaquer les gardes ;
- Détruire un générateur électrique, permet d'ouvrir les portails d'entrée du campement ;
- Interroger un général ennemi, permet d'obtenir, par exemple, la localisation de collectibles.

A.4.3 Synthèse

Finalement, tous ces éléments énoncés précédemment vont, entre autres :

- Forcer le joueur à utiliser ou à maîtriser une ou plusieurs compétences ;
- Influencer la difficulté du campement, à savoir le nombre et le niveau d'armement des ennemis ;
- Focaliser l'approche du joueur sur un type de gameplay précis, par exemple, l'infiltration ou l'affrontement ;
- Proposer un thème émotionnel.

A.5 Ghost Recon : Breakpoint

FIGURE A.6 – Exemple de campement ennemi dans le jeu *Ghost Recon : Breakpoint* [26].

Il nous a semblé intéressant de noter qu'une suite à *Ghost Recon : Wildlands*, appelée *Ghost Recon : Breakpoint* [26] a également été développée par Ubisoft. Ce jeu représente une itération du gameplay de *Wildlands* dans une nouvelle carte de jeu, davantage axée sur la technologie de drones et leurs dangers futurs dans notre société. Ubisoft Bordeaux a eu l'occasion de travailler en support sur ce titre. Cependant, l'échec commercial du titre a rapidement freiné l'intérêt concernant l'étude plus en détail de ce jeu. Néanmoins, la structure du gameplay trouve sa source dans l'épisode précédent, notamment vis-à-vis des gadgets, des armes et des campements, voir Figure A.6. Toutefois, quelques

nouveautés sur la variété des missions secondaires ont été élaborées grâce à un système de génération procédurale de quêtes journalières. Ceci avait comme objectif d'insuffler de l'intérêt supplémentaire pour le joueur afin de le motiver à relancer une partie supplémentaire.

A.6 Conclusion

Dans cette brève analyse, nous avons détaillé la structure des campements pour le jeu *Ghost Recon : Wildlands*. Ces fortins apparaissent également, de manière similaire, dans d'autres séries de jeux Ubisoft, à savoir *Far Cry* ou encore *Assassin's Creed*. Ainsi, le joueur se voit proposer un terrain de jeu avec différents objectifs qu'il pourra ou non suivre selon son bon vouloir, à savoir, par exemple, récupérer des points de compétences, collectionner les collectibles ou encore libérer des prisonniers. La disposition des éléments du campement et différents facteurs vont encourager le joueur à improviser son approche au fur et à mesure de son exploration. Sa capacité d'adaptation aux différents scénarios dépendra alors de ses propres compétences et des ressources mises à sa disposition.

Une partie de notre étude visait à maximiser la variabilité perçue par le joueur, et devait nous permettre d'identifier de nouveaux facteurs dans la distribution des éléments précédemment identifiés, constituant le campement même. Nous espérions découvrir ainsi des variables d'agencement pour les différentes couches d'objets influençant la curiosité, facteur de motivation chez le joueur.

A.7 Sources et informations supplémentaires

Davantage d'analyses sur les jeux Ubisoft précédemment cités sont disponibles :

- pour *Ghost Recon : Wildlands* [25] ;
- pour *Ghost Recon : Breakpoint* [193, 194] ;
- pour *Far Cry* [195] et *Assassin's Creed* [196], les deux [197].

Annexe B

Expérience 2 - Données supplémentaires

Exécution n°	Genetic-WFC				Genetic-WFC Non Réencodé			
	Nouveauté	Sécurité	Complexité	Total	Nouveauté	Sécurité	Complexité	Total
1	0,584	0,912	-0,046	1,450	0,494	0,919	-0,047	1,366
2	0,578	0,910	-0,045	1,443	0,484	0,918	-0,050	1,351
3	0,580	0,914	-0,045	1,449	0,499	0,915	-0,051	1,364
4	0,589	0,910	-0,046	1,454	0,499	0,915	-0,053	1,361
5	0,582	0,915	-0,046	1,451	0,511	0,915	-0,053	1,373
6	0,560	0,914	-0,041	1,434	0,496	0,918	-0,050	1,365
7	0,567	0,909	-0,048	1,428	0,480	0,917	-0,049	1,347
8	0,571	0,913	-0,046	1,438	0,487	0,917	-0,049	1,355
9	0,578	0,913	-0,047	1,444	0,487	0,916	-0,051	1,352
10	0,589	0,914	-0,047	1,457	0,497	0,913	-0,055	1,354
Moyenne	**0,578**	**0,913**	**-0,046**	**1,445**	**0,493**	**0,916**	**-0,051**	**1,359**
Ecart-type	**0,009**	**0,002**	**0,002**	**0,009**	**0,009**	**0,002**	**0,002**	**0,008**
Max	0,589	0,915	-0,041	1,457	0,511	0,919	-0,047	1,373
Min	0,560	0,909	-0,048	1,428	0,480	0,913	-0,055	1,347

Exécution n°	Algorithme génétique sans WFC				Algorithme génétique sans WFC pénalisé			
	Nouveauté	Sécurité	Complexité	Total	Nouveauté	Sécurité	Complexité	Total
1	0,572	0,925	-0,040	1,457	0,524	0,914	-0,046	1,392
2	0,574	0,916	-0,047	1,443	0,510	0,915	-0,047	1,378
3	0,571	0,924	-0,044	1,451	0,504	0,920	-0,037	1,387
4	0,555	0,923	-0,043	1,435	0,515	0,913	-0,051	1,377
5	0,573	0,923	-0,041	1,455	0,514	0,913	-0,047	1,380
6	0,574	0,922	-0,044	1,452	0,513	0,916	-0,044	1,385
7	0,577	0,921	-0,046	1,452	0,502	0,916	-0,048	1,370
8	0,566	0,926	-0,041	1,452	0,518	0,912	-0,047	1,383
9	0,574	0,923	-0,044	1,453	0,526	0,911	-0,049	1,388
10	0,570	0,918	-0,047	1,441	0,513	0,915	-0,050	1,378
Moyenne	**0,571**	**0,922**	**-0,044**	**1,449**	**0,514**	**0,915**	**-0,047**	**1,382**
Ecart-type	**0,006**	**0,003**	**0,002**	**0,007**	**0,008**	**0,003**	**0,004**	**0,007**
Max	0,577	0,926	-0,040	1,457	0,526	0,920	-0,037	1,392
Min	0,555	0,916	-0,047	1,435	0,502	0,911	-0,051	1,370

Exécution n°	WFC uniquement			
	Nouveauté	Sécurité	Complexité	Total
1	0,445	0,921	-0,044	1,322
2	0,451	0,921	-0,045	1,326
3	0,456	0,913	-0,051	1,318
4	0,457	0,916	-0,049	1,323
5	0,446	0,920	-0,043	1,323
6	0,454	0,914	-0,051	1,318
7	0,435	0,922	-0,042	1,315
8	0,465	0,917	-0,052	1,330
9	0,460	0,916	-0,047	1,329
10	0,449	0,916	-0,047	1,317
Moyenne	**0,452**	**0,917**	**-0,047**	**1,322**
Ecart-type	**0,009**	**0,003**	**0,004**	**0,005**
Max	0,465	0,922	-0,042	1,330
Min	0,435	0,913	-0,052	1,315

FIGURE B.1 – Valeurs détaillées du Tableau 6.3 page 144.

Annexe C

Expérience 3 - Mosaïques des niveaux

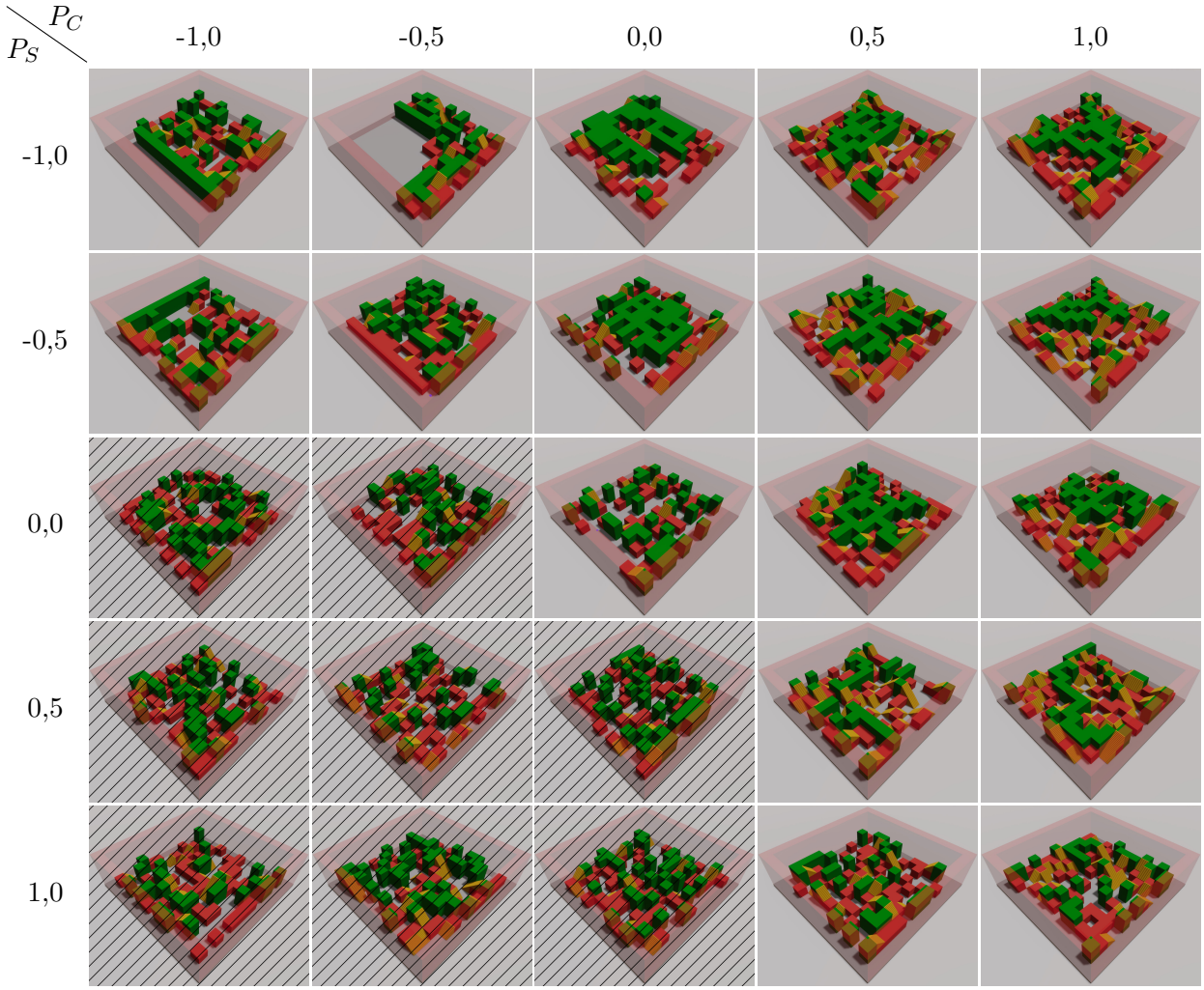


TABLE C.1 – Mosaïque des résultats pour $P_N = 0$. Les niveaux hachurés représentent les niveaux non navigables, c'est-à-dire qui possèdent un point de départ bloqué.

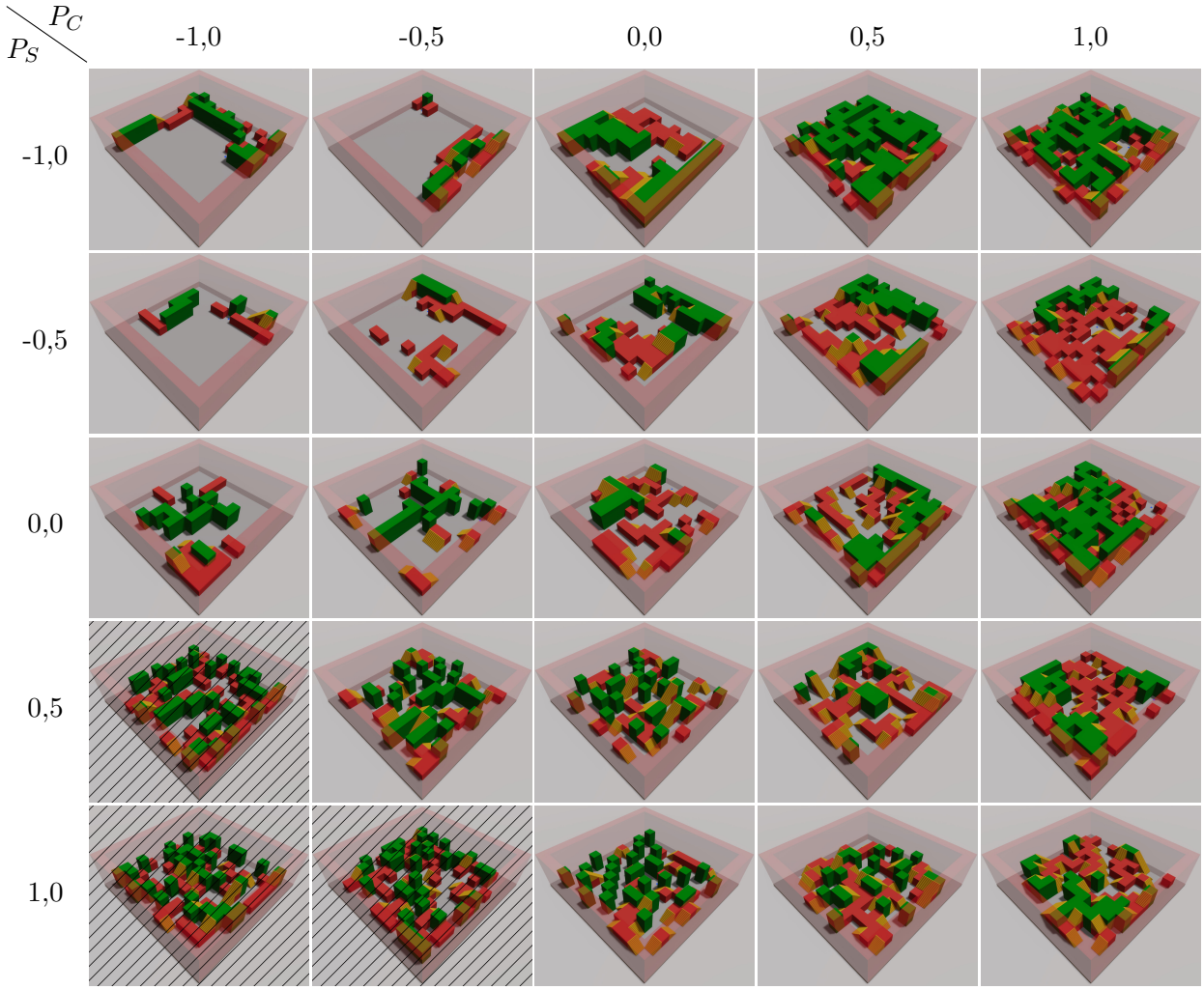


TABLE C.2 – Mosaïque des résultats pour $P_N = 0,125$. Les niveaux hachurés représentent les niveaux non navigables, c'est-à-dire qui possèdent un point de départ bloqué.

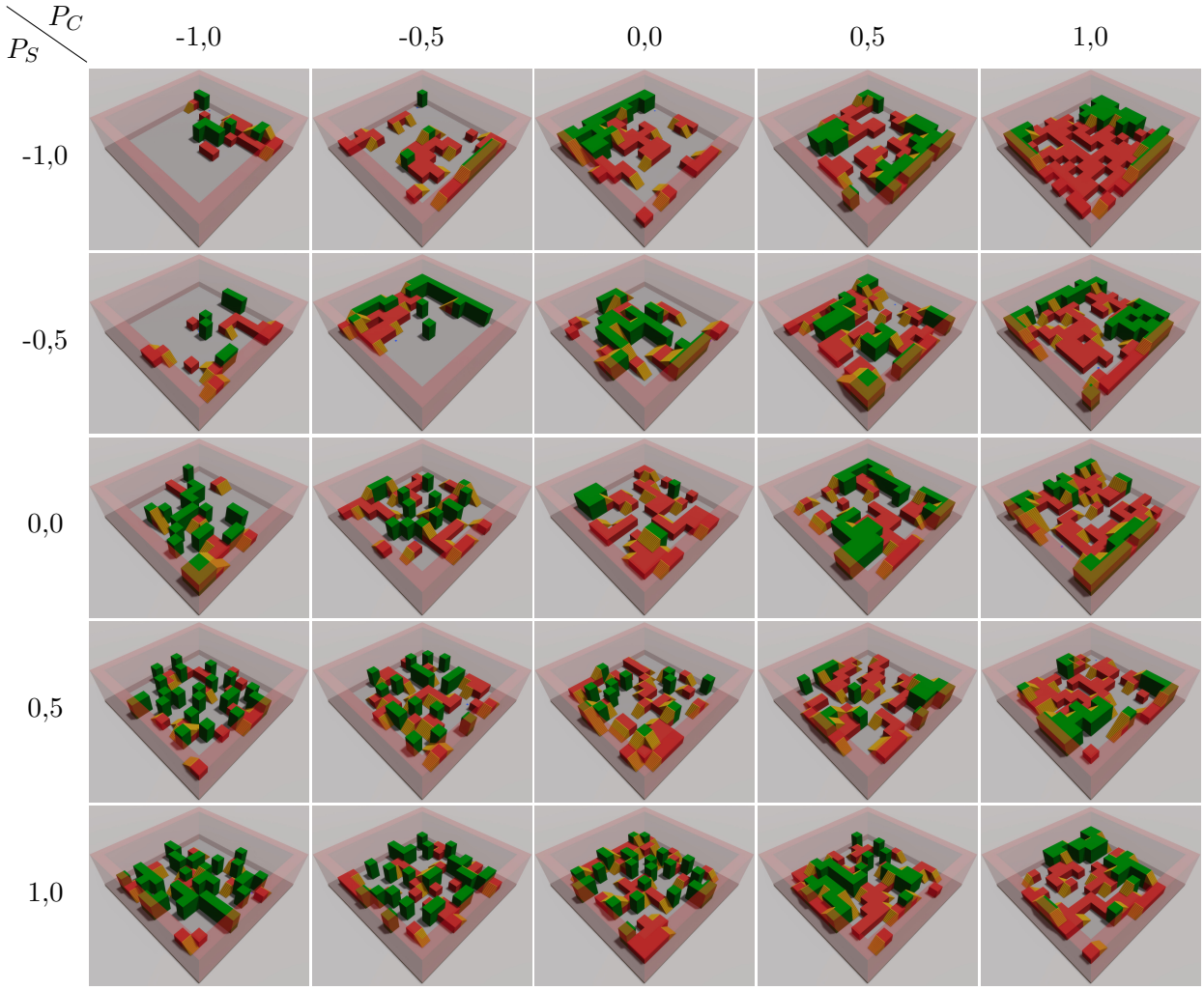


TABLE C.3 – Mosaïque des résultats pour $P_N = 0,25$.

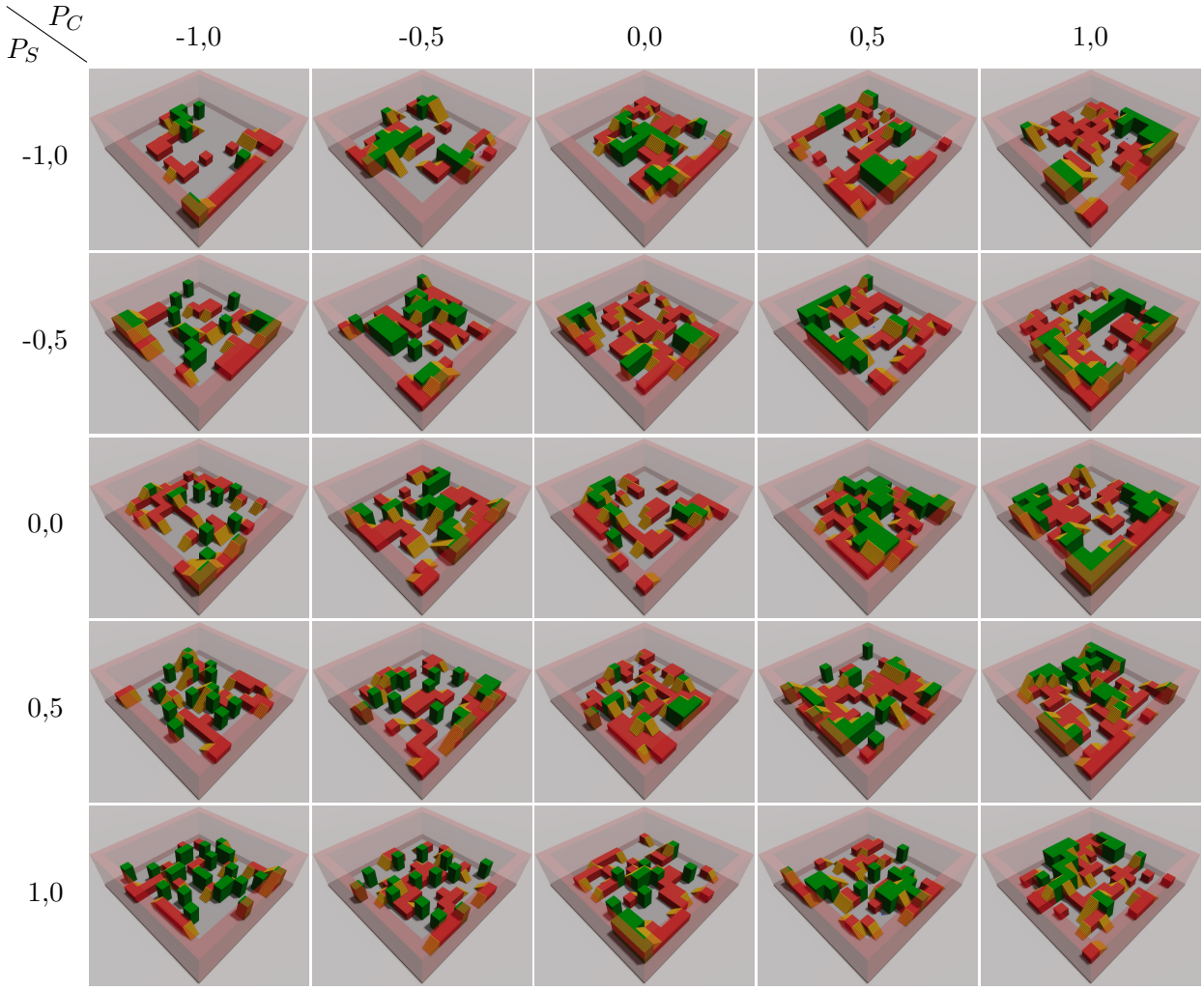


TABLE C.4 – Mosaïque des résultats pour $P_N = 0,5$.

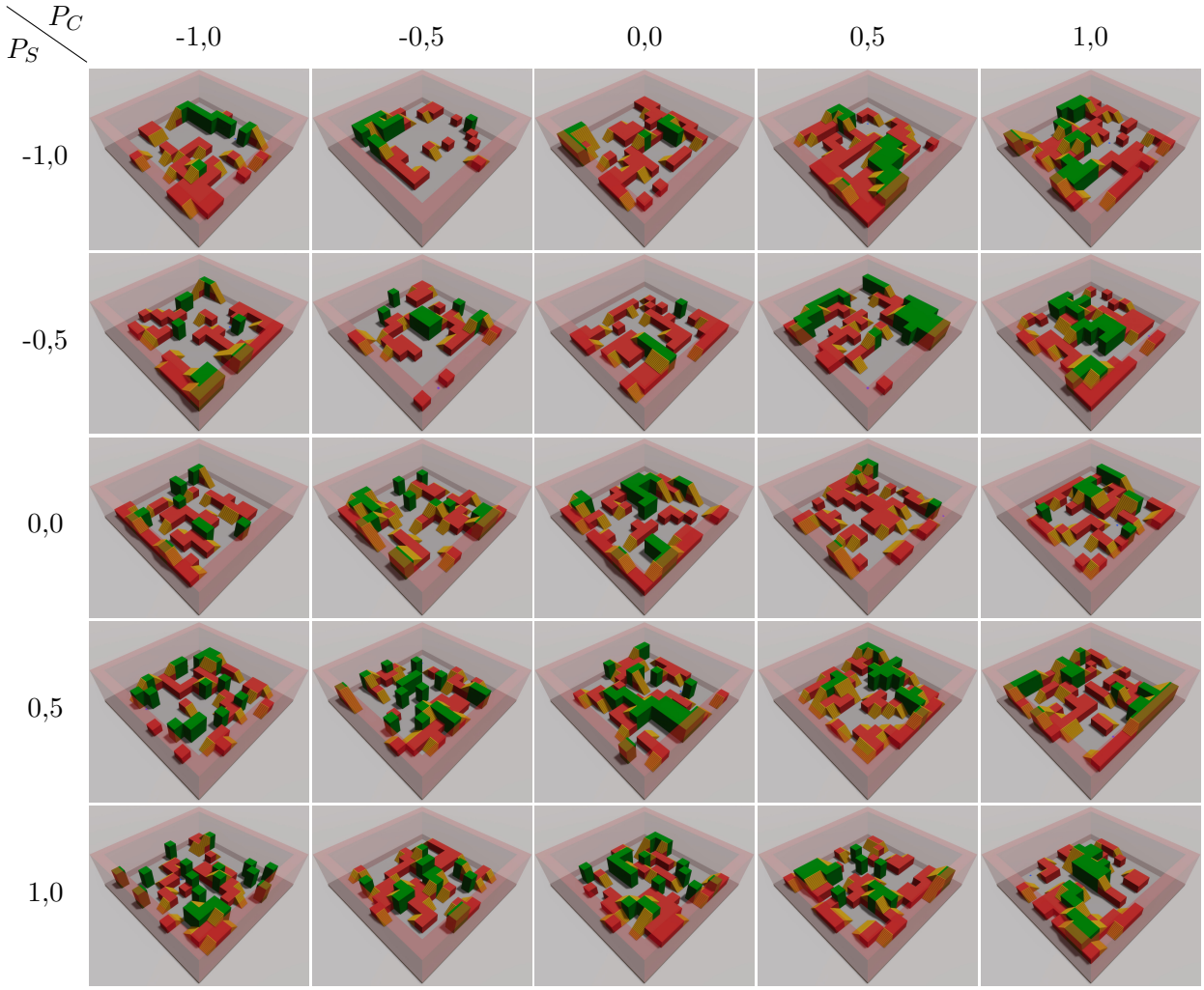


TABLE C.5 – Mosaïque des résultats pour $P_N = 0,75$.

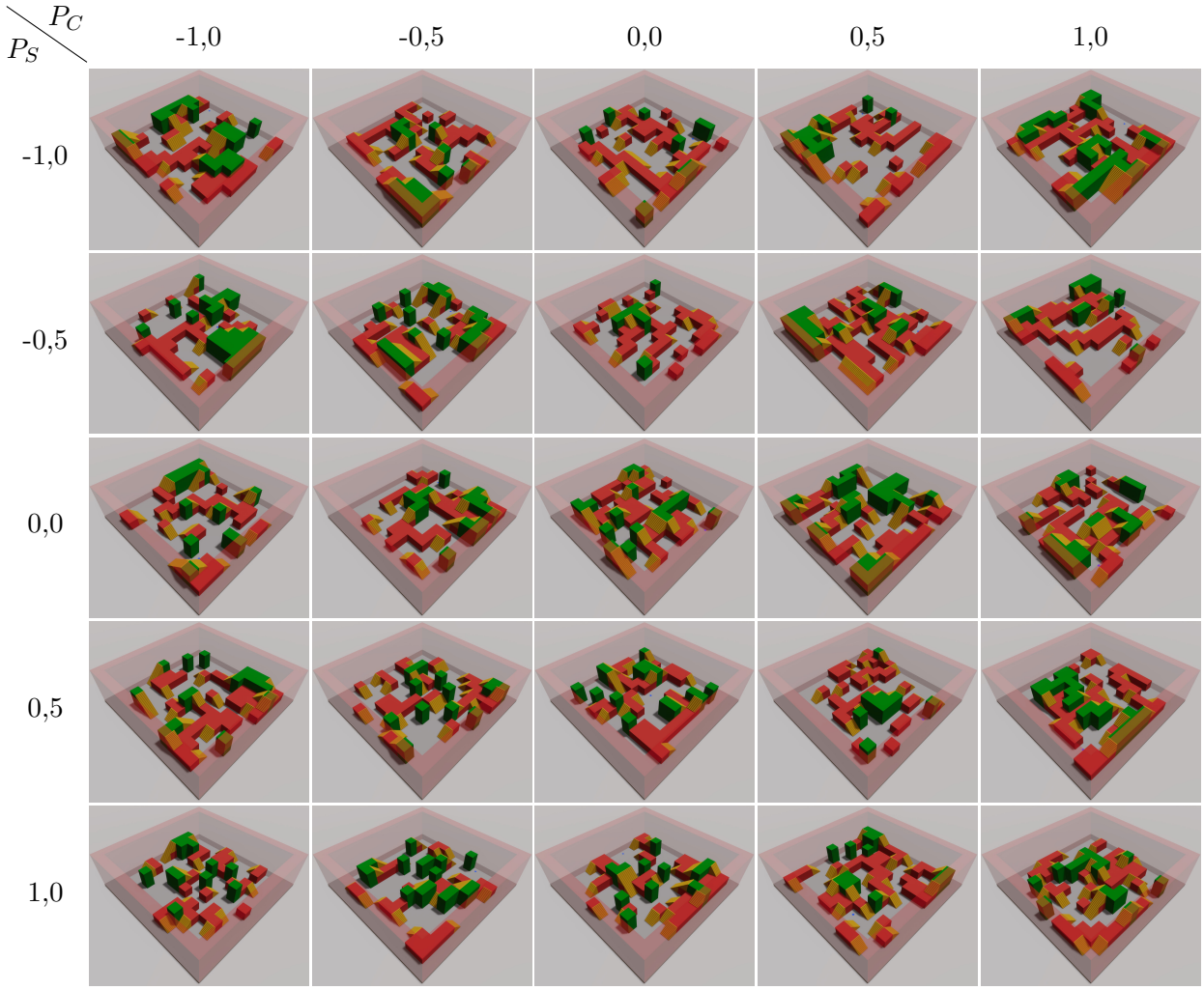


TABLE C.6 – Mosaïque des résultats pour $P_N = 1$.

Acronymes

AAA jeu vidéo dit « triple-A ».

FPS First-person shooter, jeu de tir à la première personne en français.

GAAS Game As A Service.

IA Intelligence Artificielle.

PCG Procedural Content Generation.

TPS Third-person shooter, jeu de tir à la troisième personne en français.

WFC Wave Function Collapse.

Glossaire

asset Ressource du jeu, éléments ou fichiers. Un asset fait référence habituellement à un élément de type graphique, audio, vidéo ou textuel [34, Page 656].
Remarque : ce terme est utilisé dans cette thèse et dans le contexte de l'implémentation d'un *Wave Function Collpase*, comme synonyme de module. Le terme module représente l'élément spécifique manipulé dans le déroulement d'un WFC .

design pattern Solution communément admise en réponse à un problème récurrent de conception [86, Chapitre 8]. Dans le jeu vidéo, il peut s'agir d'un arrangement caractéristique de game design ou level design, par exemple : un passage stratégique étroit, *choke point* en anglais, pour les jeux de tir exposant ainsi le joueur aux attaques ennemies [87].

dynamiques de jeu Correspondent au comportement émergent qui découle du gameplay, lorsque les mécanismes du jeu sont exploités [176].

entropie Mesure de l'incertitude et du désordre, qui peut se calculer selon la formule de Shannon : $-\sum_{i=1}^{n} p_i \log(p_i)$ [131].

fortin Un fortin ou campement ennemi, est un lieu comportant divers objectifs à atteindre pour le joueur, tels que la libération de prisonniers ou la collecte d'informations. La méthode d'approche est très souvent laissée au choix du joueur, voir Annexe A page 213.

game design Conception théorique du jeu vidéo : style, gameplay, genre de jeu, etc [37]. Il s'agit du processus de création et de mise au point des règles et autres éléments constitutifs d'un jeu.

gameplay Le gameplay définit la manière dont les joueurs peuvent interagir avec le jeu et comment ce dernier réagit à leurs actions [34, Page 661]. Il peut se décrire sous la forme de défis et d'actions

que le joueur doit exécuter pour atteindre l'objectif correspondant à un défi spécifique [35].

greyblock Dans notre recherche, ce terme se réfère à un bloc de construction intermédiaire, avant une couche visuelle, uniquement dans un but fonctionnel. Il s'agit, dans notre cas, d'une catégorie d'assets.

greyblocking Dans notre recherche, il s'agit d'une étape intermédiaire de conception de niveaux avec uniquement l'utilisation de catégories d'assets. Ce concept s'inpire d'une méthode de travail propre aux *level designers* appelée *greyboxing* [182, Chapitre 9].

level design Processus dans la création de jeu vidéo qui s'occupe de la réalisation des niveaux, c'est-à-dire la construction de l'espace de jeu à proprement parler [34, Chapitre 23].

mesh de navigation Structure de donnée utilisée en intelligence artificielle permettant de représenter les espaces navigables d'un environnement 3D, afin de permettre aux agents de s'y déplacer [186].

mixed-initiative Bien que l'initiative mixte n'ait pas de définition concrète, nous considérons que l'humain et l'ordinateur contribuent de manière proactive à la résolution d'un même problème [44]. Dans le cadre de la création de contenu, ce dernier est produit par des cycles itératifs entre le concepteur humain et le système procédural [45].

métaheuristique Algorithme d'optimisation.

méthode iterative Dans notre recherche, ce terme se réfère à l'approche *Search-Based* [12].

playtest Processus par lequel un concepteur propose à des individus de tester un nouveau jeu ou un nouveau concept afin d'obtenir des remarques et des impressions de la part des joueurs, pour améliorer l'expérience générale du jeu, détecter les bogues ou encore les défauts de conception [191, Chapitre 9].

triple-A Terme de classification utilisé pour les jeux vidéo dotés de budgets de développement et de promotion très élevés [29].

le cnam

Raphaël BAILLY

Génération procédurale de niveaux de jeu alliant approche constructive et optimisation

HESAM UNIVERSITÉ

Résumé : Notre travail de recherche se positionne dans le domaine de la génération procédurale de contenu dans le jeu vidéo. Cette thèse s'intéresse plus spécifiquement aux questionnements liés à la diversité et à la qualité du contenu généré. Ce sujet nous a interpellés et nous a conduit à l'étude de nombreuses méthodes et algorithmes, grâce à une revue de littérature de la génération procédurale de contenu. Nous nous sommes penchés sur la problématique suivante : « Comment aboutir à une méthode de génération proposant une forte diversité d'expériences de jeu, tout en conservant une certaine qualité structurelle dans ses résultats ? » Nous avons centré notamment notre étude sur la conception des niveaux de jeu et le placement d'objets. Nous avons ciblé la génération de niveaux 3D ouverts non-linéaires de type jeu de tir à la première personne, avec une infrastructure de campement dont le contenu est positionné sur une grille 2D. Cette thèse propose une nouvelle méthode, nommée *Genetic-WFC*, dans le but de répondre à notre problématique. Il s'agit d'un pipeline de génération procédurale qui combine un algorithme génétique et une évaluation par simulation avec le *Wave Function Collapse*, un algorithme de propagation de contraintes locales d'adjacence, afin de générer des niveaux ciblant des expériences de jeu spécifiques. Diverses expérimentations sur notre approche ont été menées afin d'établir, entre autres, ses performances face à d'autres méthodes similaires, et d'explorer la diversité des expériences de jeu que peut proposer notre algorithme de génération. Nous terminons en évoquant plusieurs pistes d'amélioration et de poursuite de recherche qui peuvent encore être approfondies.

Mots clés : Génération Procédurale de Contenu, Jeu Vidéo, Level Design, Diversité, Algorithme Génétique, Wave Function Collapse, Opérateur de Réparation, Expérience de Jeu, Simulation, Intelligence Artificielle

Abstract : The research work is positioned in the field of procedural content generation in video games. This study focuses more specifically on the questions related to the diversity and quality of the generated content. This subject raised our interest and led us to study numerous methods and algorithms, through a literature review of procedural content generation. We considered the following problematic : « How to achieve a generation method that offers a high degree of diversity of game experiences, while maintaining a certain structural quality in its results ? » Our study is centred in particular on the level design and the placement of objects in game levels. We also targeted a generation of non-linear open 3D levels for a first-person shooter, with a settlement infrastructure whose contents are positioned on a 2D grid. This thesis introduces a new method, named *Genetic-WFC*, in order to answer our research question. It is a procedural generation pipeline that combines a genetic algorithm and a simulation-based evaluation with the *Wave Function Collapse*, a local adjacency constraint propagation algorithm, to generate levels targeting specific game experiences. Various experiments on our approach have been performed in order to establish, among other things, its performance against other similar methods and to explore the diversity of game experiences that our generation algorithm can provide. We conclude by mentioning several directions for improvement and continuing research that can be further explored.

Keywords : Procedural Content Generation, Video Games, Level Design, Diversity, Genetic Algorithm, Wave Function Collapse, Repair Operator, Game Experience, Simulation, Artificial Intelligence