

# **Est-il possible de créer à partir d'un algorithme du contenu aussi engageant et captivant que du contenu créé par un designer ?**

**Travail de Bachelor réalisé en vue de l'obtention du Bachelor HES**

par :

**Corentin CLERC**

Conseiller au travail de Bachelor :

**Michaël MARTIN, vacataire à la HEG**

**Genève, le 13 septembre 2021**

**Haute École de Gestion de Genève (HEG-GE)**

**Filière Informatique de Gestion**

## Déclaration

Ce travail de Bachelor est réalisé dans le cadre de l'examen final de la Haute école de gestion de Genève, en vue de l'obtention du titre Bachelor of Science en Informatique de Gestion.

L'étudiant a envoyé ce document par email à l'adresse remise par son conseiller au travail de Bachelor pour analyse par le logiciel de détection de plagiat URKUND, selon la procédure détaillée à l'URL suivante : <https://www.orkund.com> .

L'étudiant accepte, le cas échéant, la clause de confidentialité. L'utilisation des conclusions et recommandations formulées dans le travail de Bachelor, sans préjuger de leur valeur, n'engage ni la responsabilité de l'auteur, ni celle du conseiller au travail de Bachelor, du juré et de la HEG.

« J'atteste avoir réalisé seul le présent travail, sans avoir utilisé des sources autres que celles citées dans la bibliographie. »

Fait à Genève, le 13 septembre 2021

Corentin Clerc

## Remerciements

J'aimerais tout d'abord remercier M. Michaël Martin, mon directeur pour ce travail. Ses conseils et son encadrement ont été d'une grande aide pour moi dans la réalisation de ce travail.

Je tiens aussi à remercier Cédric Bommart et Morgane Linder pour avoir relu ce travail, ainsi que les participants aux tests qui m'auront permis d'avoir des retours sur mon prototype.

Enfin, je souhaite remercier mes parents, ma famille et mes amis pour leur soutien lors de ces trois années passées à la HEG menant à la réalisation de ce travail.

## Résumé

Le jeu vidéo est aujourd'hui le plus important secteur dans le domaine du divertissement. La génération procédurale, qui correspond à l'utilisation d'algorithmes pour créer du contenu, a connu une explosion de nouveaux titres se reposant sur ce concept dans la dernière décennie. Une question se pose alors : est-il possible de créer à partir d'un algorithme du contenu aussi engageant et captivant que du contenu créé par un designer ?

Ce travail de bachelor a pour objectif d'apporter des éléments de réponse à cette question. Pour ce faire, j'ai conduit une étude sur l'état de l'art en matière de génération procédurale, afin d'en connaître l'histoire et les différentes utilisations de cette approche dans l'industrie pour générer du contenu. Avec cette étude, j'ai également conduit des recherches sur ce qui fait d'un jeu une expérience engageante et captivante, afin de dégager les points à prendre en compte lorsque l'on construit un algorithme.

J'ai également réalisé un prototype de jeu cherchant à prouver que du contenu généré par un algorithme peut être à la fois engageant et captivant, en créant un prototype de jeu de plateforme 3D générant ses niveaux de façon procédurale, en s'adaptant aux compétences et au rythme du joueur pour tenter de garder l'expérience dans un état idéal, quel que soit le profil du joueur.

# Table des matières

<b>Déclaration.....</b>	<b>i</b>
<b>Remerciements .....</b>	<b>ii</b>
<b>Résumé .....</b>	<b>iii</b>
<b>Liste des tableaux .....</b>	<b>vii</b>
<b>Liste des figures.....</b>	<b>vii</b>
<b>1. Introduction.....</b>	<b>1</b>
<b>2. La génération de contenu procédurale.....</b>	<b>2</b>
<b>2.1 Définition .....</b>	<b>2</b>
<b>2.2 Génération de contenu .....</b>	<b>2</b>
<b>2.3 Un peu d'histoire.....</b>	<b>3</b>
2.3.1 Avant l'informatique.....	3
2.3.2 Le premier contenu généré par ordinateur .....	3
2.3.3 Dans le domaine du jeu vidéo .....	4
2.3.4 La génération procédurale aujourd'hui .....	4
<b>2.4 Génération « aléatoire ».....</b>	<b>5</b>
<b>2.5 L'intelligence artificielle dans la pcg .....</b>	<b>7</b>
2.5.1 L'IA dans le jeu vidéo .....	7
2.5.2 La création de contenu.....	8
2.5.3 Le Machine Learning et le jeu vidéo .....	8
<b>2.6 Les raisons pour utiliser la génération procédurale.....</b>	<b>9</b>
2.6.1 Accélérer le développement.....	10
2.6.2 Améliorer la rejouabilité et/ou la durée de vie.....	10
2.6.3 Créer plus de détails .....	11
2.6.4 Optimiser l'utilisation matérielle .....	11
2.6.5 Simuler des interactions.....	12
<b>2.7 Utilisation dans l'industrie moderne.....</b>	<b>13</b>
2.7.1 Classification du contenu .....	13
2.7.2 Titres commerciaux incluant de la génération procédurale.....	14
2.7.2.1 Jusqu'en 2011 .....	14
2.7.2.2 Depuis 2011 .....	15
<b>2.8 Les possibilités d'approche techniques.....</b>	<b>18</b>
2.8.1 Constructive .....	18
2.8.2 Grammaire .....	19
2.8.3 Contrainte .....	20
2.8.4 Optimisation .....	21
<b>3. Définitions .....</b>	<b>23</b>
<b>3.1 Engageant.....</b>	<b>23</b>
3.1.1 Définition.....	23

3.1.2	Mesures .....	24
<b>3.2</b>	<b>Captivant.....</b>	<b>24</b>
3.2.1	Définition .....	24
3.2.2	Mesures .....	25
<b>3.3</b>	<b>Le designer.....</b>	<b>25</b>
3.3.1	Définition .....	25
3.3.2	Le processus créatif .....	25
<b>4.</b>	<b>Faire un jeu engageant et captivant.....</b>	<b>27</b>
<b>4.1</b>	<b>Le rythme.....</b>	<b>27</b>
4.1.1	L'apport de nouveautés.....	28
4.1.2	L'histoire du jeu .....	29
4.1.3	La difficulté .....	29
4.1.4	Les segments de gameplay .....	29
<b>4.2</b>	<b>La progression .....</b>	<b>30</b>
<b>4.3</b>	<b>Le défi .....</b>	<b>31</b>
4.3.1	Les IA Directrices .....	31
4.3.2	Avec la génération procédurale.....	31
<b>4.4</b>	<b>La taxonomie de Bartle.....</b>	<b>32</b>
4.4.1	Les carreaux .....	33
4.4.2	Les trèfles .....	33
4.4.3	Les cœurs.....	33
4.4.4	Les piques.....	33
4.4.5	Dans le cadre de ce travail .....	33
<b>5.</b>	<b>Travail pratique.....</b>	<b>34</b>
<b>5.1</b>	<b>Le choix du projet .....</b>	<b>34</b>
<b>5.2</b>	<b>Le choix de technologie .....</b>	<b>35</b>
5.2.1	Le moteur de jeu .....	35
5.2.2	Les packages utilisés .....	36
<b>6.</b>	<b>Le prototype.....</b>	<b>37</b>
<b>6.1</b>	<b>Le personnage du joueur .....</b>	<b>37</b>
6.1.1	Le script Player .....	38
6.1.1.1	Déplacements horizontaux .....	39
6.1.1.2	Gravité .....	40
6.1.1.3	Saut .....	41
6.1.1.4	Application des déplacements.....	41
6.1.1.5	Animations.....	42
6.1.1.6	Chutes .....	42
6.1.1.7	Pertes de contrôle .....	43
6.1.2	Le script CamControl .....	44
6.1.3	Le script PlayerHealth .....	44
<b>6.2</b>	<b>Les niveaux .....</b>	<b>45</b>

6.2.1	Structure .....	45
6.2.2	Les éléments composant les niveaux.....	46
6.2.2.1	Les plateformes.....	46
6.2.2.2	Les obstacles.....	46
6.2.2.3	Les ennemis .....	48
6.2.2.4	Les pièces .....	50
6.2.2.5	La condition de victoire.....	51
<b>6.3</b>	<b>L'analyse du joueur .....</b>	<b>51</b>
6.3.1	Le niveau témoin.....	51
6.3.2	Les points d'observation du joueur .....	52
6.3.3	La lecture de ces points .....	53
6.3.4	L'implémentation .....	53
<b>6.4</b>	<b>La génération de niveaux .....</b>	<b>57</b>
6.4.1	Placer les plateformes.....	57
6.4.2	Placer les obstacles, ennemis et pièces.....	60
6.4.3	Définir les paramètres de génération.....	62
6.4.3.1	Nombre de plateformes.....	62
6.4.3.2	Nombre d'obstacles et d'ennemis .....	63
6.4.3.3	Nombre de pièces .....	64
<b>7.</b>	<b>Le résultat .....</b>	<b>65</b>
7.1	Améliorations possibles.....	65
7.2	Phase de tests .....	66
7.2.1	La fiche de test.....	66
7.2.2	Les résultats.....	66
7.2.3	Mes observations .....	67
7.3	Conclusion du travail pratique.....	68
<b>8.</b>	<b>Réponse à la problématique.....</b>	<b>69</b>
8.1	Evolution de mon point de vue .....	69
8.2	Les forces de la génération procédurale.....	69
8.3	Les limites de la génération procédurale .....	70
8.4	Les options contre ces limites .....	71
<b>9.</b>	<b>Conclusion .....</b>	<b>72</b>
	<b>Bibliographie .....</b>	<b>73</b>
	<b>Annexe 1 : Fiche de Test .....</b>	<b>79</b>
	Instructions .....	79
	Votre profil .....	79
	Votre expérience avec les aventures procédurales de Beep-O :.....	80
	JOUEZ .....	81
	<b>Annexe 2 : Résultats des Tests .....</b>	<b>1</b>

## Liste des tableaux

Tableau 1 : Liste non exhaustive des jeux commerciaux sortis après 2011, avec une composante de génération procédurale .....	16
Tableau 2 : Les méthodes de génération de contenu orienté gameplay .....	22
Tableau 3 : Mesures du caractère engageant d'un jeu .....	24
Tableau 4 : Mesures du caractère captivant d'un jeu.....	25
Tableau 5 : Tableau d'évaluation des options pour le choix de moteur.....	35
Tableau 6 : Liste des packages utilisés pour la réalisation du prototype.....	36
Tableau 7 : Les différents points observés par le prototype.....	52

## Liste des figures

Figure 1 : L'ILLIAC I, l'ordinateur ayant produit la suite Illiac .....	4
Figure 2 : Graph de titres utilisant la génération procédurale.....	5
Figure 3 : Bruit blanc généré de façon aléatoire .....	5
Figure 4 : Bruit généré avec l'algorithme de Perlin .....	6
Figure 5 : Terrain généré en plaçant les points avec l'algorithme Fractional Brownian Motion .....	7
Figure 6 : Exemple d'arbre de comportement dans le moteur Unreal Engine 4 .....	8
Figure 7 : Le jeu Galactic Arms Race .....	9
Figure 8 : L'éditeur de SpeedTree .....	10
Figure 9 : Exemple de shader entièrement procédural, fait avec Blender .....	11
Figure 10 : Le jeu Elite, paru en 1980.....	12
Figure 11 : Illustration de l'inverse kinematics, utilisé ici pour adapter les pieds du personnage au terrain .....	13
Figure 12 : Les types de contenus d'un jeu qui peuvent être générés de façon procédurale .....	14
Figure 13 : Tableau de l'étude de Hendrikx et al. ....	15
Figure 14 : Graphique représentant la répartition des types de contenus créés procéduralement dans les jeux vidéo .....	17
Figure 15 : Exemple de l'approche constructive pour le placement de salles sur une grille. ....	19
Figure 16 : Exemple de règles de grammaire et d'application de ces règles pour générer du contenu .....	20
Figure 17 : Project Hastur, tower defense où les ennemis sont générés via évolution. ....	21
Figure 18 : Représentation des motivations intrinsèques et extrinsèques.....	24
Figure 19 : Approches du game design .....	26
Figure 20 : Courbe de tension dramatique .....	28
Figure 21 : Entrée du repaire de Kraid dans Super Metroid.....	29
Figure 22 : Une ferme dans le jeu Stardew Valley .....	30
Figure 23 : Courbe de difficulté par rapport aux compétences du joueur .....	31
Figure 24 : Taxonomie de Bartle .....	32
Figure 25 : Beep-O, le personnage incarné dans ce prototype de jeu .....	37
Figure 26 : Contrôles de base pour le personnage du prototype, sur manette Xbox....	38
Figure 27 : Script Player - Update (Partie 1) .....	39
Figure 28 : Script Player - InputCamMapping.....	39
Figure 29 : Script Player - Input.....	40
Figure 30 : Script Player - Gravity.....	40
Figure 31 : Script Player - Jump .....	41
Figure 32 : Script Player - Jump .....	41
Figure 33 : Script Player - Update (Partie 2) .....	42



Figure 34 : Script Player - LateUpdate.....	43
Figure 35 : Script Player - ControlLoss .....	43
Figure 36 : Script Player - HitKnockBack.....	43
Figure 37 : Les pivots de la caméra par rapport au joueur.....	44
Figure 38 : Script CamControl - Update.....	44
Figure 39 : Script CamControl – CamController .....	44
Figure 40 : Script PlayerHealth – TakeHit & Die .....	45
Figure 41 : Un niveau secret dans Mario Sunshine .....	45
Figure 42 : Les différentes plateformes .....	46
Figure 43 : Un obstacle .....	47
Figure 44 : Le compostantTrigger Sensor présent sur le joueur pour la détection de coups reçus.....	47
Figure 45 : Un ennemi.....	48
Figure 46 : Script EnnemyBehaviour - Update.....	49
Figure 47 : Script EnnemyBehaviour - Roaming.....	49
Figure 48 : Script EnnemyBehaviour - Pause.....	50
Figure 49 : Script EnnemyBehaviour - DetectsPlayer .....	50
Figure 50 : Pièce .....	50
Figure 51 : Drapeau de fin de niveau .....	51
Figure 52 : Vue d'ensemble du premier niveau de Beep-O .....	52
Figure 53 : Script LevelPerformance .....	54
Figure 54 : Script DataCollector - InitDataCollection.....	54
Figure 55 : Script DataCollector - AddLevel.....	54
Figure 56 : Script DataCollector – Properties.....	55
Figure 57 : Script PlayerGameplayData – vue Inspecteur .....	55
Figure 58 : Script DataCollector – Fonctions réponses.....	56
Figure 59 : Script DataCollector - FixedUpdate .....	56
Figure 60 : Script DataCollector – Fonction de contrôle.....	57
Figure 61 : Une règle de génération .....	58
Figure 62 : Script BuildingBlock.....	58
Figure 63 : Script BuldingGenerator – Generate (Partie 1) .....	59
Figure 64 : Script BuldingGenerator – Generate (Partie 2) .....	59
Figure 65 : Script BuldingGenerator - BiasedOffset.....	59
Figure 66 : Script BuldingGenerator - PlaceObstacles.....	60
Figure 67 : Script BuldingGenerator - PlaceEnemies.....	61
Figure 68 : Script BuldingGenerator – PlaceCoins .....	61
Figure 69 : Script LevelInfos.....	62
Figure 70 : Script BuldingGenerator – CalculatePlatformCount (Partie 1).....	62
Figure 71 : Script BuldingGenerator – CalculatePlatformCount (Partie 2).....	63
Figure 72 : Script BuldingGenerator – CalculatePlatformCount (Partie 3).....	63
Figure 73 : Début d'un niveau généré par l'algorithme .....	65

# 1. Introduction

Le jeu vidéo est aujourd'hui le plus important marché du divertissement au monde, surpassant l'industrie du cinéma et de la musique cumulés. (Richter, 2020)

Dans ce marché hautement compétitif, la création de contenu est souvent considérée comme étant le point d'étranglement dans le développement d'un jeu. Créer du contenu prend du temps, et les attentes des joueurs sur la quantité et la qualité n'ont cessé d'augmenter avec les années.

La génération procédurale, solution qui consiste à créer du contenu à partir d'un algorithme, semble être une réponse sensée à ce problème, mais ce contenu sera-t-il aussi captivant et engageant que du contenu créé par un designer ?

Cette question est un point critique de la recherche dans le domaine de la génération procédurale, domaine actif ces 35 dernières années.

Nous commencerons par établir un état de l'art, en étudiant les avancées de la génération de contenu procédurale, et son utilisation dans le domaine du jeu vidéo. Cela nous permettra d'identifier les défis que la conception d'un tel système apporte, ainsi que les solutions choisies par les développeurs de différents titres commerciaux incorporants de la génération procédurale.

Ensuite, nous chercherons à apporter une définition aux termes employés dans le sujet, ce afin d'orienter notre recherche de façon plus précise, ainsi que d'apporter un premier élément de réponse.

Enfin, j'essaierai d'apporter une réponse à la problématique, en réalisant un projet ayant pour objectif de créer du contenu engageant et captivant.

## 2. La génération de contenu procédurale

### 2.1 Définition

La génération procédurale est un terme englobant un grand nombre de techniques, réparties dans de nombreux domaines de natures très différentes. Je vais chercher dans cette section à donner une définition que j'utiliserai tout au long de l'étude.

Togelius, Kastbjerg, Schedl et Yannakakis (chercheurs à l'IT University of Copenhagen) tentent lors d'un travail de recherche en 2011 de définir la génération procédurale, et arrivent à la conclusion (2011, p. 7) :

*« We can therefore tentatively redefine PCG as the algorithmical creation of game content with limited or indirect user input. »*

« User » faisant ici référence au joueur, ou à un designer (Togelius et al., 2011). « PCG » est l'acronyme de Procedural Content Generation, traduit par génération de contenu procédurale.

Togelius qualifiera plus tard cette même définition, avec Shaker (qui a participé indirectement au travail de 2011) et Nelson (également chercheur de l'IT University of Copenhagen) (2016, p. 2) :

*« Like all definitions (except perhaps those in mathematics), our definition of PCG is somewhat arbitrary and rather fuzzy around the edges. »*

Ainsi, la définition du concept de génération procédurale est encore imprécise, entre autres par la grande quantité de domaines concernés. On peut cependant accepter celle proposée en 2011 comme applicable dans la grande majorité des cas, et j'utiliserai celle-ci dans le reste de cette étude.

### 2.2 Génération de contenu

Cette définition indique que la génération de contenu procédural est une méthode de création de contenu via un algorithme, avec un certain degré d'intervention de la part d'un utilisateur, joueur ou designer.

Cette intervention prendra généralement la forme d'ensembles de données ou d'assets<sup>1</sup> dans le cas d'un designer, ou de commandes de la part du joueur (Togelius et al., 2016).

L'algorithme prendra alors ces données, assets ou commandes pour générer du contenu avec. La nature du contenu peut varier, permettant par exemple la création de niveaux,

---

<sup>1</sup> Une ressource numérique dans un jeu vidéo

modèles 3D, images 2D, sons, musiques ou encore d'histoires et de scénarios (Togelius et al., 2016).

Par exemple, SpeedTree est un middleware<sup>2</sup> permettant la création d'arbres détaillés de façon procédurale. Il est employé dans un vaste catalogue de jeux commerciaux, incluant des titres majeurs comme Uncharted 4 ou Far Cry 4 & 5 (Interactive Data Visualization, 2017). Les arbres sont créés avec une combinaison de génération de meshes<sup>3</sup> par SpeedTree à partir de paramètres entrés par l'utilisateur, et l'import de textures prêtes pour les feuilles ou l'écorce. SpeedTree offre des outils pour modifier l'arbre comme on le souhaite, en changeant par exemple la densité du feuillage ou le niveau de séparation des branches à l'aide de paramètres définis par l'utilisateur (SpeedTree, 2012).

## **2.3 Un peu d'histoire**

### **2.3.1 Avant l'informatique**

La génération procédurale n'est pas quelque chose de si récent que l'on pourrait penser.

Par exemple, le compositeur Johann Kirnberger créa en 1757 *Der allezeit fertige Menuetten- und Polonaisencomponist* (traduisible par "Le menuet et compositeur polonais toujours prêt"), une façon de faire de la musique en assemblant différents éléments précomposés en lançant des dés (Nierhaus, 2009).

### **2.3.2 Le premier contenu généré par ordinateur**

Le premier cas de contenu généré par un ordinateur est accepté comme étant la Suite Illiac, une suite musicale pour quartet de cordes, qui est créée par l'ordinateur ILLIAC I via un algorithme écrit par les professeurs Lejaren Hiller et Leonard Issacson en 1957 (Hiller et al., 1959).

---

<sup>2</sup> Logiciel tiers liant différentes applications, en apportant des modifications si nécessaire.

<sup>3</sup> Ensemble de points, d'arêtes et de faces constituant un modèle 3D

Figure 1 : L'ILLIAC I, l'ordinateur ayant produit la suite Illiac



(Hiller, 1957)

### 2.3.3 Dans le domaine du jeu vidéo

Dans le domaine du jeu vidéo, les pionniers de la génération procédurale sont généralement admis comme étant *Beneath Apple Manor*, sortis en 1978, et *Rogue*, en 1980. Ces jeux inspirés de *Dungeons & Dragons* (Gygax et Arneson, 1974) créent des donjons en ASCII<sup>4</sup> en plaçant des salles, portes, couloirs, monstres ou trésors.

La popularité de *Rogue* sera telle que le jeu donnera son nom à un genre, le roguelike, qui regroupe les jeux au tour par tour où le joueur traverse des environnements aléatoirement générés. Le terme qui sera affiné lors de l'International Roguelike Development Conference 2008, donnant lieu à la controversée Berlin Interpretation (RogueBasin, 2013).

### 2.3.4 La génération procédurale aujourd'hui

Aujourd'hui, le terme roguelike sera plus généralement employé pour parler des jeux intégrant un élément de mort permanente, ainsi que des niveaux générés de façon procédurale (Brown, 2019). Ce genre englobe des jeux tels que *The Binding of Isaac* (MCMILLEN, Edmund et HIMSL, Florent, 2011) ou *FTL* (Subset Games, 2012).

Pour ce qui est de l'utilisation moderne, de nombreux jeux utilisent des outils basés sur la génération procédurale pour de la création de contenu, que ce soit lorsque le jeu est

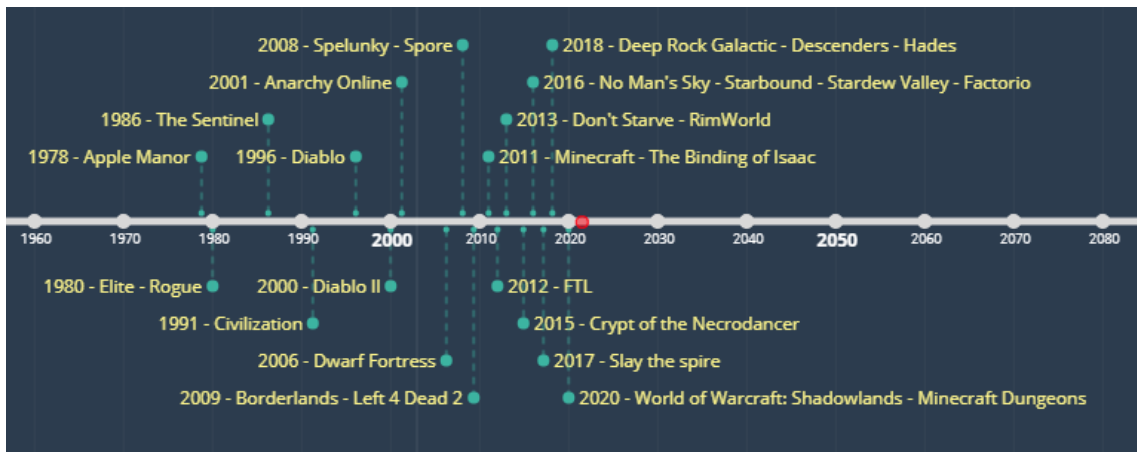
---

<sup>4</sup> Standard d'encodage de caractères, ici faisant référence à l'art ASCII, qui consiste à utiliser des caractères de texte pour créer des éléments visuels (tel que des personnages ou cartes de donjons)

lancé, en générant des niveaux par exemple, ou alors comme outil pour accélérer les différents processus créatifs intervenant dans le développement d'un jeu vidéo.

Afin d'avoir un aperçu de cet historique dans le jeu vidéo, j'ai réalisé ce graph représentant certains titres utilisant la génération procédurale comme une partie importante de l'expérience, de 1978 à 2021.

Figure 2 : Graph de titres utilisant la génération procédurale



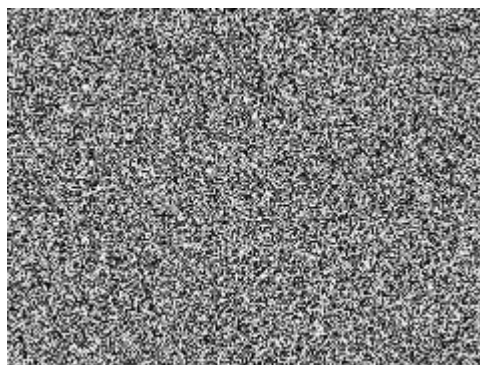
Fait à l'aide de l'outil : Time Graphics

## 2.4 Génération « aléatoire »

La génération procédurale est fréquemment associée à l'aléatoire. Cependant, même si de nombreux algorithmes utilisent des variables choisies au hasard dans un ensemble de données pour créer du contenu très varié, ils ne sont pas pour autant entièrement aléatoires.

Placer des points au hasard pour créer un modèle résulte le plus souvent en un amas de points sans intérêt particulier, pas en un monde que l'on a envie d'explorer, ou un personnage avec qui on souhaite converser.

Figure 3 : Bruit blanc généré de façon aléatoire



(Stolfi, 2020)

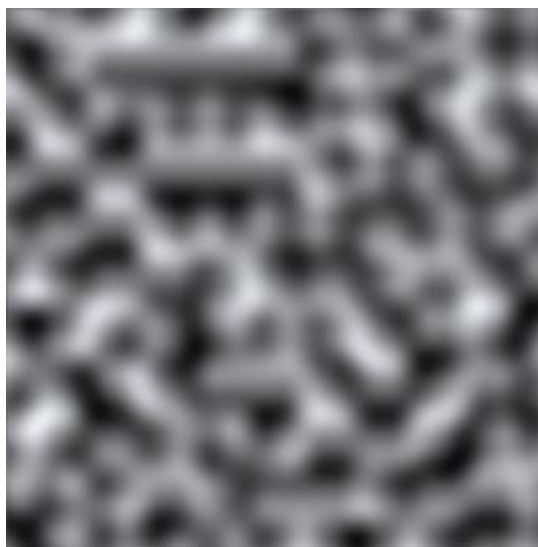
Cet aspect “inintéressant pour l’être humain” du purement aléatoire peut s’expliquer par la prédominance de patterns dans la nature, où des éléments similaires auront tendance à se regrouper et former des clusters ou des patterns, comme sur les feuilles d’un arbre, ou la structure des ruches.

En tant qu’êtres humains, cette répétition de patterns et regroupements d’éléments similaires nous paraît alors plus naturelle, car elle correspond aux environnements auxquels nous sommes habitués.

La création de structure et patterns est donc un élément important de la génération procédurale, le but étant ultimement de créer du contenu qui sera apprécié d’autres êtres humains.

Les algorithmes de génération procédurale ont donc pour but de créer un ensemble de données qui répond à une certaine logique ou structure. Par exemple, l’algorithme de Perlin permet de créer du bruit structuré.

Figure 4 : Bruit généré avec l’algorithme de Perlin

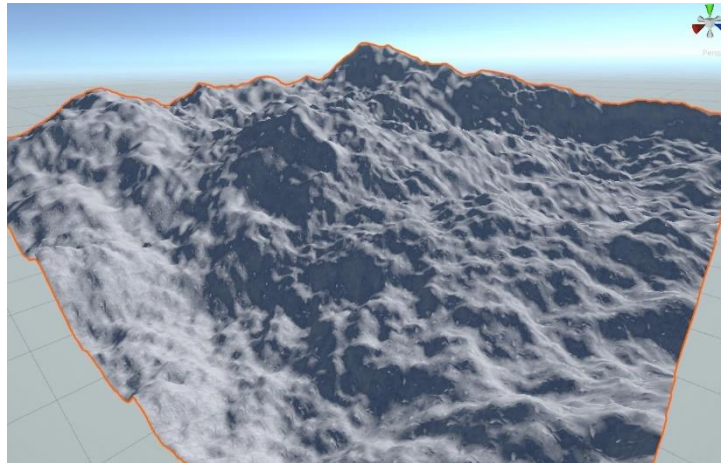


(Unity, 2021)

Le fait de regrouper ces points permet de générer des ensembles plus cohérents, qui paraîtront plus naturels, et donc plus plaisants visuellement.

On peut ensuite, par exemple, combiner les résultats obtenus avec différents ensembles générés pour obtenir des résultats plus détaillés. L’algorithme Fractional Brownian Motion s’appuie sur celui de Perlin pour créer du bruit plus détaillé, dont chaque niveau de détail est appelé “octave”.

Figure 5 : Terrain généré en plaçant les points avec l'algorithme Fractional Brownian Motion



(Unity, 2021)

De plus, le caractère aléatoire n'est pas nécessaire pour de la génération procédurale (Togelius et al., 2011). Certains algorithmes tels que le bruit de Perlin ont des résultats déterminables mathématiquement, du moment que l'on connaît les données en entrée.

L'introduction d'un élément aléatoire, le plus souvent le choix des données numériques en entrée du système de génération procédurale, servira le plus souvent à donner un degré de variation dans les résultats obtenus.

## **2.5 L'intelligence artificielle dans la pcg**

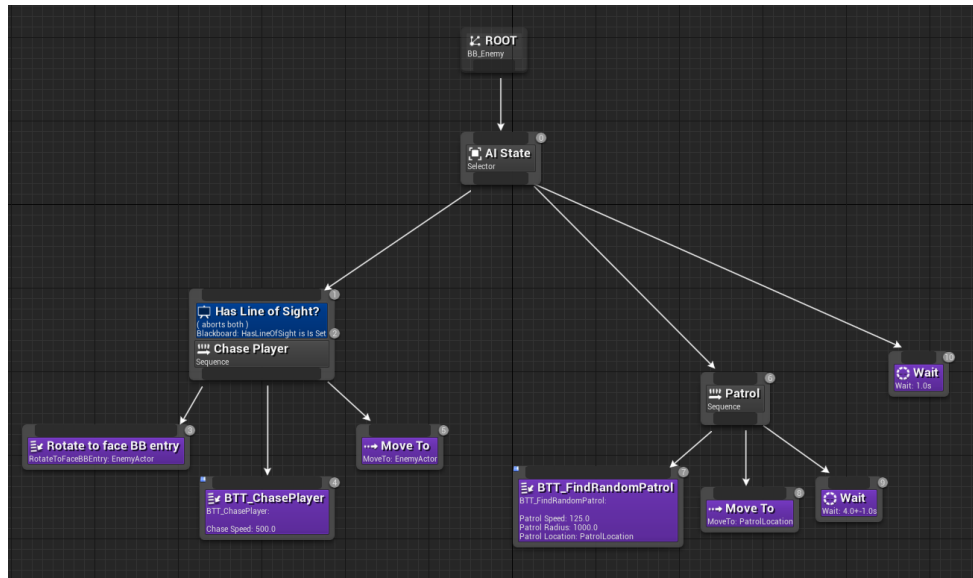
### **2.5.1 L'IA dans le jeu vidéo**

L'intelligence artificielle dont nous parlons ici correspond à une intelligence simple, programmée pour exécuter une tâche spécifique. Ce type d'intelligence artificielle est appelée IA faible, ou IA étroite.

Dans le jeu vidéo, le terme désigne principalement trois classes d'IA différentes : les IA basées sur le paterne d'état-transition, celles basées sur les arbres de comportement, et celles étant construites sur le principe de plan d'action axé sur les objectifs, de l'anglais goal oriented action planning (ou GOAP).



Figure 6 : Exemple d'arbre de comportement dans le moteur Unreal Engine 4



(Epic Games, 2021)

### 2.5.2 La création de contenu

Ces classes d'intelligence artificielle sont surtout employées afin de créer du comportement pour les personnages non joueurs (ou PNJ), mais il arrive que ces principes algorithmiques soient employés pour créer du contenu en cours de jeu.

L'exemple le plus parlant est celui des directeurs, des algorithmes chargés d'adapter différents éléments pendant le jeu. L'IA directrice de Left 4 Dead (jeu de tir où les joueurs affrontent des hordes de morts-vivants) est l'exemple le plus connu : son rôle est d'assurer un rythme de jeu idéal, en altérant l'apparition des hordes d'ennemis ou d'objets à récupérer selon un degré d'intensité mesuré à l'aide de l'interaction des joueurs avec leur environnement, comme le nombre de coups en mêlée reçus ou donnés, les munitions ou la vie restante, ou encore le temps passé depuis la dernière horde (Thompson, 2020).

### 2.5.3 Le Machine Learning et le jeu vidéo

Aujourd'hui, le développement des intelligences artificielles dites fortes basées sur des principes d'apprentissage machine est un champ de recherche actif. Un titre commercial existe, Galactic Arms Race, jeu développé par Evolutionary games et sorti en 2010 qui génère via des réseaux de neurones des armes par rapport aux préférences du joueur en analysant ses données de jeu (Hastings et al., 2009).

Figure 7 : Le jeu Galactic Arms Race



(Evolutionary Games, 2021)

Des études cherchent également à développer des niveaux pour plateformer à l'aide d'algorithmes d'apprentissage machine, telle que l'étude de l'université de Santa Cruz, qui propose une méthode pour générer des niveaux de Super Mario Bros. basée sur le principe de l'arbre de recherche de Monte Carlo (Summerville et al., 2015).

L'apprentissage machine ne fait cependant pas partie du cadre de cette étude, étant un champ de recherche à part, et ainsi je ne développerais pas ma réponse autour de ce sujet. Cependant, étant donné que cette technologie se développe rapidement dans d'autres domaines que le jeu vidéo, il est possible d'imaginer que l'apprentissage machine prenne une place importante dans le domaine de la génération procédurale de contenu à l'avenir.

## 2.6 Les raisons pour utiliser la génération procédurale

Il existe différentes raisons pour lesquelles un studio peut choisir d'inclure de la génération procédurale dans un titre. Dans cette étude, nous en verrons cinq :

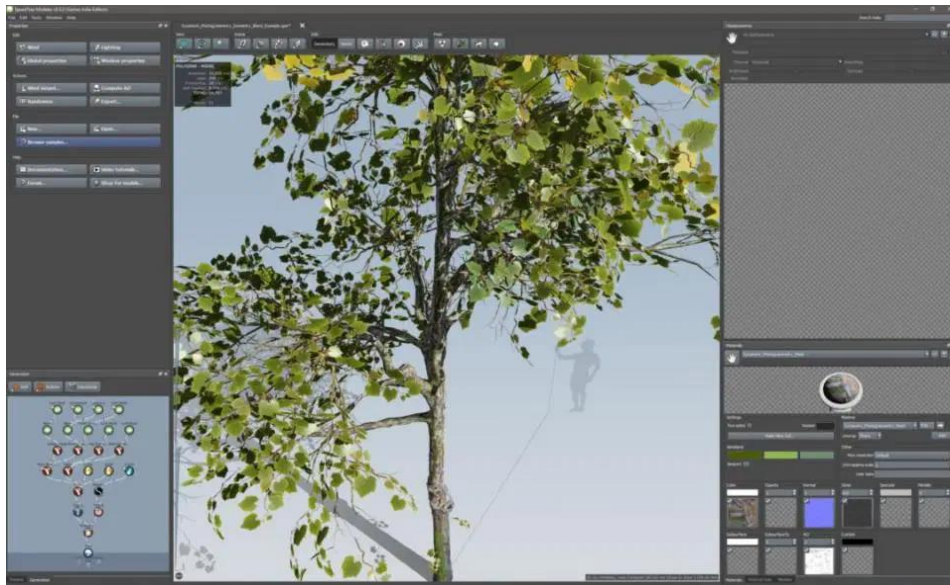
- Accélérer le développement
- Améliorer la rejouabilité et/ou augmenter la durée de vie
- Créer plus de détails
- Optimiser l'utilisation matérielle
- Créer de l'interaction.

D'autres raisons peuvent également amener à l'utilisation de génération procédurale, mais nous nous concentrerons sur celles-ci.

### 2.6.1 Accélérer le développement

La génération procédurale est souvent employée pour accélérer le processus de développement d'un jeu. En reprenant l'exemple de SpeedTree, modéliser un arbre prend plusieurs heures à un artiste, tandis que le middleware permet de le faire beaucoup plus rapidement, en plus de permettre de créer des déclinaisons en quelques clics (SpeedTree, 2012).

Figure 8 : L'éditeur de SpeedTree



(SpeedTree, 2021)

Il faut cependant noter que développer la solution de génération procédurale en elle-même prend un temps non négligeable, aussi une évaluation de la nécessité de cette solution sera primordiale avant de se lancer dans son développement.

### 2.6.2 Améliorer la rejouabilité et/ou la durée de vie

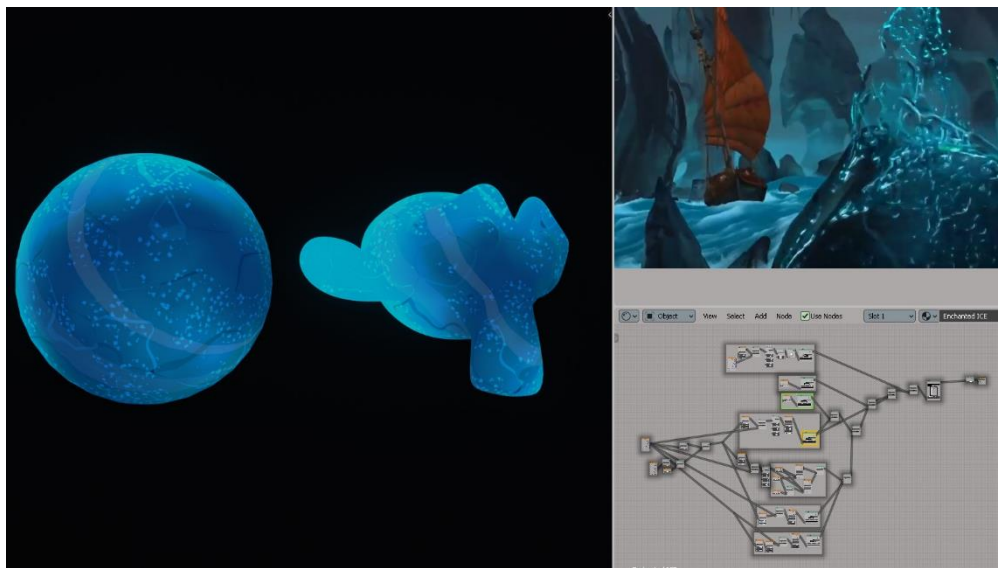
Inclure des éléments générés de façon procédurale permet d'accroître (parfois considérablement) la durée de vie et la rejouabilité d'un titre. Le genre des roguelikes est un exemple parlant de titres s'appuyant sur la génération procédurale pour ajouter de la rejouabilité, en créant des agencements de niveaux et d'évènements très différents d'une partie à l'autre. Le joueur est donc amené à comprendre le jeu, avec ses règles et ses mécaniques, au lieu d'apprendre par cœur les niveaux (Brown, 2019).

D'autres jeux que les roguelikes l'utilisent également pour la rejouabilité. XCOM : Enemy Unknown et XCOM 2 créent leurs cartes et leurs missions de façon procédurale, assurant une expérience fraîche pour chaque partie, en plus de permettre de présenter au joueur une importante diversité de défis tactiques (Hess, 2018).

### 2.6.3 Créer plus de détails

La génération procédurale peut être employée pour créer beaucoup de détails en peu de temps. La génération de texture est très utilisée notamment dans le développement de shaders, morceaux de code définissant la façon dont un objet 3D sera rendu dans le moteur d'un jeu. Ces shaders peuvent créer des détails de façon algorithmique, permettant un niveau de détails et de déclinaisons possibles qui nécessiteraient un temps bien plus important pour un artiste.

Figure 9 : Exemple de shader entièrement procédural, fait avec Blender



(Schiller, 2020)

Autre exemple, les effets de particules permettent de créer de façon procédurale des éléments tels que des flammes ou de la fumée détaillées, en plus de les animer, chose qui prendrait beaucoup plus de temps à faire à la main à partir de textures et modèles 3D.

### 2.6.4 Optimiser l'utilisation matérielle

L'optimisation matérielle peut être un facteur poussant à l'utilisation de génération procédurale. Le jeu Elite, paru en 1980, générait son univers avec 8 galaxies, chacune composée de 256 planètes à explorer. Une telle échelle n'était possible qu'avec de la génération procédurale : le jeu crée ses planètes en passant un nombre (seed) un certain nombre de fois dans un algorithme, qui génère les données d'une planète sous la forme d'un ensemble de nombres (Spufford, 2003). Cet ensemble étant trouvable avec une seed et un algorithme fixe, il est possible de retrouver les informations de 2048 planètes différentes et uniques, à un coût très faible en mémoire.

Figure 10 : Le jeu Elite, paru en 1980



(Harte, 2007)

Un autre exemple, le jeu de tir à la première personne .kkrieger est un jeu ayant un exécutable de 96 kilo-octets, qui génère l'équivalent de centaines de mégaoctets de textures et de modèles 3D une fois lancé (Giesen, 2012).

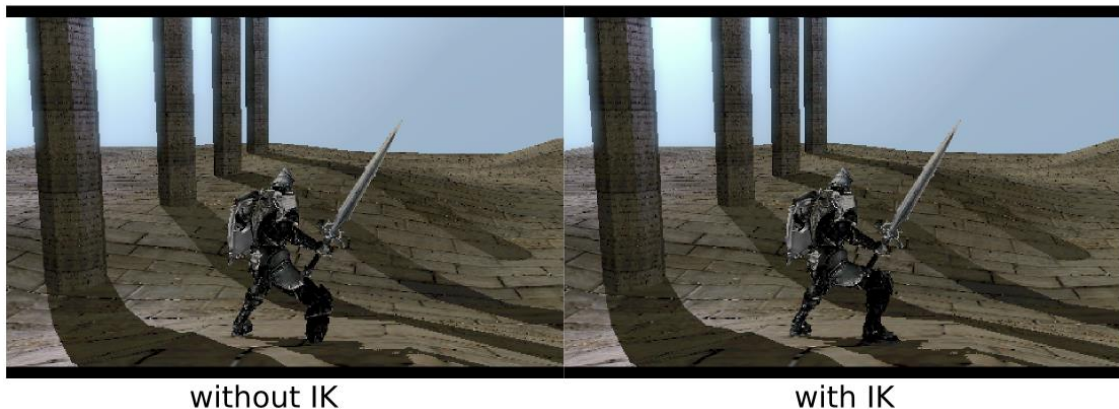
Plus récemment, le MMO EVE Online, ou le jeu No Man's Sky génèrent aussi leurs univers respectifs de façon procédurale, permettant de proposer aux joueurs un ensemble de plusieurs milliards de mondes explorables, sans avoir à conserver chacun de ces mondes sur un serveur en tout temps (Peckham, Date inconnue).

### 2.6.5 Simuler des interactions

La génération procédurale est aussi un moyen d'apporter de l'interaction entre différents éléments du jeu. L'animation procédurale est en grande partie dédiée à cet effet, avec par exemple l'Inverse Kinematics, qui peut permettre d'aligner correctement le bras d'un personnage avec un objet à saisir, sans avoir à prévoir tous les cas lorsque le personnage est animé (Rosen, 2014).

Un animateur réalisera une animation générique, et cette animation sera modifiée de façon procédurale pour s'adapter aux objets environnants. On peut également faire réagir des objets ou personnages à des forces physiques à l'aide de code, pour animer par exemple des cordes ou du tissu (Mach, 2017).

Figure 11 : Illustration de l'inverse kinematics, utilisé ici pour adapter les pieds du personnage au terrain



(Yeung, 2012)

L'audio sera également traité de façon procédurale dans un souci d'interaction avec l'environnement, en ajoutant par exemple de la réverbération dans une grotte, ou en modifiant l'atténuation des différentes fréquences d'un son pour ajouter une impression de distance (Riot Games, 2018). Le faire de façon procédurale permet de créer beaucoup de variations auditives à partir d'un seul clip audio.

## 2.7 Utilisation dans l'industrie moderne

La génération procédurale est donc applicable dans une grande variété de domaines. Nous nous appuyerons sur une étude réalisée en 2011 par Hendrikx et al. pour classifier l'utilisation de génération procédurale dans les jeux commerciaux récents.

Cette étude inclut des titres parus de 1980 à 2011, et la dernière décennie étant riche en titres utilisant différentes méthodes de génération de contenu, nous compléterons le tableau proposé par Hendrikx et al. avec les titres récents présentant une composante de génération procédurale importante.

### 2.7.1 Classification du contenu

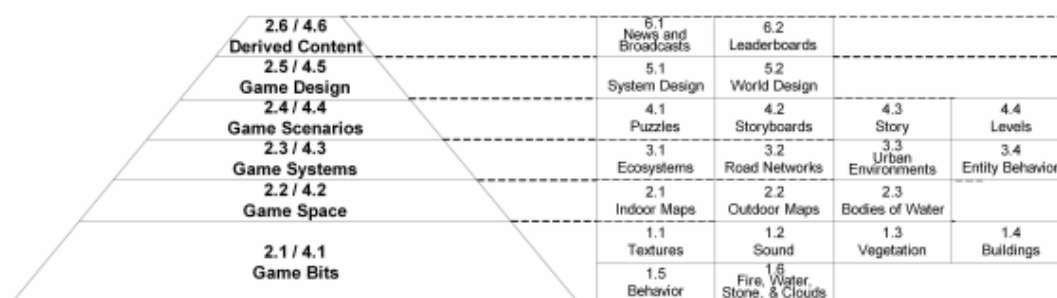
Cette classification se base sur une autre : la classification du contenu d'un jeu. Hendrikx et al. (2011) classent donc le contenu de la façon suivante (traduit de l'anglais) :

- Morceaux de jeu
  - Inclus les textures, sons, la végétation, les bâtiments, les comportements ou encore le feu, l'eau, la pierre ou les nuages
- Espace de jeu
  - Inclus les cartes d'intérieur, les cartes d'extérieur ou les corps d'eau
- Systèmes de jeu



- Inclus les écosystèmes, réseaux routiers, environnements urbains ou le comportement d'entités
- Scénarios de jeu
  - Inclus les puzzles, storyboards, histoires et niveaux
- Game design
  - Inclus le design des systèmes et le design du monde (les habitants, relations, histoire...)
- Contenu dérivé
  - Inclus les nouvelles, broadcasts ou tableaux de scores

Figure 12 : Les types de contenus d'un jeu qui peuvent être générés de façon procédurale



(Hendrikx et al., 2011)

## 2.7.2 Titres commerciaux incluant de la génération procédurale

### 2.7.2.1 Jusqu'en 2011

Le tableau donné dans l'étude de Hendrikx et al. classe des jeux allant de 1984 à 2011 qui soit ont été inclus dans la sélection Vintage Games, on gagné une récompense de l'industrie, soit sont sortis récemment (au moment de l'étude) et sont devenus populaires.

Le tableau n'inclut pas de colonne pour le Game Design ou le contenu dérivé, l'étude n'ayant pas trouvé de titres commerciaux générant ces types de contenu procéduralement.

Figure 13 : Tableau de l'étude de Hendrikx et al.

Games (with year of release)	Game Bits	Game space	Game Systems	Game Scenarios
Borderlands (2009)	x			
Diablo I (2000)		x		
Diablo II (2008)		x		x
Dwarf Fortress (2006)		x	x	x
Elder Scrolls IV: Oblivion (2007)	x			
Elder Scrolls V: Skyrim (2011)				x
Elite (1984)		x	x	x
EVE Online (2003)	x	x		x
Facade (2005)				x
FreeCiv and Civilization IV (2004)		x		
Fuel (2009)		x		
Gears of War 2 (2008)	x			
Left4Dead (2008)				x
.kkrieger (2004)	x			
Minecraft (2009)		x	x	
Noctis (2002)		x		
RoboBlitz (2006)	x			
Realm of the Mad God (2010)	x			
Rogue (1980)		x		x
Spelunky (2008)	x	x		x
Spore (2008)	x	x		
Torchlight (2009)		x		
X-Com: UFO Defense (1994)		x		

(Hendrikx et al., 2011)

### 2.7.2.2 Depuis 2011

J'ai donc repris ce tableau, en incluant les titres majeurs depuis 2011.

J'ai pour cela choisi les titres selon trois critères principaux :

- Le jeu propose quelque chose d'unique ou jamais fait auparavant dans le domaine.
- Le jeu est un succès critique et commercial intégrant une importante composante de génération procédurale dans son développement et/ou dans le jeu en lui-même.
- Le jeu est sorti sur PC ou console (la liste ne traite pas des jeux mobiles.)

Cette liste n'est pas exhaustive, mais offre une représentation de l'état de la génération procédurale dans l'industrie du jeu vidéo aujourd'hui.



Tableau 1 : Liste non exhaustive des jeux commerciaux sortis après 2011, avec une composante de génération procédurale

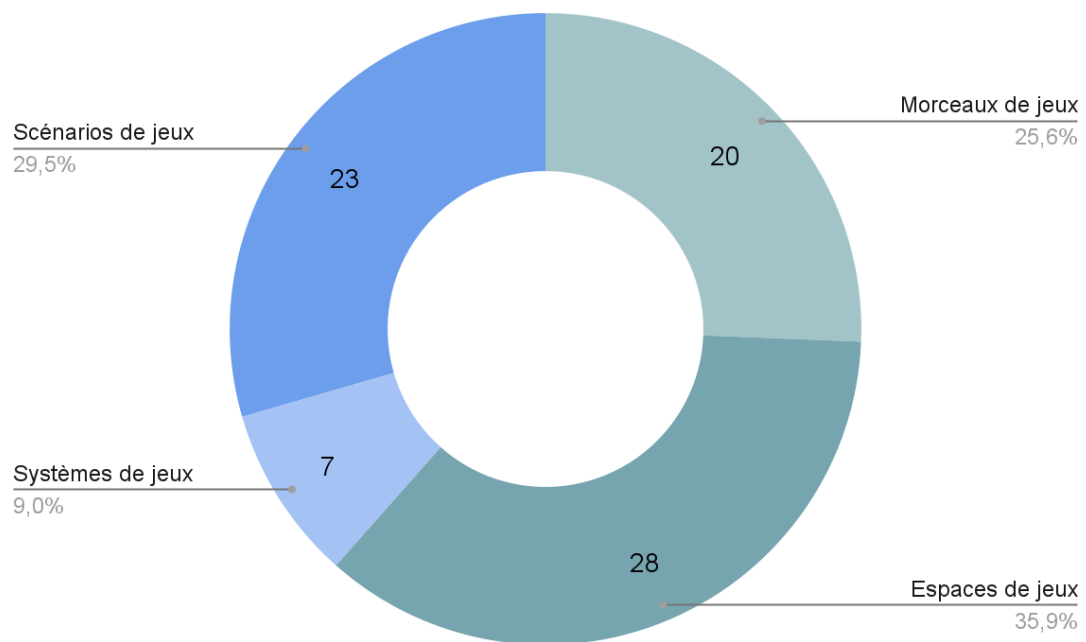
Jeux (avec l'année de sortie)	Morceaux de jeux	Espaces de jeux	Systèmes de jeux	Scénarios de jeux
Bad North (2018)		x		x
Borderlands 2 (2012)	x			
Borderlands 3 (2019)	x			
Caves of Qud (2015)				x
Civilization VI (2016)		x		
Cyberpunk 2077 (2020)	x			
Diablo III (2014)		x		x
Dead Cells (2017)		x		
Enter the Gungeon (2016)		x		x
FRACT OSC (2014)	x			
Far Cry 4 (2014)	x			
Far Cry 5 (2018)	x			
FTL : Faster than light (2012)	x	x		x
Grand Theft Auto V (2013)	x			
Hades (2018)		x		x
Middle Earth : Shadow of Mordor (2014)			x	x
Middle Earth : Shadow of War (2017)			x	x
Moon Hunters (2016)		x		x
No Man's Sky (2016)	x	x	x	x
Rimworld (2016)		x	x	x
The Binding of Isaac (2011)	x	x		x
Townscaper (2020)		x		
Uncharted 4 (2016)	x			
XCOM : Enemy Unknown (2012)		x		x
XCOM 2 (2016)		x		x

Tout comme pour l'étude originale, aucun titre commercial intégrant du game design procédural n'a été trouvé. Cependant, quelques expériences académiques existent, telles que Variations Forever qui génère des jeux d'arcade, ou encore Angelina, qui crée

des jeux complets, incluant mécaniques, niveaux et choisit les éléments d'art (Smith et al., 2015).

À partir de ce tableau, nous pouvons regrouper sous un graphique les titres selon leurs composantes de génération procédurale, afin d'avoir un aperçu de la répartition du type d'utilisation faite de ces techniques. Ce graphique prend en compte les observations de Hendrikx et al. ainsi que les miennes.

Figure 14 : Graphique représentant la répartition des types de contenus créés procéduralement dans les jeux vidéo



On peut constater que la génération procédurale est surtout utilisée dans les espaces de jeux, alors que relativement peu de jeux commerciaux génèrent des systèmes de jeux.

On peut également remarquer que la plupart des jeux AAA<sup>5</sup> intégrant de la génération procédurale le font sur des morceaux de jeu, principalement pour l'animation, comme la réponse à des forces physiques pour les personnages de Uncharted 4 (Mach, 2017) ou la synchronisation labiale<sup>6</sup> dans Cyberpunk 2077 (Cyberpunk 2077, 2020).

<sup>5</sup> Les jeux AAA sont les titres disposant d'un budget important, c'est une désignation similaire au terme "blockbuster" pour l'industrie du cinéma.

<sup>6</sup> La synchronisation entre le discours oral d'un personnage 3D et l'animation de son visage, en particulier ses lèvres.

La génération procédurale est donc aujourd'hui principalement employée pour générer des espaces, morceaux et scénarios de jeu.

## **2.8 Les possibilités d'approche techniques**

Pour la génération de contenu procédural, de nombreuses approches techniques sont possibles.

Dans leur étude, Hendrikx et al. (2011) donnent une classification des types d'algorithmes utilisés pour la génération procédurale. Nous en utiliserons cependant une autre : lors d'une Game Developer Conference de 2015, Smith et al. présentent une classification pour les approches de la génération procédurale, concentrée autour de l'approche et non de l'algorithmique concrète.

Ces classes d'approches sont au nombre de quatre :

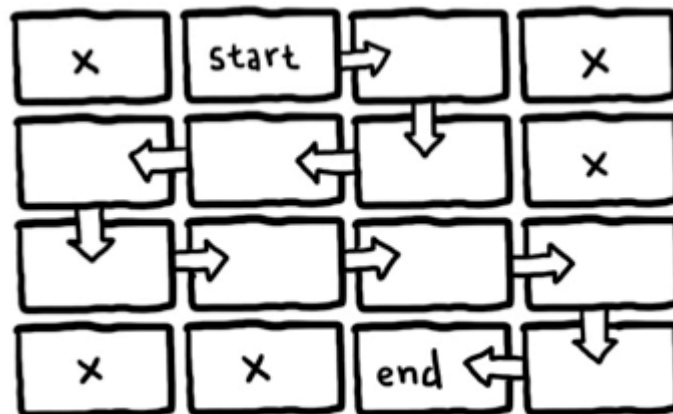
- Constructive
- Basé sur la grammaire
- Basé sur la contrainte
- Basé sur l'optimisation

### **2.8.1 Constructive**

Cette approche (également connue sous l'appellation "Additive"(Compton, 2015)) consiste à utiliser des blocs de construction créés par des designers afin de générer du contenu (Smith et al., 2015). C'est une approche très commune pour les roguelikes : par exemple, The Binding of Isaac utilise cette approche pour générer les étages de donjons à partir de salles existantes, et Spelunky génère des salles à partir de cases créées à la main par Derek Yu, le créateur du jeu (Yu, 2008).

Smith et al. (2015) décrivent l'approche comme étant plutôt légère algorithmiquement, la charge étant surtout mise sur les designers et artistes pour créer du contenu modulaire. Ils évoquent également le fait que cette approche est spécifique au design du jeu, ce qui est à la fois une bonne et une mauvaise chose (le système est adapté au jeu en question, mais peu de choses peuvent être conservées entre projets ou rendues génériques).

Figure 15 : Exemple de l'approche constructive pour le placement de salles sur une grille.



(Bidarra, 2016)

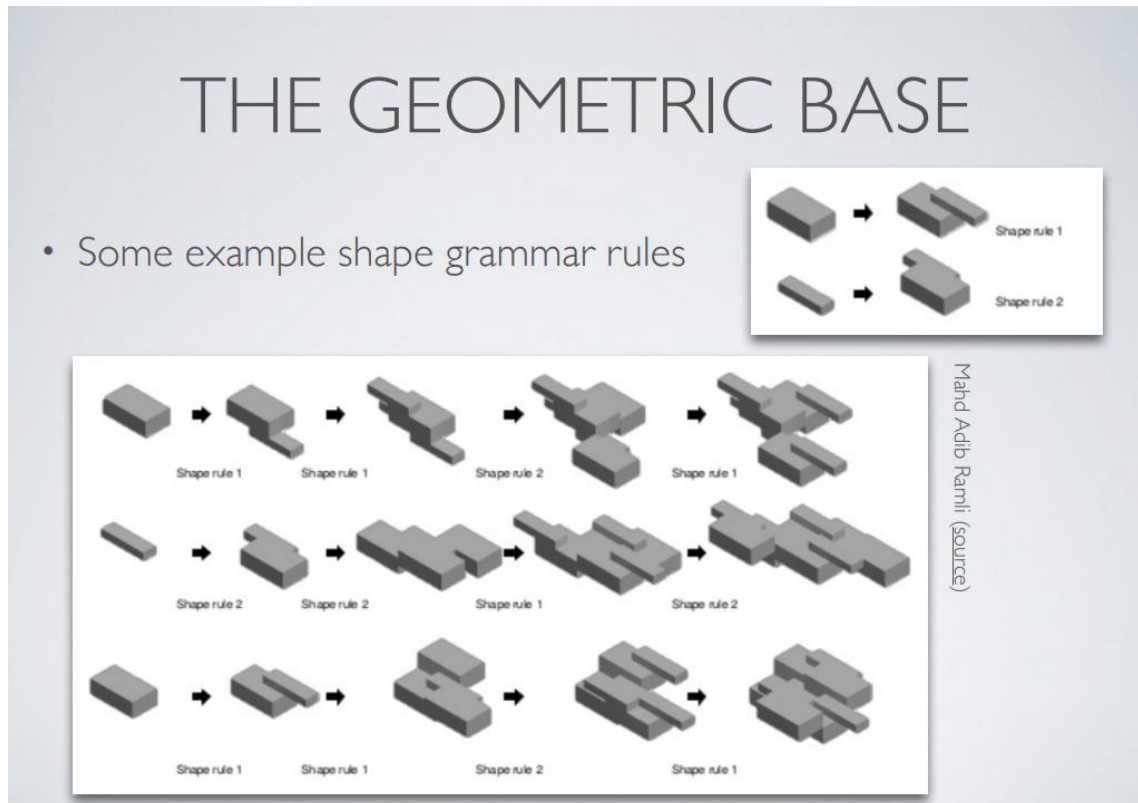
Smith et al. (2015) donnent comme exemple le générateur de donjons d'Advanced Dungeons & Dragons, qui permet de construire des donjons en jetant des dés avec des tables de correspondances pour placer des salles et des rencontres d'ennemis.

### 2.8.2 Grammaire

Cette approche consiste à définir un ensemble d'objets pouvant exister dans le monde, ainsi qu'un ensemble de règles de production qui définit quel objet sera transformé, en quoi il le sera, et à quel moment de l'exécution du processus cette transformation a lieu (Smith et al., 2015). Le résultat sera un graph construit en prenant en compte les étapes précédentes et des règles définies par les designers (Dormans, 2011).

Cette approche est particulièrement commune pour générer de la végétation, mais peut également être utilisée pour générer des niveaux. Becky Lavender est l'auteur de The Zelda Dungeon Generator, qui utilise cette approche pour générer des donjons inspirés de la licence The Legend of Zelda (Lavender, 2015).

Figure 16 : Exemple de règles de grammaire et d'application de ces règles pour générer du contenu



(Hwang, 2017)

Smith et al. décrivent l'approche comme facilitant l'édition de morceaux de contenus, en plus de permettre de générer des structures surprenamment complexes. Ils pointent cependant une tendance à la répétitivité, qui peut être corrigée par des procédures de test intervenant après la génération pour s'assurer à l'aide de paramètres mesurables que le résultat à l'air beau ou se joue correctement (Smith et al., 2015).

### 2.8.3 Contrainte

Cette approche consiste à définir un ensemble de propriétés et de contraintes numériques et/ou logiques avec lesquelles générer du contenu, en passant par un solveur. C'est un concept très utilisé dans le développement d'outils à l'attention des designers.

Un exemple de jeu poussant le concept particulièrement loin est le jeu Variations Forever, qui permet à l'aide d'ensembles de règles logiques et modulaires de générer des jeux d'arcades.

Smith et al. décrivent l'approche comme ayant pour principal défi la définition du domaine, ainsi qu'un processus de debugging difficile. La force sera la flexibilité qu'offre

l'approche, en plus d'avoir la possibilité de forcer le système à faire des promesses fortes sur de potentiels problèmes de design.

#### 2.8.4 Optimisation

Aussi connue sous le terme “Recherche”, cette approche se base sur le principe d'évolution : en partant d'un ensemble de contenu, on trie en conservant le contenu considéré comme “bon” (déterminé à l'aide d'une “fitness function”, qui évalue la qualité du contenu à l'aide de contraintes et mesures prédéfinies), que l'on fera évoluer soit en croisant différents éléments entre eux, soit en créant des mutations (en modifiant par exemple des paramètres numériques au contenu) (Smith et al., 2015).

Le jeu de plateau Yavalath est un exemple de jeu créé de cette manière, à partir d'un système faisant évoluer des jeux de plateau (Smith et al., 2015). Aussi, le jeu Project Hastur est un tower defense faisant évoluer des créatures afin de contrecarrer les défenses du joueur.

Figure 17 : Project Hastur, tower defense où les ennemis sont générés via évolution.



(Polymorphic Games, 2019)

Smith et al. décrivent l'approche comme prenant du temps, et la définition de la fitness function peut être difficile. L'approche peut également donner des solutions inattendues, ce qui peut être une bonne ou une mauvaise chose. C'est cependant une approche très générale et qui requiert peu de connaissances sur le domaine, ce qui la rend très adaptable (Smith et al., 2015).

Pour clore cette partie, voici un tableau récapitulatif des méthodes de génération de contenu orienté gameplay proposé lors de la conférence, avec une description complétée avec ce que Smith dit à l'oral dans sa présentation (traduit de l'anglais).

Tableau 2 : Les méthodes de génération de contenu orienté gameplay

<b>Méthode</b>	<b>Puissance</b>	<b>Péril</b>
Constructive	Simple à éditer Customisation	Contenu répétitif Ad Hoc (peu expansible/interchangeable)
Basé sur la contrainte	Garanties sur le design Déclarative (mode de pensée plus accessible pour les designers non orientés techniques)	Traduire en contraintes Debugger (contraintes contradictoires potentiellement difficiles à identifier)
Basé sur l'optimisation	Généralité Émergence	La fitness function Vitesse
Grammaire	Émergence Simple à éditer	Potentiel pour sur- et sous-générer

### 3. Définitions

Pour répondre à la question posée au début, à savoir s'il est possible de faire via un algorithme du contenu aussi captivant et engageant que du contenu fait main par un designer, il nous faut définir les termes de cette problématique.

Il y a ici trois termes auxquels nous donnerons une définition :

- Engageant
- Captivant
- Designer

Ensuite, nous donnerons les principaux éléments qui rendent un jeu captivant et engageant.

#### 3.1 Engageant

##### 3.1.1 Définition

La définition d'engageant est la suivante (Larousse, Date inconnue) :

*« Qui attire, séduit, qui exerce un certain attrait : Propositions engageantes. »*

Dans le cadre du jeu vidéo, du contenu engageant sera du contenu qui attire le joueur. Il s'agit du contenu qui donne envie au joueur de commencer à interagir avec, qui intrigue et donne envie d'essayer.

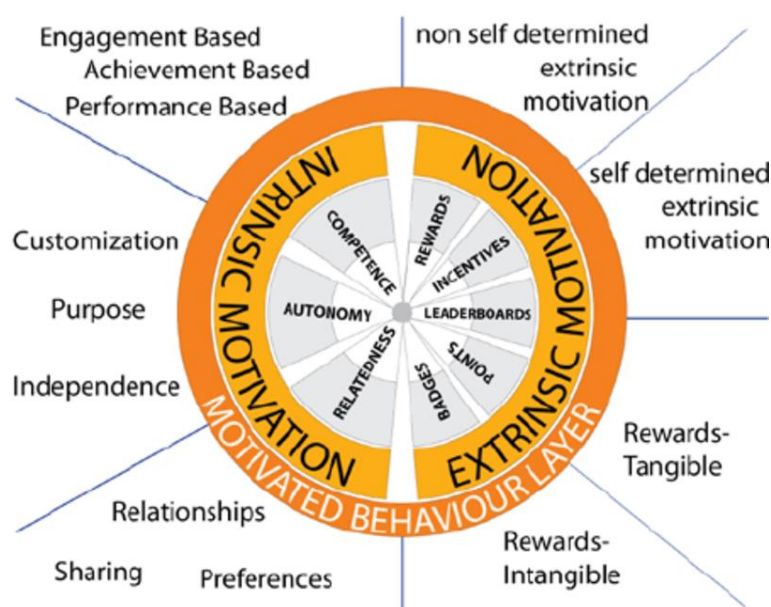
Je pourrai juger du caractère engageant d'un jeu en analysant les interactions d'un joueur avec l'ensemble du contenu, notamment les éléments à caractère optionnel pour la complétion du jeu.

À noter, il est important de faire la distinction entre les différents types de motivation, à savoir intrinsèque et extrinsèque (Brown, 2020) :

- La motivation extrinsèque consiste à donner envie au joueur de jouer, car il peut obtenir une récompense. De la monnaie du jeu, des objets ou de l'expérience par exemple.
- La motivation intrinsèque consiste à donner envie au joueur de jouer, car le contenu est intéressant ou amusant, car il prend du plaisir à faire les tâches présentées par le jeu : faire la tâche est une récompense en soi.



Figure 18 : Représentation des motivations intrinsèques et extrinsèques.



(Kappen, 2016)

Dans cette étude, j'essaierai donc de pousser du côté de la motivation intrinsèque, car c'est cette classe de motivation qui permettra le mieux de juger du caractère engageant d'un jeu.

### 3.1.2 Mesures

Voici une proposition de mesures qui permettent de déterminer si un jeu est engageant.

Tableau 3 : Mesures du caractère engageant d'un jeu

Type	Métrique	Unité
Quantitatif	Proportion d'interaction	En pourcentage, nombre d'interactions différentes explorées par le joueur par rapport au nombre d'interactions possibles
Qualitatif	Avis des testeurs	Retour écrit

## 3.2 Captivant

### 3.2.1 Définition

Pour captiver, on obtient la définition suivante (Larousse, Date inconnue) :

*« Retenir l'attention de quelqu'un, le séduire par un intérêt puissant : Captiver un auditoire. »*

Dans le cadre du jeu vidéo, un contenu captivant sera du contenu dans lequel le joueur passera volontiers plusieurs heures. Cela peut prendre la forme d'une histoire principale

passionnante ou d'un monde riche à explorer. Il s'agit du contenu qui garde le joueur, une fois qu'il a commencé à s'intéresser au jeu.

Il ne faut cependant pas confondre captivant et addictif, le second employant des méthodes basées sur la psychologie pour piéger l'attention du joueur. Nous n'aborderons pas ces méthodes dans cette étude.

### 3.2.2 Mesures

Voici une proposition de mesures qui permettent de déterminer si un jeu est captivant.

Tableau 4 : Mesures du caractère captivant d'un jeu

Type	Métrique	Unité
Quantitatif	Temps de jeu	En minutes
	Pourcentage de complétion	En pourcentage, contenu complété par rapport à l'ensemble du contenu disponible
Qualitatif	Avis des testeurs	Retour écrit

## 3.3 Le designer

### 3.3.1 Définition

Le game designer a pour responsabilité de contrôler comment un joueur interagit avec le jeu, en définissant les règles et systèmes sur lesquels le jeu repose dans le but de créer une expérience de jeu souhaitée.

Le designer et développeur Robert Zubek définit dans son livre Elements of Game Design le game design comme étant composé de trois éléments (Zubek, 2020) :

- Les systèmes et mécaniques, qui sont les règles et les objets du jeu.
- Le gameplay, qui constitue l'interaction entre le joueur, les mécaniques et les systèmes du jeu.
- L'expérience du joueur, qui correspond à comment les utilisateurs se sentent en jouant au jeu.

### 3.3.2 Le processus créatif

Zubek décrit le processus de design pour un jeu vidéo comme ayant deux approches : top down ou bottom up (Zubek, 2020).

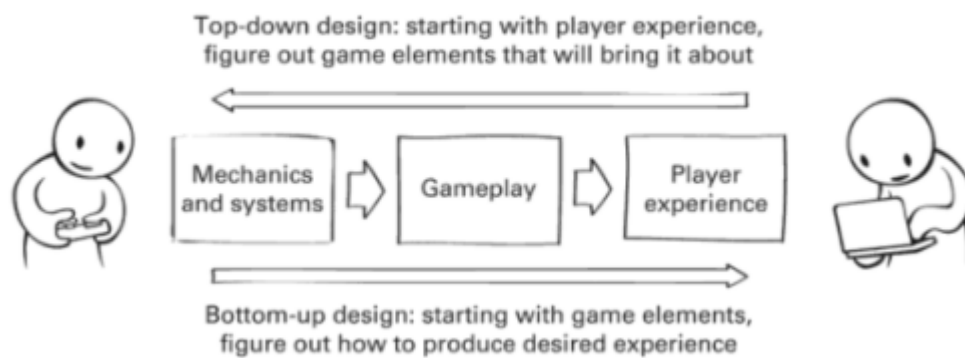
L'approche top down consiste à commencer avec une expérience désirée pour le joueur, et déterminer comment la séparer en plusieurs pièces. On recherche quel type de

gameplay génère cette expérience, et comment l'implémenter avec les mécaniques avec lesquelles nous sommes familiers (Zubek, 2020).

L'approche bottom up consiste à explorer l'espace de jeu, les mécaniques et systèmes en premier, et de les essayer avec des joueurs, testant de façon continue quel type de gameplay et d'expérience est produit (Zubek, 2020).

Zubek complète en expliquant qu'aucune de ses approches seules ne fonctionne réellement en pratique. La solution proposée est d'opter pour un modèle hybride et itératif, créant des plans top down et expérimenter en bottom up, et en créant des prototypes pour tester ces idées de design et les faire converger (Zubek, 2020).

Figure 19 : Approches du game design



(Zubek, 2020)

Ces approches seraient également valides pour créer des parties spécifiques du jeu, comme les niveaux et environnements (pour le level design), ou les combats (encounter design).

## 4. Faire un jeu engageant et captivant

Mark Brown, journaliste spécialisé autour du game design et auteur de Game Maker's Toolkit (référéncé par Riot Games comme ressource pour le game design) donne plusieurs points comme étant importants pour créer un jeu qui garde les joueurs engagés et captivés, sans tomber dans l'addiction (Brown, 2018) :

- Le rythme du jeu (traduit de l'anglais "pacing").
- Le progrès vers un objectif.
- Un défi à la hauteur des compétences du joueur.

De plus, étant donné que chaque joueur est différent, leurs attentes seront elles aussi différentes. Toutefois, un outil existe pour classer les différents profils en quatre catégories : la taxonomie de Bartle.

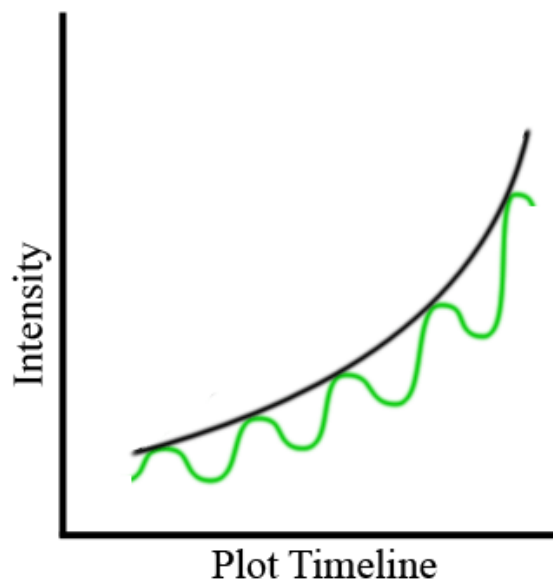
### 4.1 Le rythme

Cette notion est primordiale pour s'assurer qu'un jeu ne devienne pas répétitif ou ennuyeux. Rester trop longtemps sur du gameplay à faible intensité mènera inévitablement à un certain ennui, alors que du gameplay intense sur de trop longues périodes créera de l'épuisement ainsi qu'une désensibilisation, perdant ainsi l'attention du joueur.

Il faut donc trouver un équilibre entre les deux, en augmentant graduellement l'intensité, en laissant des périodes plus calmes pour laisser aux joueurs le temps de se reposer.

Le professeur Mata Haggis donne comme outil pour maîtriser ce déroulement une courbe utilisée en écriture narrative : la courbe de tension, une courbe abstraite plaçant en relation la tension narrative et le temps (Haggis, 2017).

Figure 20 : Courbe de tension dramatique



(Lopez, 2008)

Cette courbe peut également être utilisée pour maîtriser le rythme d'un jeu, en ayant l'intensité du gameplay à la place de la tension narrative.

Les principaux outils qu'ont les développeurs pour contrôler la tension sont l'apport de nouveautés, l'histoire du jeu, la difficulté, mais aussi l'intensité des segments de gameplay.

#### 4.1.1 L'apport de nouveautés

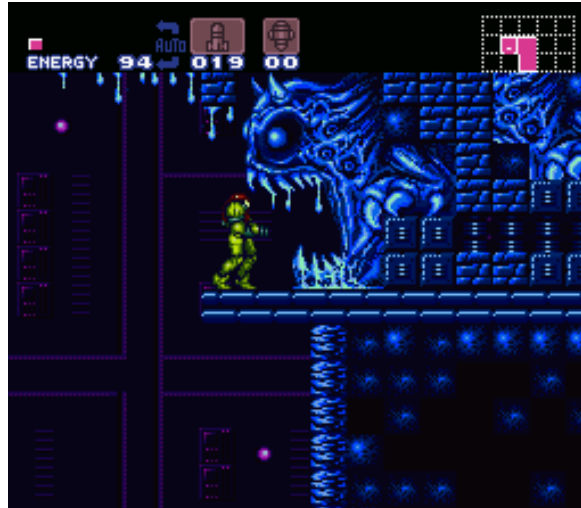
L'apport de nouveautés consiste à introduire des éléments de gameplay ou d'histoire petit à petit. Il peut s'agir de nouveaux pouvoirs, de nouvelles zones à explorer, ou d'éléments d'intrigue.

Ces nouveautés peuvent être sublimées en ayant un élément de mystère, par exemple en montrant un passage inaccessible, ou en évoquant une zone avant que le joueur y ait accès pour créer de l'anticipation, et ainsi étaler ce sentiment de nouveauté plus longtemps, atténuant la nécessité de développer plus de contenu pour ce même sentiment de nouveauté.

Cette approche est très présente dans les metroidvanias, où fréquemment le joueur est présenté à un passage inaccessible sans le pouvoir nécessaire pour le passer. Cela a pour effet de sublimer l'obtention du pouvoir en question.

Par exemple, l'entrée du repaire de Kraid dans Super Metroid est inaccessible avant d'obtenir les high jump boots. Mais le design unique de cette porte l'ancre dans la mémoire du joueur, l'incitant à revenir pour savoir ce qu'il y a derrière cette porte.

Figure 21 : Entrée du repaire de Kraid dans Super Metroid, 1994



(Bearbog, date inconnue)

#### 4.1.2 L'histoire du jeu

L'histoire du jeu peut influencer sur l'intensité perçue par le joueur. Un combat paraîtra plus intense si la survie d'un personnage dépend de son issue, ou moins intense s'il s'agit par exemple d'une session d'entraînement en milieu contrôlé.

#### 4.1.3 La difficulté

La difficulté d'un combat, d'un puzzle ou d'une négociation influe sur l'intensité perçue par le joueur. En effet, un combat contre un boss difficile sera plus intense qu'un combat contre un petit groupe d'ennemis relativement faibles.

Cette augmentation de difficulté peut par exemple se faire en augmentant le niveau de compétence demandé au joueur, en ajoutant une limite de temps ou de ressources, ou en retirant des outils sur lesquels le joueur se repose habituellement.

#### 4.1.4 Les segments de gameplay

Différents segments de gameplay auront naturellement des degrés d'intensité différents.

Un puzzle sera généralement moins intense qu'un combat contre une horde d'ennemis, et une section de plateforme paraîtra plus périlleuse qu'une exploration en plaine.

Par exemple, la série Mass Effect alterne entre ses principaux piliers que sont le combat, les dialogues, et les phases d'exploration afin de contrôler cette courbe d'intensité.

## 4.2 La progression

Le sentiment de progression est un puissant moteur pour un joueur. Il s'agit du cœur du genre du jeu de rôle, que ce soit par rapport à la progression d'un personnage en termes de niveaux ou de compétences, de progression dans l'histoire, ou de progression dans le monde.

Avoir un objectif à long terme sert de moteur pour continuer au travers du contenu présenté au joueur, et des objectifs à court et moyen terme permettent de lui éviter d'être submergé par l'ampleur de la tâche et d'avoir des choses vers lesquelles avancer tout au long de son expérience.

Aussi, un objectif défini par le joueur lui-même permettra de créer une bien meilleure connexion entre le joueur et le jeu, l'objectif servant de motivation intrinsèque.

Stardew Valley est un très bon cas d'étude pour la mise en œuvre de la progression pour captiver le joueur, avec des objectifs long terme définis par le joueur (tel que la construction d'une grande ferme, l'amélioration des relations avec les nombreux habitants du village, devenir riche...), parsemés de petites tâches et autres marques de progrès à plus court terme (comme la création d'un bâtiment, l'automatisation d'une partie de la ferme, l'achat de certaines plantes...).

Le joueur décide de ces objectifs et dispose de nombreuses possibilités pour y parvenir, ce qui lui donne de nombreuses raisons de continuer de jouer pour les atteindre.

Figure 22 : Une ferme dans le jeu Stardew Valley

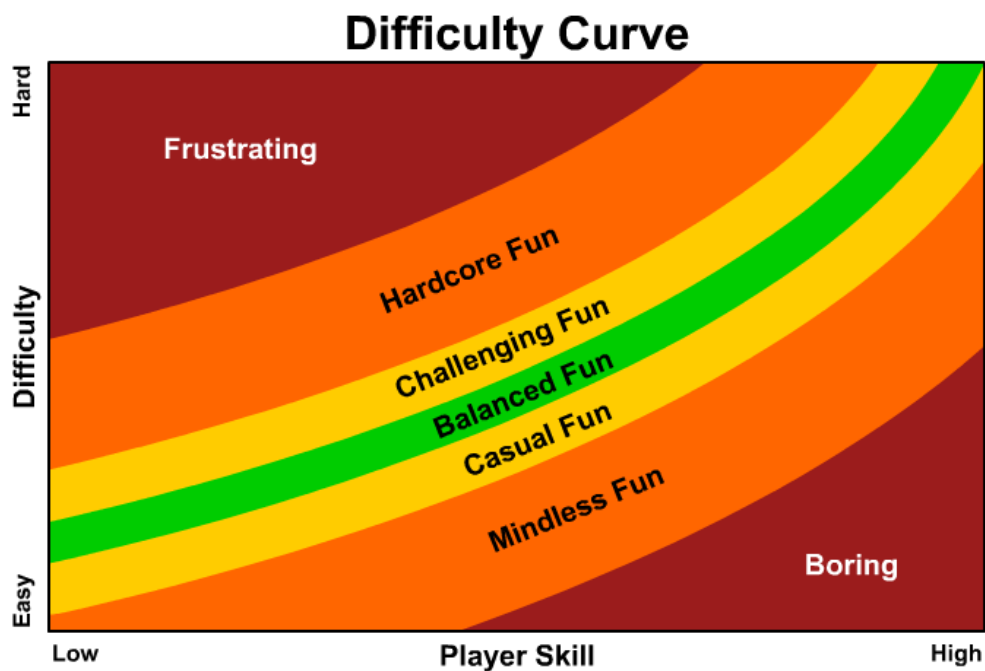


(Stardew Valley, 2016)

### 4.3 Le défi

Enfin, la notion de défi est à considérer. Haggis montre la courbe de défi, outil plaçant la difficulté en relation avec le niveau de compétence du joueur. Un jeu trop facile peut devenir ennuyeux, alors qu'un jeu trop difficile court le risque de devenir stressant (Haggis, 2017).

Figure 23 : Courbe de difficulté par rapport aux compétences du joueur



(Maletz, 2012)

Il est donc important de trouver un entre-deux, où le joueur est suffisamment défié par le jeu pour ne pas s'ennuyer, sans devenir difficile au point d'être frustrant.

#### 4.3.1 Les IA Directrices

Comme mentionné dans la section sur l'intelligence artificielle, certains jeux implémentent des IA chargées d'adapter l'apparition d'ennemis ou la quantité de munitions disponibles afin d'adapter la difficulté aux compétences des joueurs et à la situation actuelle.

Cela a pour but d'adapter la difficulté pour rester dans la zone "fun" décrite par la figure de la courbe de difficulté ci-dessus.

#### 4.3.2 Avec la génération procédurale

Une étude se concentre sur ce point afin de permettre de créer de façon procédurale des niveaux prenant en compte des courbes de difficulté apportées par des designers (Bakkes et al., 2015). Ces niveaux sont générés avec une combinaison de génération



basée sur la grammaire et l'optimisation, afin de créer des niveaux logiques qui sont mutés afin de s'approcher des courbes données par un designer.

Cette étude conclut en donnant comme évolution suivante à cette solution la réalisation d'un algorithme qui adaptera le niveau de difficulté aux performances du joueur, afin de garantir une expérience au niveau de difficulté souhaité quel que soit le niveau de compétences du joueur.

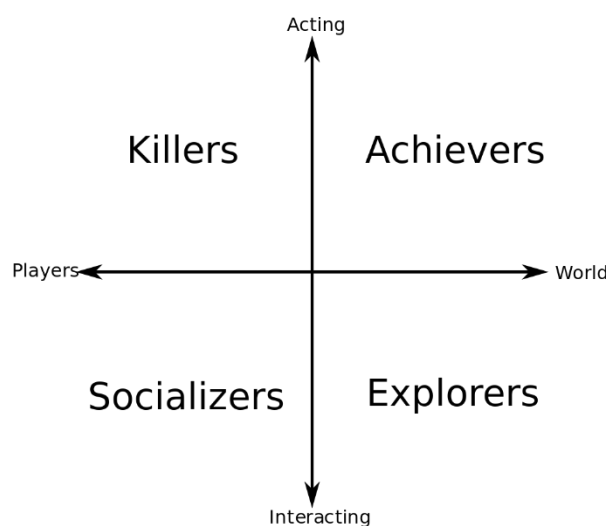
#### 4.4 La taxonomie de Bartle

Il est important de garder à l'esprit que chaque joueur est différent. Différents joueurs auront des motivations différentes, et un jeu ne plaira pas forcément à tous.

Aussi, la taxonomie de Bartle nous donne des éléments pour classer les joueurs. Cette taxonomie, théorisée par Richard Bartle en 1996, répartit les joueurs selon ce qu'ils recherchent dans un jeu. D'abord théorisée pour les MMORPG<sup>7</sup> ou les jeux multijoueurs, elle s'applique aussi aux jeux à un joueur.

Cette taxonomie donne quatre groupes de joueurs : les carreaux, trèfles, cœurs et piques, aussi appelés achievers, killers, socializers et explorers, et sont classés selon leurs préférences, soit en termes d'interaction par opposition à l'action directe, soit en termes de sujet pour cette interaction ou action, entre le monde du jeu et les autres joueurs.

Figure 24 : Taxonomie de Bartle



(Ramiomagalhaes, 2013)

---

<sup>7</sup> Massively Multiplayer Online Role-Playing Game, un genre de jeux en ligne massivement multijoueur.

#### **4.4.1 Les carreaux**

Les carreaux, aussi appelés achievers, sont les joueurs attirés par les mesures concrètes de la progression dans un jeu, tels que des niveaux, l'équipement, ou des succès. Ce sont des joueurs qui sont capables de faire beaucoup pour des récompenses telles que de simples cosmétiques (Bartle, 1996).

#### **4.4.2 Les trèfles**

Les trèfles, aussi appelés killers, sont des joueurs attirés par la compétition avec d'autres joueurs (Bartle, 1996). Bien que la possibilité de causer du trouble parmi des opposants contrôlés par un ordinateur peut les amuser, il s'agit d'un profil principalement attiré par les jeux multijoueurs.

#### **4.4.3 Les cœurs**

Les cœurs, aussi appelés socializers, sont les joueurs pour qui l'aspect le plus intéressant dans un jeu est la rencontre d'autres joueurs avec lesquels interagir (Bartle, 1996). Principalement orienté multijoueur, ce type de joueurs peut également se plaire sur des jeux à un joueur, tels que les jeux proposant des relations avec des personnages poussés comme Mass Effect ou Stardew Valley, ou alors les jeux ayant une importante communauté autour. Ce seront généralement des jeux créatifs avec des forums actifs, tels que les Sims, ou Dwarf fortress, les joueurs partageant leurs exploits et créations.

#### **4.4.4 Les piques**

Enfin, les piques, aussi appelés explorers, sont les joueurs attirés par la découverte de zones, et l'immersion dans le monde que propose le jeu. Ils préfèrent explorer selon leur propre rythme, et sont friands de secrets à trouver (Bartle, 1996).

#### **4.4.5 Dans le cadre de ce travail**

Pour notre projet, les profils les plus pertinents sont les explorers, et quelque peu les achievers. N'ayant aucun élément multijoueur, les killers ne sont pas représentés, et les socializers ne trouveront pas de personnages avec lesquels interagir.

Connaître les profils qui peuvent caractériser un joueur me permet d'identifier les points qui permettent d'influencer sur l'aspect engageant et captivant d'un jeu.

## 5. Travail pratique

Afin de répondre à la problématique, j'ai choisi d'utiliser la génération procédurale pour quelque chose que seule cette approche permet : altérer le contenu en cours de jeu, pour prendre en compte le profil du joueur.

L'idée est de créer dynamiquement des niveaux en s'appuyant sur les actions et les performances du joueur pour altérer la structure de l'environnement où évolue celui-ci.

Pour ce faire, il est possible de s'appuyer sur les trois éléments importants pour rendre un jeu engageant et captivant donné plus tôt : le rythme, la progression et le défi.

Le but est de créer du contenu qui sera engageant pour ce joueur, en créant par exemple plus d'éléments à collecter s'il semble apprécier cet aspect du jeu, ou en ajustant la difficulté pour rester à un niveau où l'expérience est satisfaisante.

### 5.1 Le choix du projet

J'ai choisi pour ce projet de faire un prototype de platformer 3D.

La première raison étant que je suis nettement plus à l'aise pour manipuler des environnements 3D.

Ensuite, les platformers tels que les jeux des séries Rayman ou Mario sont traditionnellement coupés en niveau, ce qui me permet d'avoir une structure idéale pour segmenter la génération. Il suffit de prendre les données du joueur sur un niveau, évaluer ces données, et générer le niveau suivant en fonction.

Aussi, le genre est plus facile à prendre en main que d'autres, ce qui me permettra d'obtenir des tests d'une plus grande variété de joueurs.

Enfin, le platformer étant un genre où le level design est primordial, il s'agit du genre qui permettra à la génération de niveau d'être au centre de l'attention des joueurs, évitant ainsi au maximum les éléments parasites susceptibles de biaiser les résultats.

Cependant, un point important est à noter : le caractère engageant d'un jeu, défini comme étant un aspect très lié aux objectifs secondaires, ne pourra être testé que succinctement par rapport à l'aspect captivant.

La génération de contenu supplémentaire par rapport à la base prévue (à savoir la génération de niveau) aurait pu prendre la forme de niveaux bonus, de secrets à trouver, ou encore de chemins alternatifs à explorer. Mais cette étude ayant un temps limité pour être réalisée, l'intégration d'éléments aussi complexe risquerait de mettre l'ensemble du

projet en retard. Aussi, je représenterais cet aspect sous la forme de pièces à collectionner. Bien qu'imparfaite, cette solution permet tout du moins d'avoir un aperçu de réponse vis-à-vis de ce critère.

## 5.2 Le choix de technologie

### 5.2.1 Le moteur de jeu

Pour réaliser ce prototype, j'ai dû choisir un moteur de jeu. En développer un aurait demandé trop de temps et n'aurait rien apporté à cette étude, aussi j'ai comparé mes options.

Tableau 5 : Tableau d'évaluation des options pour le choix de moteur.

Critère / Moteur	Unity	Unreal Engine	Godot	GameMaker
Connaissances préalables	Oui	Limitées	Aucune	Aucune
Langage	C#	C++, Blueprints	GDScript	GML
Documentation	Complète et claire	Complète et claire	Complète et claire	Complète et claire
Tutoriels et autres ressources	En abondance	Relativement nombreux	De plus en plus nombreux	Limités
Support 3D	Intégral	Intégral	Relativement incomplet	Très limité

Les critères déterminants ici sont le support de la 3D, ainsi que l'expérience préalable : le jeu à réaliser sera en 3D, et avec une limite de temps, avoir des connaissances préalables sur la technologie à utiliser est un atout non négligeable.

Ainsi, Unity apparaît comme le choix le plus pertinent, et sera donc le moteur de jeu choisi pour la réalisation de ce prototype. La version 2020.3.12 LTS sera la version utilisée pour ce projet.

### 5.2.2 Les packages utilisés

Unity est un moteur permettant beaucoup d'extension. Afin de simplifier certains aspects de la réalisation de ce prototype, j'ai utilisé des packages disponibles sur l'asset store.

Voici un tableau récapitulatif des packages utilisés ainsi que leur utilisation.

Tableau 6 : Liste des packages utilisés pour la réalisation du prototype.

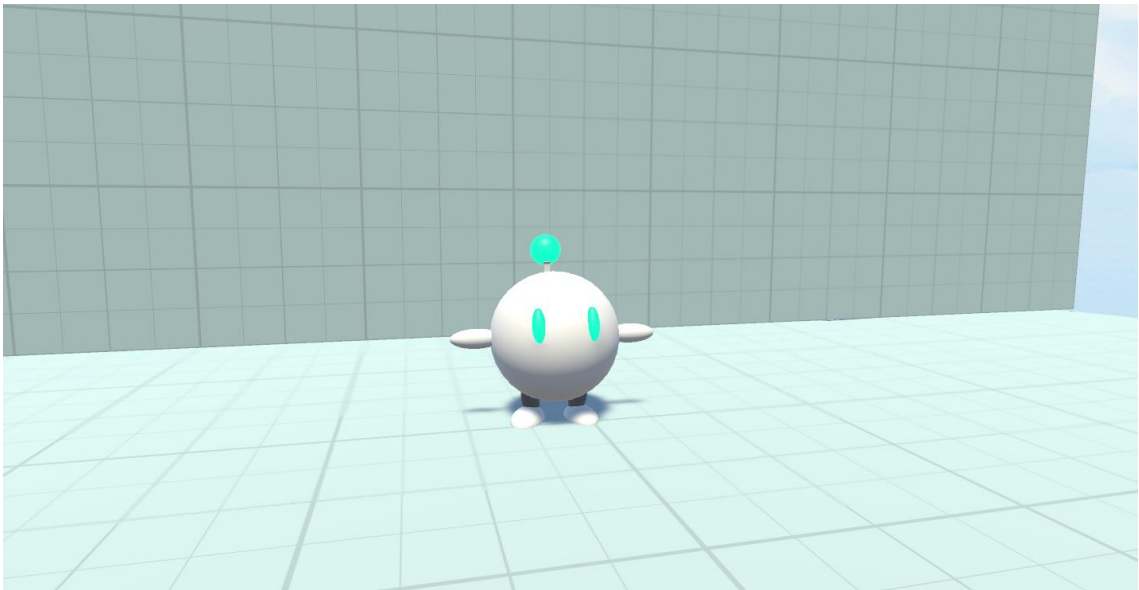
Nom	Auteur	Utilisation dans le projet
Rewired	Guavaman Enterprises	Généraliser la prise d'inputs, notamment pour l'adaptation de différents périphériques
Odin	Sirenix	Donner des wrapper aux fonctions de personnalisation d'éditeur sous forme d'annotations
Sensor Toolkit	Micosmo	Outils génériques pour gestion de collision ou détection
Rainbow Folder	Borodar	Outil colorant les dossiers du projet
Polygon - Prototype	Synty studio	Ensemble d'assets 3D prêts pour le prototypage
DOTween	Demigrant	Librairie de "tweening" (transitions de valeurs, comme des positions ou des nombres)
Dynamic Bones	Will Hong	Utilisé pour animer procéduralement l'antenne et les bras de Beep-O

## 6. Le prototype

### 6.1 Le personnage du joueur

Le jeu que j'ai créé propose d'incarner un petit personnage qui évolue dans un monde 3D. Le personnage en question est un petit robot, aspect choisi pour coller à la thématique de génération procédurale, nommé Beep-O. J'ai également choisi de donner ce nom au prototype.

Figure 25 : Beep-O, le personnage incarné dans ce prototype de jeu

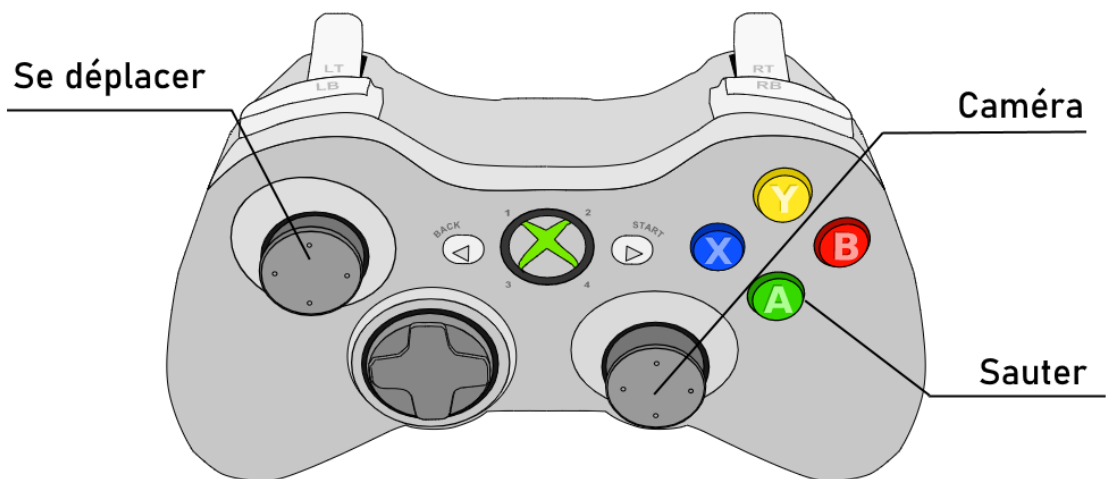


(Beep-O, 2021)

Le jeu est un plateforme 3D, et se joue avec une manette.

Le personnage est capable de se déplacer et sauter. Il dispose de 3 vies, et retourne au point de départ du niveau lorsque ces vies sont épuisées.

Figure 26 : Contrôles de base pour le personnage du prototype, sur manette Xbox



(Jishenaz, 2013), Modifié à l'aide de l'outil : Paint.net

L'objet du joueur est fait de quatre composants :

- Un character controller, composant natif de Unity permettant de simplifier la gestion du mouvement d'un personnage par rapport aux collisions sans avoir à se reposer sur un rigid body<sup>8</sup>
- Un script Player
- Un script CamControl
- Un script PlayerHealth

### 6.1.1 Le script Player

Le script Player gère les déplacements du joueur, par rapport aux commandes reçues et à l'environnement. Il gère également les animations du personnage.

Il fonctionne en construisant un vecteur, appelé moveVector, en fonction du comportement voulu par le joueur.

À la base de ce script se trouve la fonction Update, fonction fournie par Unity qui sera exécutée à chaque frame<sup>9</sup>.

---

<sup>8</sup> composant de Unity modélisant la physique pour un objet

<sup>9</sup> Une image générée par le jeu, unité de temps utilisée pour l'exécution de processus dans un jeu

Figure 27 : Script Player - Update (Partie 1)

```
Message Unity | 0 références
void Update()
{
    // Récupération des inputs
    if(!controlLost)
        moveVector = InputCamMapping() * moveSpeed;
    // Gestion de la gravité
    Gravity();
    if (!controlLost)
        JumpInput();
    // Application des intentions du joueur vers le character controller
    Move();
}
```

(Beep-O, 2021)

Cette fonction fait appel à trois fonctions principales pour la gestion des déplacements.

Elle définit également moveVector comme étant l'entrée saisie à la manette relative à la caméra, multipliée par la vitesse de déplacement maximale du personnage.

#### 6.1.1.1 Déplacements horizontaux

La fonction InputCamMapping fonctionne en récupérant le transform de la caméra, puis en multipliant les vecteurs normalisés dirigés vers l'avant et la droite de la caméra de la caméra par les axes X et Z donnés comme entrée via la fonction Input.

Figure 28 : Script Player - InputCamMapping

```
1 référence
private Vector3 InputCamMapping()
{
    Vector3 camF, camR;
    camF = camTransform.forward;
    camR = camTransform.right;

    camR.y = 0;
    camF.y = 0;

    return camR.normalized * Input().x + camF.normalized * Input().z;
}
```

(Beep-O, 2021)

La fonction Input récupère, ici en utilisant le package Rewired, les entrées du joystick gauche d'une manette type Xbox et les place dans un vecteur, plaçant l'axe horizontal sur l'axe X du vecteur input, et l'axe vertical sur l'axe Z.



Figure 29 : Script Player - Input

```
2 références
private Vector3 Input()
{
    Vector3 input = new Vector3();
    input.x = ReInput.players.GetPlayer(0).GetAxis("MoveHorizontal");
    input.z = ReInput.players.GetPlayer(0).GetAxis("MoveVertical");
    return input;
}
```

(Beep-O, 2021)

#### 6.1.1.2 Gravité

Une fois les intentions du joueur représentées dans le vecteur moveVector, la gravité est appliquée.

Si le personnage n'est pas au sol, on réduira la valeur verticalMovement, qui représente la vitesse verticale du joueur, en prenant en compte le temps passé depuis la dernière frame.

Figure 30 : Script Player - Gravity

```
1 référence
private void Gravity()
{
    if (!controller.isGrounded || controlLost)
    {
        verticalMovement -= gravity * Time.deltaTime;
    }
    else
    {
        verticalMovement = -defaultGroundedGravity;
    }
}
```

(Beep-O, 2021)

Passer par une valeur intermédiaire permet d'effectuer une transition douce d'un état à un autre, et simplifie la lecture lorsque la composante verticale du mouvement est altérée indépendamment de sa composante horizontale.

Cette gravité est également appliquée lors d'une perte de contrôle, car celle-ci signifie que le joueur est repoussé par un ennemi ou un obstacle, levant le joueur hors du sol sans qu'un saut n'ait été initialisé.

Si le joueur est au sol, une gravité par défaut est appliquée, afin d'adoucir la descente de pentes.

### 6.1.1.3 Saut

Après l'application de gravité, on vérifie si le joueur a le contrôle du personnage, si le personnage se trouve au sol, et s'il souhaite sauter.

Si c'est le cas, on applique la force du saut définie pour le joueur à `verticalMovement`.

Figure 31 : Script Player - Jump

```
1 référence
private void JumpInput()
{
    if (ReInput.players.GetPlayer(0).GetButtonDown("Jump") && controller.isGrounded)
    {
        Jump();
    }
}
1 référence
public void Jump()
{
    verticalMovement = jumpSpeed;
}
```

(Beep-O, 2021)

Faire deux fonctions séparées permet d'appeler la fonction `Jump` indépendamment des entrées du joueur, par exemple pour faire rebondir le joueur lorsqu'il saute sur un ennemi.

### 6.1.1.4 Application des déplacements

Enfin, une fois ces transformations faites au vecteur `moveVector` et à la valeur `verticalMovement`, la fonction `Move` est chargée d'appliquer `verticalMovement` à la composante Y de `moveVector`, puis de déplacer le character controller selon ce vecteur, en prenant en compte le temps passé depuis la dernière frame.

Figure 32 : Script Player - Jump

```
1 référence
private void Move()
{
    moveVector.y = verticalMovement;
    controller.Move(moveVector * Time.deltaTime);
}
```

(Beep-O, 2021)

Prendre en compte ce delta de temps permet d'avoir un comportement consistant, et ce même avec des différences de framerate<sup>10</sup>.

---

<sup>10</sup> Taux de rafraîchissement de l'image dans un jeu

#### 6.1.1.5 Animations

Ce script gère également les animations du personnage, ainsi que les chutes sous le niveau.

Figure 33 : Script Player - Update (Partie 2)

```
// Application des intentions du joueur vers le character controller
Move();

// Visuels
Animate();
if (!controlLost)
    RotateCharTowardPos(new Vector3(moveVector.x, 0, moveVector.z));
// Gestion de chutes
if (controller.isGrounded)
    lastGroundedPos = transform.position;
if (transform.position.y < -10)
{
    onPlayerFall.Invoke();
    ReturnToCheckpoint();
}
```

(Beep-O, 2021)

Les animations prennent en compte l'état (au sol ou non, ainsi que la perte de contrôle) et la vitesse du personnage, et le modèle est orienté par rapport à la direction donnée par le joueur.

#### 6.1.1.6 Chutes

La gestion des chutes est très simple, on enregistre la dernière position où le personnage touchait le sol, et si le joueur passe sous le niveau (identifié comme étant une position ayant -10 sur l'axe Y), le joueur est placé à la position enregistrée.

Le retour du joueur se fait dans la fonction `lateUpdate`, ceci afin de passer après toute modification de position faite par le `characterController`. On désactive ce dernier, réinitialise toutes les composantes de mouvement du joueur, et place le personnage à la dernière position au sol connue.

Figure 34 : Script Player - LateUpdate

```
Message Unity | 0 références
void LateUpdate()
{
    if (returnToLastGroundedPos)
    {
        // Mise à l'arrêt du personnage et désactivation temporaire du character controller
        // évite une override de la position par le character controller
        moveVector = Vector3.zero;
        verticalMovement = 0;
        controller.enabled = false;
        transform.position = lastGroundedPos;
        // Coroutine pour la perte de contrôle, évite au joueur de tomber directement après avoir réapparu
        StartCoroutine(ControlLoss(knockBackDuration));
        returnToLastGroundedPos = false;
    }
}
```

(Beep-O, 2021)

#### 6.1.1.7 Pertes de contrôle

Enfin, ce script s'occupe de faire perdre le contrôle au joueur lorsque le joueur touche un obstacle ou un ennemi. Cela se fait via une coroutine, ajustant le booléen controlLost pour une durée définie.

Figure 35 : Script Player - ControlLoss

```
2 références
IEnumerator ControlLoss(float controlLossDuration)
{
    yield return new WaitForSeconds(controlLossDuration);
    controller.enabled = true;
    moveVector.y = 0;
    controlLost = false;
}
```

(Beep-O, 2021)

Pour la communication avec la hitbox, qui se trouve dans un objet enfant de l'objet joueur, une fonction publique HitKnockBack permet au travers de sa signature de communiquer avec un évènement appelé dans l'objet contenant la hitbox et d'obtenir la position de l'obstacle pour pouvoir déplacer le joueur par rapport à cet obstacle.

Figure 36 : Script Player - HitKnockBack

```
0 références
public void HitKnockBack(GameObject origin, Sensor sensor)
{
    moveVector = (transform.position - origin.transform.position).normalized * knockBackStrength;
    verticalMovement = knockBackStrength;
    controlLost = true;
    modelTransform.LookAt(origin.transform.position);
    StartCoroutine(ControlLoss(knockBackDuration));
}
```

(Beep-O, 2021)

### 6.1.2 Le script CamControl

Ce script permet de contrôler la caméra, en passant par deux transforms. Ces transforms servent de pivots, permettant de simplifier les calculs de rotation.

Figure 37 : Les pivots de la caméra par rapport au joueur



(Beep-O, 2021)

La caméra, une caméra virtuelle du package cinemachine, suit camPivotX.

Deux fonctions principales composent ce script : CamController, qui permet d'orienter la caméra avec le joystick droit, et CamDistance, qui permet d'éviter à la caméra de passer au travers d'un mur.

Figure 38 : Script CamControl - Update

```
Message Unity | 0 références
void Update()
{
    CamController();
    CamDistance();
}
```

(Beep-O, 2021)

Dans CamController, on récupère les axes horizontaux et verticaux du joystick droit à l'aide de Rewired. Après avoir vérifié que faire tourner le pivot X ne faisait pas basculer la caméra à l'envers, on fait tourner les pivots en fonction de l'entrée saisie par le joueur.

Figure 39 : Script CamControl – CamController

```
1 référence
private void CamController()
{
    if(camX.eulerAngles.x >= camVertClamp.x
        && -ReInput.players.GetPlayer(0).GetAxis("CamVertical") < 0
        || camX.eulerAngles.x <= camVertClamp.y
        && -ReInput.players.GetPlayer(0).GetAxis("CamVertical") > 0)
        camX.Rotate(Vector3.right * -ReInput.players.GetPlayer(0).GetAxis("CamVertical") * camSpeed * Time.deltaTime);
    camY.Rotate(Vector3.down * -ReInput.players.GetPlayer(0).GetAxis("CamHorizontal") * camSpeed * Time.deltaTime);
}
```

(Beep-O, 2021)

### 6.1.3 Le script PlayerHealth

Le script PlayerHealth quant à lui est responsable de la gestion des points de vie du joueur, ainsi que de l'appel d'évènements en cas de décès ou de coup reçu.

Figure 40 : Script PlayerHealth – TakeHit & Die

```
0 références
public void TakeHit()
{
    currentHealth--;
    onHealthChange.Invoke(currentHealth);
    if(currentHealth <= 0)
    {
        Die();
    }
}

1 référence
private void Die()
{
    onDeath.Invoke();
}
```

(Beep-O, 2021)

Ces évènements me permettent de relancer le jeu depuis le GameManager, ainsi que de compter ces occurrences, ce qui me permettra de dresser le profil du joueur.

## 6.2 Les niveaux

### 6.2.1 Structure

Les niveaux seront construits avec comme influence principale les niveaux secrets de Mario Sunshine ou de A Hat in Time. Il s'agit de niveaux composés de blocs simples, flottant dans le vide, et formant un chemin principalement linéaire.

Figure 41 : Un niveau secret dans Mario Sunshine



(Hairball, 2020)

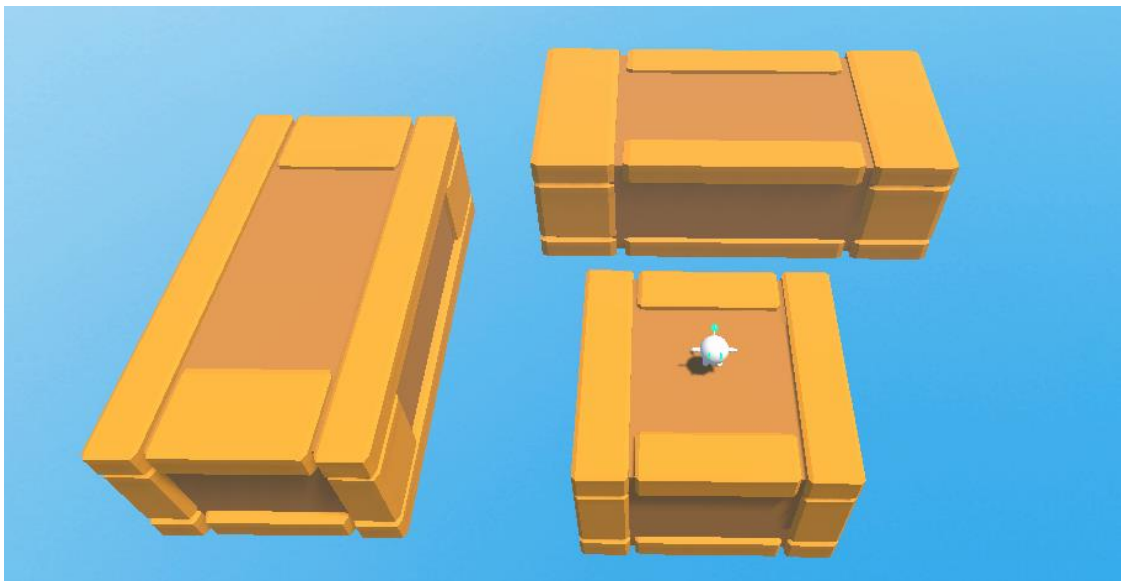
Ce choix de design me permettra de me concentrer sur l'aspect gameplay et l'impact du level design, tout en évitant la complexité que représente un niveau ouvert comme les niveaux standards de ces titres.

## **6.2.2 Les éléments composant les niveaux**

### **6.2.2.1 Les plateformes**

Plusieurs plateformes statiques composent la base d'un niveau. Le joueur peut s'y déplacer librement. Il s'agit de blocs de proportions différentes, qui constituent le squelette d'un niveau.

Figure 42 : Les différentes plateformes

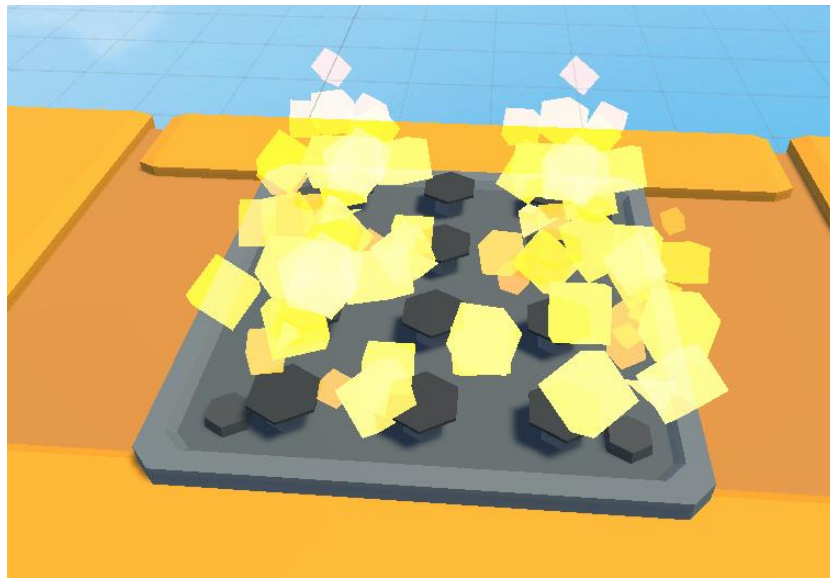


(Beep-O, 2021)

### **6.2.2.2 Les obstacles**

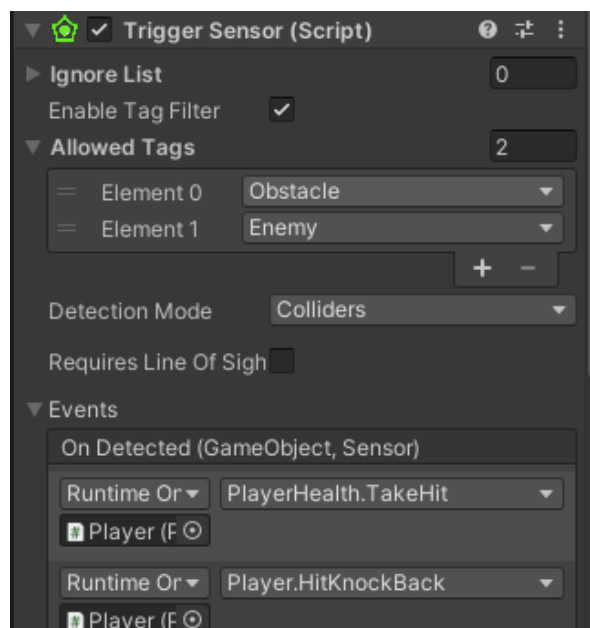
Des obstacles sont placés sur et entre les plateformes afin de créer du défi pour le joueur. Entrer en collision avec un obstacle fait reculer le joueur, lui fait brièvement perdre le contrôle de son personnage, et lui fait perdre une vie. Ils servent à apporter un élément de défi, requérant de l'attention et de l'adresse de la part du joueur.

Figure 43 : Un obstacle



(Beep-O, 2021)

Figure 44 : Le composant Trigger Sensor présent sur le joueur pour la détection de coups reçus



(Beep-O, 2021)

La détection d'obstacle se fait du côté du joueur : la hitbox du personnage contient un Trigger Sensor, simple script offert par sensor toolkit appelant un évènement Unity standard dans la fonction OnTriggerEnter, avec filtre par tag.

À cet évènement sont inscrits :

- PlayerHealth, afin de compter le coup reçu et perdre un point de vie



- Player afin de compter le recul que fait subir l'obstacle
- Quelques systèmes de particule pour signaler au joueur qu'un coup a été reçu
- DataCollector, script responsable d'enregistrer les données de jeu du joueur.

Cet objet est marqué du tag Obstacle, ce qui permet au senseur de savoir que le joueur a touché un obstacle.

### 6.2.2.3 Les ennemis

Des ennemis chercheront à empêcher le joueur d'atteindre son but. Le joueur peut rebondir sur un ennemi en sautant dessus. Ils permettent d'introduire un élément de défi non statique.

Figure 45 : Un ennemi



(Beep-O, 2021)

La détection de collision se fait de la même façon que pour les obstacles.

Un ennemi est composé d'un character controller, et du script EnemyBehaviour, qui est chargé de gérer la logique de déplacement ainsi que le comportement de cet ennemi.

Figure 46 : Script EnemyBehaviour - Update

```
Message Unity | 0 références
void Update()
{
    if (hasSeenPlayer)
    {
        target = playerTransform.position - transform.position;
        target.y = 0;
    }
    else
    {
        if (endOfPause)
        {
            StartCoroutine(Roaming());
        }
    }
    transform.LookAt(
        Vector3.SmoothDamp(
            transform.position + transform.forward,
            transform.position + target,
            ref smoothVelocity,
            lookDelay));
    velocity = target.normalized * currentSpeed * Time.deltaTime;
    velocity.y = -10 * Time.deltaTime;
    controller.Move(velocity);
}
```

(Beep-O, 2021)

Tant que ce robot n'a pas vu le joueur, il alterne entre un état de pause et un état de roaming : il choisit une direction au hasard, avance quelques pas, puis fait une pause, avant de choisir une nouvelle destination.

Figure 47 : Script EnemyBehaviour - Roaming

```
1 référence
IEnumerator Roaming()
{
    endOfPause = false;
    target = new Vector3(Random.Range(-10, 10), 0, Random.Range(-10, 10));
    target.y = 0;
    yield return new WaitForSeconds(Random.Range(roamingTime.x, roamingTime.y));
    target = Vector3.zero;
    yield return new WaitForSeconds(Random.Range(pauseTime.x, pauseTime.y));
    endOfPause = true;
}
```

(Beep-O, 2021)

Figure 48 : Script EnemyBehaviour - Pause

```
1 référence
IEnumerator Pause()
{
    hasSeenPlayer = false;
    endOfPause = false;
    target = Vector3.zero;
    yield return new WaitForSeconds(1f);
    hasSeenPlayer = true;
    endOfPause = true;
}
```

(Beep-O, 2021)

Un senseur attaché à l'ennemi permet de détecter lorsque le joueur approche. Quand il est suffisamment près, l'ennemi passe en mode poursuite, et chasse le joueur jusqu'à ce qu'il le perde de nouveau. Il chassera le joueur, même si pour cela il doit tomber dans le vide.

Figure 49 : Script EnemyBehaviour - DetectsPlayer

```
0 références
public void DetectsPlayer(GameObject playerGo, Sensor sensor)
{
    currentSpeed = runSpeed;
    playerTransform = playerGo.transform;
    hasSeenPlayer = true;
}
```

(Beep-O, 2021)

Afin d'éviter que tous les ennemis du niveau tombent dans le vide, une petite sécurité sous la forme d'un déclencheur devant l'ennemi lance un demi-tour lorsque le déclencheur ne détecte plus de sol.

#### 6.2.2.4 Les pièces

De simples pièces à collectionner peuvent être placées afin de récompenser le joueur. Elles n'ont pas de but en jeu autre que l'augmentation d'un score artificiel.

Figure 50 : Pièce



Ces pièces sont détectées de la même manière que les obstacles et ennemis : via un capteur sur la hitbox du joueur. Le tag Coin est appliqué sur les pièces, ce qui permet au capteur de l'identifier comme tel, et ainsi notifier le DataCollector qu'une pièce a été récoltée.

Cet élément permet d'inclure le profil des Achievers au jeu, bien que ce ne soit qu'une façon très simpliste de le faire. De plus, il s'agit du seul élément du jeu permettant de jauger l'aspect engageant.

#### **6.2.2.5 La condition de victoire**

Le joueur doit atteindre la sortie, marquée par un drapeau, pour remporter le niveau et passer au niveau suivant.

Figure 51 : Drapeau de fin de niveau



### **6.3 L'analyse du joueur**

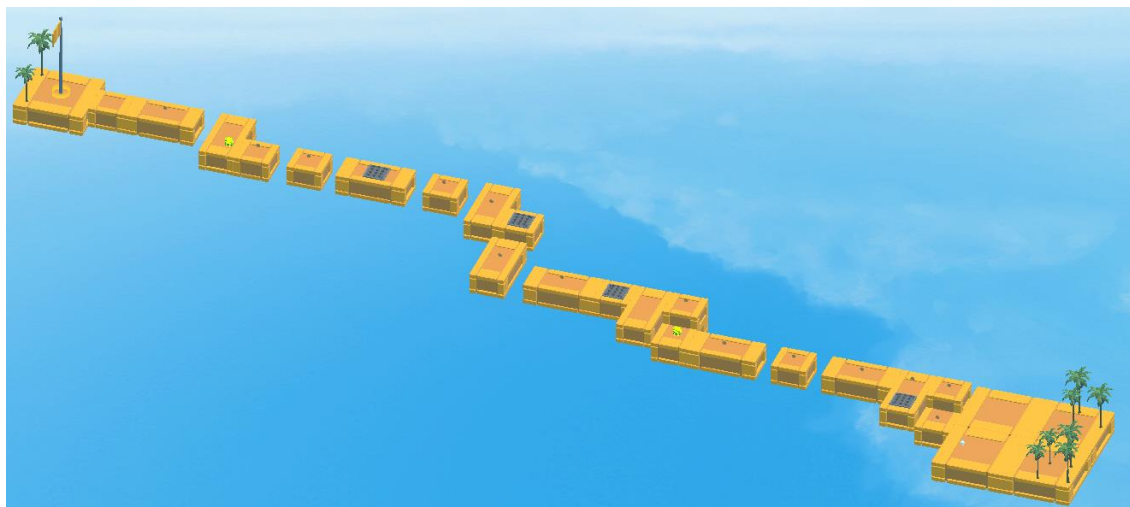
Afin de générer des niveaux adaptés aux compétences et préférences du joueur, il est important d'en brosser un portrait précis. Pour ce faire, j'ai développé un script chargé d'analyser le joueur pendant le jeu, en récoltant diverses informations, telles que le nombre de chutes, et le nombre d'obstacles que le joueur n'a pas esquivé.

#### **6.3.1 Le niveau témoin**

Pour permettre de dresser une première image du joueur, un niveau fait main, similaire pour tous les joueurs, permet de récolter un premier ensemble de données pour alimenter l'algorithme de génération de niveaux.

Ce niveau est prévu pour être simple, et ainsi permettre à tous de le terminer.

Figure 52 : Vue d'ensemble du premier niveau de Beep-O



(Beep-O, 2021)

Les pièges sont placés pour qu'on ait toujours la place pour sauter par-dessus ou se déplacer autour, et les ennemis sont placés sur de grandes plateformes, ce qui facilite leur esquive. Quelques pièces sont également placées le long du parcours.

### 6.3.2 Les points d'observation du joueur

Ci-après, j'ai établi une liste des points que j'observerais via le script DataCollector.

Tableau 7 : Les différents points observés par le prototype

But	Variable	Type
Évaluer les compétences du joueur	Nombre de coups reçus, par ennemis ou obstacles	Entier
	Nombre de décès	Entier
	Nombre de chutes	Entier
Évaluer le rythme de progression du joueur	Position selon le temps	Dictionnaire de vecteurs identifiés par des réels
	Temps de complétion	Réel
Évaluer l'intérêt du joueur pour la collecte	Pièces ramassées	Entier
	Pourcentage de pièces ramassées sur l'ensemble du niveau	Réel

Cette liste donne des points d'observation pour les trois axes identifiés pour rendre un jeu engageant et captivant : le défi, le rythme, et la progression.

Bien que simple, cet ensemble permet d'illustrer les possibilités offertes par cette approche. On peut alors imaginer une version plus complète pour donner des profils plus détaillés.

### **6.3.3 La lecture de ces points**

À partir de ces éléments, je peux brosser un portrait du joueur. Ce portrait permet d'orienter la génération, et ainsi obtenir des niveaux correspondants aux compétences et au rythme de progression du joueur.

Aussi, une augmentation de difficulté d'un niveau à un autre est ajoutée après avoir dressé ce profil, ceci afin de conserver une augmentation de difficulté pour coïncider avec l'amélioration du joueur au jeu. En effet, simplement analyser le niveau actuel du joueur, et construire des niveaux correspondant exactement à ses compétences sans jamais augmenter risque de mener à des niveaux plats, sans progression de l'un à l'autre.

De plus, cette approche permet de représenter la courbe de tension narrative évoquée plus tôt, et ainsi se rapprocher d'un rythme idéal.

### **6.3.4 L'implémentation**

Le script DataCollector s'appuie sur un objet de données appelé PlayerGameplayData, d'une référence sur le joueur.

PlayerGameplayData est un Scriptable Object, une classe donnée par Unity permettant entre autres de créer des objets de données persistants dans les dossiers du jeu.

Cet objet est principalement une liste de LevelPerformance, une classe encapsulant les points d'observation listés plus tôt.

Figure 53 : Script LevelPerformance

```
6 références
public class LevelPerformanceData
{
    public int playerHits;
    public int playerFalls;
    public int playerDeaths;

    public int collectedCoins;
    public int totalCoins;

    public float levelStartingTime;
    public float levelCompletionTime;

    public Dictionary<float, Vector3> posAtTime;

    1 référence
    public LevelPerformanceData(float levelStartingTime)
    {
        this.levelStartingTime = levelStartingTime;
        posAtTime = new Dictionary<float, Vector3>();
    }
}
```

(Beep-O, 2021)

Lorsqu'il souhaite commencer à enregistrer les données, le DataCollector appelle la fonction InitDataCollection de PlayerGameplayData, ce qui initialise la liste des performances, marque le niveau actuel comme -1 (car ce nombre est incrémenté dans AddLevel), et crée un premier ensemble de données.

Figure 54 : Script DataCollector - InitDataCollection

```
1 référence
public void InitDataCollection(float currentFixedTime)
{
    currentLevel = -1;
    levelDatas = new List<LevelPerformanceData>();
    AddLevel(currentFixedTime);
}
```

(Beep-O, 2021)

La fonction AddLevel sert à la création d'un set de données pour un niveau, en initialisant les valeurs qui ont besoin de l'être.

Figure 55 : Script DataCollector - AddLevel

```
1 référence
public void AddLevel(float currentFixedTime)
{
    currentLevel++;
    levelDatas.Add(new LevelPerformanceData(currentFixedTime));
}
```

(Beep-O, 2021)

La property CurrentLevelPerformance permet d'obtenir les performances du joueur sur le niveau en cours, et la property LevelDatas donne un accès en lecture seule sur la liste des données.

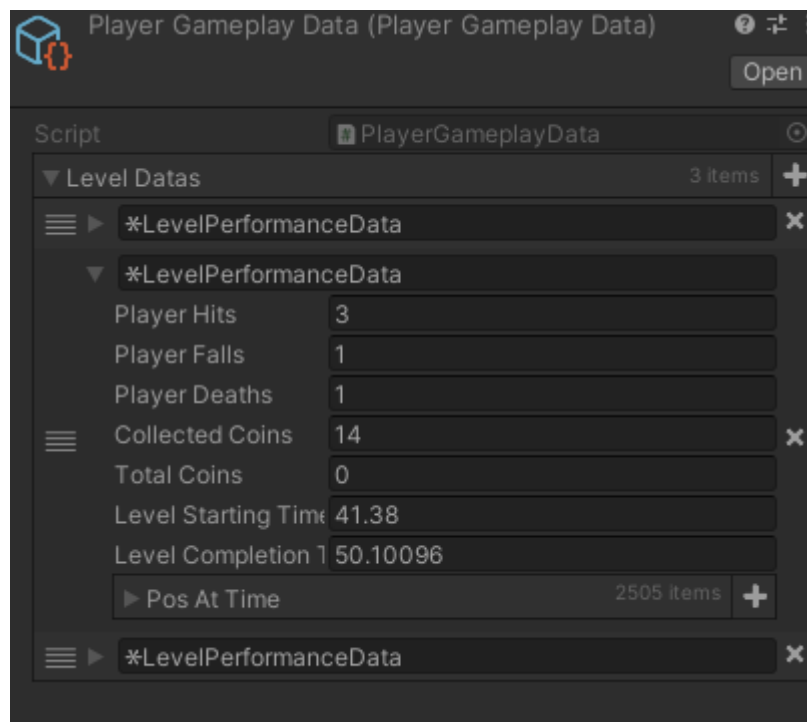
Figure 56 : Script DataCollector – Properties

```
0 références
public LevelPerformanceData CurrentLevelPerformance { get => levelDatas[currentLevel]; }
2 références
public List<LevelPerformanceData> LevelDatas { get => levelDatas; }
```

(Beep-O, 2021)

Cette approche sous forme de scriptable object nous permet de lire les données enregistrées en regardant l'objet PlayerGameplayData lorsque le jeu est lancé sur l'éditeur de Unity.

Figure 57 : Script PlayerGameplayData – vue Inspecteur



(Beep-O, 2021)

DataCollector permet de faire le lien entre PlayerGameplayData et le reste du jeu par le biais de plusieurs méthodes publiques qui peuvent être appelées par divers événements.



Figure 58 : Script DataCollector – Fonctions réponses

```
0 références
public void OnPlayerHit()
{
    playerData.CurrentLevelPerformance.playerHits++;
}
0 références
public void OnPlayerDeath()
{
    playerData.CurrentLevelPerformance.playerDeaths++;
}
0 références
public void OnPlayerFall()
{
    playerData.CurrentLevelPerformance.playerFalls++;
}
0 références
public void OnCoinGathered()
{
    playerData.CurrentLevelPerformance.collectedCoins++;
}
```

(Beep-O, 2021)

Aussi, il permet de prendre en charge la mise à jour des performances liées au temps, telles que le temps de complétion du niveau, et l'enregistrement des positions prises par le joueur.

Figure 59 : Script DataCollector - FixedUpdate

```
© Message Unity | 0 références
private void FixedUpdate()
{
    // Update du completion time
    playerData.CurrentLevelPerformance.levelCompletionTime += Time.fixedDeltaTime;
    // Position at time;
    playerData.CurrentLevelPerformance.posAtTime.Add(Time.fixedTime, playerTransform.position);
}
```

(Beep-O, 2021)

Enfin, cette classe est utilisée par le GameManager, en offrant trois méthodes permettant de gérer la collecte : StartAnalyser, StopAnalyser, et NextLevel.

Figure 60 : Script DataCollector – Fonction de contrôle

```
1 référence
public void StartAnalyser()
{
    playerData.InitDataCollection(Time.fixedTime);
    recording = true;
}

1 référence
public void StopAnalyser()
{
    recording = false;
}

1 référence
public void NextLevel()
{
    playerData.AddLevel(Time.fixedTime);
    recording = true;
}
```

(Beep-O, 2021)

## 6.4 La génération de niveaux

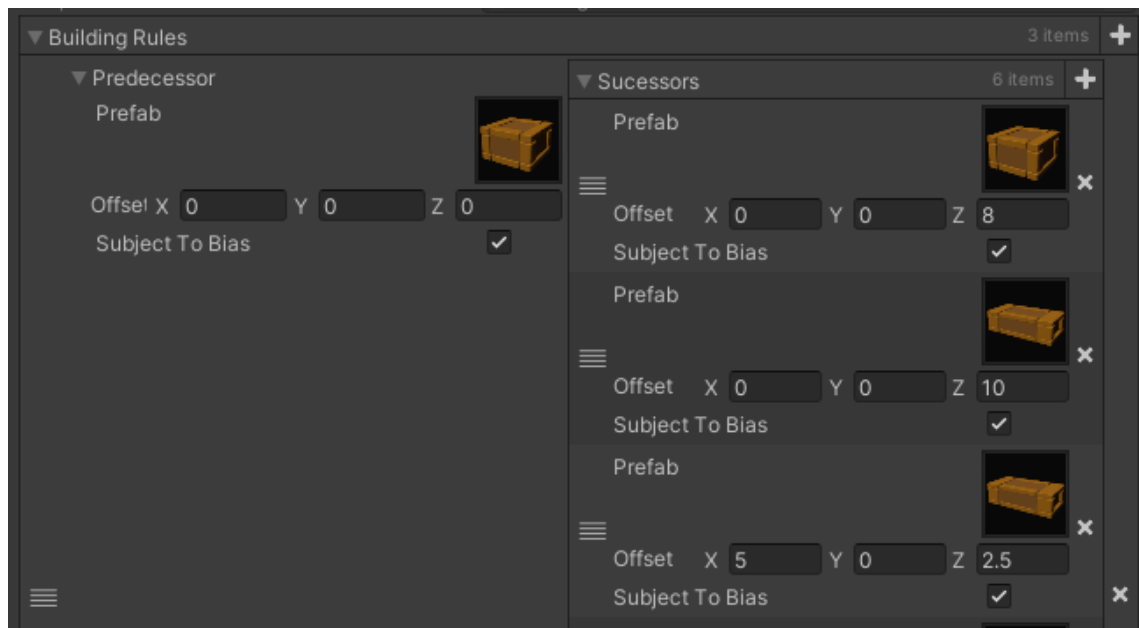
L'approche choisie est relativement simple : on place un ensemble de plateformes par rapport à un ensemble de règles établies, puis sur certaines de ces plateformes on place un obstacle, sur d'autres un ennemi, et celles qui restent une pièce. La quantité de chaque élément est déterminée par les performances du joueur sur les niveaux précédents.

### 6.4.1 Placer les plateformes

Pour placer les plateformes, j'ai choisi une approche similaire à la grammaire générative.

Ainsi, les plateformes sont placées en s'appuyant sur un ensemble de règles. Ces règles sont définies sous forme de liste, et sont composées d'un bloc nommé prédécesseur, et d'une liste de blocs successeurs potentiels.

Figure 61 : Une règle de génération



(Beep-O, 2021)

Chaque bloc est fait du prefab<sup>11</sup> d'une plateforme, d'un vecteur offset, et d'un booléen définissant le bloc comme étant sujet au biais global ou non (je reviens là-dessus plus loin).

Figure 62 : Script BuildingBlock

```

7 références
public class BuildingBlock
{
    [SerializeField]
    public GameObject prefab;
    [LabelWidth(50)]
    [SerializeField]
    private Vector3 offset;
    public bool subjectToBias = true;
    8 références
    public Vector3 Offset { get { return offset; } }
}

```

(Beep-O, 2021)

Lorsque l'on appelle le générateur de niveaux, le premier bloc de la liste est choisi, et son successeur est pris au hasard.

<sup>11</sup> Objets enregistrés dans les dossiers du jeu, utilisés ici pour servir de modèle pour générer des clones

Figure 63 : Script BuldingGenerator – Generate (Partie 1)

```
platforms = new List<Transform>();
BuildingBlock predecessor = platformRuleSet.buildingRules[0].predecessor;

GameObject predecessorClone = Instantiate(predecessor.prefab, transform, false);
```

(Beep-O, 2021)

Ensuite, on place un certain nombre de plateformes, chaque fois en suivant la règle correspondante au dernier bloc placé.

Figure 64 : Script BuldingGenerator – Generate (Partie 2)

```
for(int i = 0; i < platformsCount; i++)
{
    // Sélection de règle
    BuildingRule rule = SelectRule(predecessor.prefab.name);

    // Instanciation conformément à la règle
    BuildingBlock successor = rule.GetSucessor();
    predecessorClone = Instantiate(
        successor.prefab,
        predecessorClone.transform.position + BiasedOffset(successor),
        Quaternion.identity,
        transform);

    // Enregistrer les position des plateformes
    platforms.Add(predecessorClone.transform);
    predecessor = successor;
}
Instantiate(endOfLevel, predecessorClone.transform.position, Quaternion.identity, transform);
```

(Beep-O, 2021)

Le bloc est placé par rapport à son prédécesseur, avec l'offset défini par la règle, en prenant compte du biais global si la règle le stipule.

Figure 65 : Script BuldingGenerator - BiasedOffset

```
1 référence
private Vector3 BiasedOffset(BuildingBlock block)
{
    if (block.subjectToBias)
        return new Vector3(
            block.Offset.x * Random.Range(offsetBiasMin.x, offsetBiasMax.x),
            block.Offset.y * Random.Range(offsetBiasMin.y, offsetBiasMax.y),
            block.Offset.z * Random.Range(offsetBiasMin.z, offsetBiasMax.z)
        );
    else
        return block.Offset;
}
```

(Beep-O, 2021)

Le Transform des plateformes est sauvegardé, afin de pouvoir y placer des obstacles, ennemis et pièces.

Le bloc placé devient alors le prédécesseur pour la boucle suivante, et le bloc de fin de niveau est placé en fin de boucle.

Bien que n'étant pas de la grammaire à proprement parler, par le manque de symboles terminaux et l'impossibilité de générer plus d'un bloc par règle, cette approche convient à la structure choisie pour la construction des niveaux de ce prototype.

### 6.4.2 Placer les obstacles, ennemis et pièces

Une fois les plateformes placées, les obstacles sont disposés selon un ratio déterminé au préalable.

Figure 66 : Script BuldingGenerator - PlaceObstacles

```
private int PlaceObstacles(float ratio)
{
    int generatedObstacles = 0;
    for (int i = platforms.Count-1; i >= 0 ; i--)
    {
        if(Random.Range(0,1f) < ratio)
        {
            int obstacleId = Mathf.Clamp(Random.Range(0, obstacles.Count),0,levelNumber-1);
            Instantiate(
                obstacles[obstacleId],
                platforms[i].transform.position,
                Quaternion.identity,
                transform);
            generatedObstacles++;
            platforms.RemoveAt(i);
        }
    }
    return generatedObstacles;
}
```

(Beep-O, 2021)

De plus, au lieu d'utiliser immédiatement l'ensemble des obstacles disponibles, chaque niveau débloquent un obstacle supplémentaire pour la génération, apportant ainsi un axe de progression supplémentaire. L'apport de nouveauté étant un des points

Les plateformes sur lesquelles sont placés les obstacles sont alors retirées des candidats possibles pour recevoir un ennemi, dont le placement est très similaire, seule la progression n'en faisant pas partie, n'ayant qu'un seul type d'ennemi.

Figure 67 : Script BuldingGenerator - PlaceEnemies

```
1 référence
private int PlaceEnemies(float ratio)
{
    int generatedEnemies = 0;
    for (int i = platforms.Count - 1; i >= 0; i--)
    {
        if (Random.Range(0, 1f) < ratio)
        {
            Instantiate(
                enemies[Random.Range(0, enemies.Count)],
                platforms[i].transform.position,
                Quaternion.identity,
                transform);
            generatedEnemies++;
            platforms.RemoveAt(i);
        }
    }
    return generatedEnemies;
}
```

(Beep-O, 2021)

Enfin, les plateformes restantes reçoivent des pièces. Bien que la fonction inclût la prise en compte d'un ratio, ce ratio est toujours de 1.

Figure 68 : Script BuldingGenerator – PlaceCoins

```
1 référence
private int PlaceCoins(float ratio)
{
    int generatedCoins = 0;
    for (int i = 0; i < platforms.Count; i++)
    {
        if (Random.Range(0, 1f) < ratio)
        {
            Instantiate(coin, platforms[i].transform.position + Vector3.up, Quaternion.identity, transform);
            generatedCoins++;
        }
    }
    return generatedCoins;
}
```

(Beep-O, 2021)

Une fois ces éléments placés, le générateur enregistre dans une liste les informations de ce niveau, prêtes à être utilisées pour définir les paramètres de génération pour les niveaux suivants.

Figure 69 : Script LevelInfos

```
public class LevelInfos
{
    public int platforms;
    public int coins;
    public int enemies;
    public int obstacles;

    1 référence
    public LevelInfos(int platforms, int coins, int enemies, int obstacles)
    {
        this.platforms = platforms;
        this.coins = coins;
        this.enemies = enemies;
        this.obstacles = obstacles;
    }
}
```

(Beep-O, 2021)

### 6.4.3 Définir les paramètres de génération

Pour générer ces niveaux, le générateur définit quatre paramètres. Ces paramètres sont générés à partir des niveaux précédents et des performances du joueur, et influencent la génération.

#### 6.4.3.1 Nombre de plateformes

Le nombre de plateformes est défini comme étant la durée idéale d'un niveau divisé par le temps moyen pour parcourir une plateforme.

Le temps moyen de complétion est récupéré des données enregistrées sur le joueur, soit l'objet PlayerGameplayData.

Figure 70 : Script BuldingGenerator – CalculatePlatformCount (Partie 1)

```
// Temps moyen de complétion
float avgCompletionTime = 0;
for(int i = 0; i < playerData.LevelDatas.Count - 1; i++)
    avgCompletionTime += playerData.LevelDatas[i].levelCompletionTime;

avgCompletionTime /= playerData.LevelDatas.Count-1;
```

(Beep-O, 2021)

Le nombre moyen de plateformes par niveau est quant à lui trouvé à l'aide des informations que le générateur enregistre sur les niveaux précédents.

Figure 71 : Script BuldingGenerator – CalculatePlatformCount (Partie 2)

```
// Nombre moyen de plateformes des niveaux précédents
float avgPlatformCount = 0;
foreach(LevelInfos infos in levelInfos)
    avgPlatformCount += infos.platforms;

avgPlatformCount /= levelInfos.Count;
```

(Beep-O, 2021)

Pour la durée idéale d'un niveau, je n'ai trouvé aucune source donnant un nombre précis. Le consensus semble être que cette durée dépend des jeux, et que seuls des tests permettent de trouver quelle durée convient le mieux, le ressenti servant d'unité de mesure principale.

Ainsi, après avoir testé plusieurs paramètres, 45 secondes par niveaux semble être correct. La nature simple des éléments composant le jeu a tendance à rendre les niveaux plus longs ennuyeux.

Enfin, une augmentation statique est ajoutée, afin d'assurer le suivi de la courbe de tension narrative et ainsi donner une augmentation de difficulté entre les niveaux.

Figure 72 : Script BuldingGenerator – CalculatePlatformCount (Partie 3)

```
// Augmentation statique
return platformCount + staticPlatformIncrease;
}
```

(Beep-O, 2021)

#### 6.4.3.2 Nombre d'obstacles et d'ennemis

Pour les obstacles et les ennemis, le calcul se base sur la densité de ces acteurs sur les niveaux précédents par rapport aux plateformes, auquel une augmentation est ajoutée. Cette augmentation est une augmentation fixe, modérée par le nombre de coups reçus en moyenne dans les niveaux précédents.

Ainsi, chaque niveau aura au minimum autant d'obstacles et d'ennemis que la moyenne des autres niveaux, et une augmentation de la difficulté constante est ajoutée pour conserver un élément de progression d'un niveau à l'autre, tout en prenant en considération les performances des joueurs sous la forme de la moyenne des coups reçus dans les niveaux précédents.



#### **6.4.3.3 Nombre de pièces**

Enfin, les plateformes restantes se voient attribuer une pièce. L'idée initiale était de générer plus ou moins de pièces selon l'intérêt porté par le joueur, mais j'ai dû choisir de laisser tomber cet aspect de la génération en arrivant au terme du temps accordé à ce projet.

Cette idée reste néanmoins intéressante, car elle permettrait d'inclure le profil du joueur selon la taxonomie de Bartles comme paramètre pour la génération, ainsi que de permettre d'avoir un élément simpliste permettant d'identifier le caractère engageant du jeu. Aussi ai-je gardé cet élément.

## 7. Le résultat

Le résultat est un petit jeu très simple, permettant d'avoir un aperçu de ce que permet une approche de génération avec pour objectif de créer du contenu engageant et captivant.

Figure 73 : Début d'un niveau généré par l'algorithme



(Beep-O, 2021)

### 7.1 Améliorations possibles

Comme dit précédemment, avec un mois à consacrer au développement de ce projet, des éléments ont dû être laissés de côté. Voici une liste non exhaustive.

Premièrement, l'utilisation du biais global pour la génération. Le projet final n'utilise pas cette fonction, bien que les premières versions utilisaient le biais pour générer des niveaux visuellement plus distincts, qui semblaient moins attachés à une grille. Cet élément générait cependant parfois des niveaux impossibles à finir, ainsi ai-je choisi de laisser cette idée, l'implémentation de vérification étant quelque chose qui aurait pris beaucoup trop de temps.

Lié à cette suppression, l'utilisation de la position par rapport au temps, ainsi que du décompte des chutes. Ces éléments étaient prévus pour altérer le biais, afin de changer la densité de plateformes d'un niveau. Ne plus utiliser le biais global a rendu ces éléments obsolètes.

Autre élément, le fait de placer des pièces par rapport aux obstacles. Cela aurait nécessité une couche supplémentaire de métadonnées pour prendre en compte les positions possibles pour les pièces pour chaque type d'obstacle, et comme le problème m'est apparu tard dans le développement, j'ai préféré laisser cet élément pour ne pas risquer de compromettre la stabilité relative du projet.

En lien avec ce point, le prototype ne dispose pas de la diversité de contenu nécessaire pour mesurer de façon satisfaisante le caractère engageant. Des systèmes de progression, chemins alternatifs ou niveaux bonus auraient permis d'observer la répartition des interactions entre les joueurs et les différents types de contenu que le jeu propose, mais ces réalisations auraient demandé bien plus de temps que celui à disposition.

Cependant, comme dit plus haut, le pourcentage de pièces ramassées nous donne un élément même simpliste pour prendre en compte cette caractéristique. Le prototype reste suffisamment complet malgré ces éléments manquants, et me donne une base sur laquelle mener des tests.

## **7.2 Phase de tests**

### **7.2.1 La fiche de test**

Afin d'obtenir des retours, j'ai rédigé une fiche posant plusieurs questions.

Certaines de ces questions portent sur le profil de la personne, tel que la tranche d'âge, le temps de jeu hebdomadaire ou si la personne a une expérience préalable avec des jeux de plateforme 3D.

Les autres permettent de détacher un ressenti sur l'expérience du joueur, en demandant si les niveaux étaient trop difficiles ou trop simples, ou si une augmentation de difficulté était ressentie au fur et à mesure de la session de jeu.

Afin de déterminer la capacité de l'algorithme pour générer des niveaux cohérents, j'ai également glissé une question demandant combien de niveaux parmi les 5 premiers étaient faits à la main. La réponse est un seul, le premier, mais les testeurs n'étant pas au courant, cela me permet d'avoir un retour indirect sur la qualité des niveaux générés, et de la proximité entre les capacités de l'algorithme et mes propres compétences en level design.

La fiche complète utilisée pour les tests se trouve en annexe.

### **7.2.2 Les résultats**

Après avoir demandé autour de moi, j'ai réussi à obtenir un échantillon de test de 9 personnes différentes. Les profils sont divers, allant de 5 ans à 60 ans, avec des temps de jeux moyens par semaine très différents. Trois d'entre eux n'avaient jamais joué à un plateformer 3D auparavant.

Pour ce qui est de la question sur le nombre de niveaux générés contre le nombre de niveaux fait main, la plupart ont donné trois niveaux, certains en ayant donné cinq. Chose

à noter, lors des tests, j'ai demandé aux personnes si elles étaient capables de répondre à cette question, et attribué un seul comme réponse à ceux qui m'ont répondu non. Ainsi, parmi les participants, un seul parmi ceux capables de faire la différence a répondu un seul niveau.

Presque tout le monde a ressenti une augmentation légère de la difficulté, bien que plus de la moitié des participants (cinq sur neuf) ont trouvé les niveaux trop simples.

Cependant, la durée des niveaux était de l'avis général de bonne durée, avec deux participants les ayant trouvés trop courts.

Pour ce qui est des chiffres, le nombre de niveaux médians complétés est de 9, soit 4 de plus que le nombre de niveaux demandés, avec un temps de jeu médian de 469 secondes, ou 7m48s. Le pourcentage médian de pièces ramassées est quant à lui de 76%.

Par rapport au temps moyen par niveau, la médiane de celui-ci se trouve à 54,2s, soit 10 secondes de plus que le temps idéal donné. Cependant, le nombre de morts n'est pas pris en compte, bien qu'il ait un impact sur ce chiffre : la médiane des morts étant de 3.

Le tableau complet des résultats des tests se trouve en annexe.

### **7.2.3 Mes observations**

Lors de ces tests, j'ai pu noter une importante différence entre les niveaux générés pour les différents participants : les habitués aux jeux vidéo avaient généralement des niveaux d'une quarantaine de plateformes truffées d'obstacles et d'ennemis, et les néophytes avaient des niveaux plus courts et nettement moins denses en défis.

Cette différence lors de la génération a permis d'amener le temps de complétion médian proche de la durée idéale définie, et ce quel que soit le niveau de compétences. Certains ont trouvé les niveaux trop courts, mais la faute pèse sur ma décision en tant que designer sur la durée idéale d'un niveau, et non sur l'algorithme de génération. De plus, la majorité considère tout de même les niveaux comme étant de bonne durée.

Ainsi, je pense pouvoir dire que ce système permet de créer des niveaux adaptés aux joueurs du point de vue du rythme avec pour but de se rapprocher d'une valeur idéale définie. Cependant, le problème que posent les résultats est que cette valeur idéale ne prend pas en compte le joueur. Cela pourrait être une piste intéressante pour poursuivre cette étude.

Bien que l'avis général des habitués aux jeux vidéo était que le jeu est trop simple, des ajustements visant à corriger ce problème auraient permis d'obtenir un générateur de niveaux offrant un défi intéressant sans être frustrant quel que soit le niveau du joueur.

Pour ce qui est des pièces, bien que le taux d'engagement est élevé, cela peut s'expliquer de par le fait que les ramasser soit un réflexe, plus qu'une activité que les testeurs aient appréciée.

Enfin, dans l'hypothèse où cette série de tests ne marquerait pas la fin de la réalisation de ce prototype, ces tests auraient permis de réaliser des ajustements à l'algorithme de génération, et ainsi je l'espère se rapprocher d'une solution plus consistante, et répondant mieux aux attentes des joueurs. Aussi, bien que ce prototype soit incomplet, l'approche a du potentiel.

### **7.3 Conclusion du travail pratique**

Mon principal regret est que malgré une couverture satisfaisante du caractère captivant du contenu généré par un algorithme, par le biais du temps de jeu et de la quantité de niveaux parcourus, le caractère engageant n'a pu être que superficiellement évalué. Un prototype plus complet, intégrant plus de contenus variés aurait été nécessaire pour pouvoir mesurer plus concrètement cet aspect.

Aussi, les tests réalisés ont permis de voir les imperfections du générateur. Les niveaux sont jugés trop simples par les joueurs réguliers, et le fait que la durée idéale soit définie à l'avance et non adaptée au joueur donne des niveaux qui semblent trop courts pour certaines personnes.

Finalement, ce prototype en tant que tel devient relativement lassant, principalement à cause d'un manque de contenu varié.

Cependant, les tests montrent un bon engagement avec le contenu présent, la plupart jouant plus que les 5 niveaux demandés. Bien que l'algorithme soit relativement simple et les paramètres très vagues, cette approche montre un potentiel intéressant. Surtout, l'adaptation au rythme des joueurs est globalement une réussite, démontrant du potentiel de la génération procédurale pour gérer cet aspect clé pour des jeux engageants et captivants. Neuf personnes parmi mon entourage personnel ne constituent évidemment pas un échantillon suffisant pour pouvoir juger avec exactitude des performances d'un système, mais les résultats restent engageants.

Ainsi, même si ce prototype est imparfait, il permet néanmoins un regard encourageant sur les perspectives offertes par cette approche, et permet de voir que du contenu généré par un algorithme peut être captivant.

## **8. Réponse à la problématique**

Pour rappel, cette étude cherchait à répondre à cette question :

Est-il possible de faire créer par un algorithme du contenu aussi engageant et captivant que du contenu fait par un designer ?

### **8.1 Évolution de mon point de vue**

Dans cette étude, j'ai établi ce qui se fait aujourd'hui dans le domaine de la génération procédurale, ainsi que définis les termes importants de ce sujet. Cette étude préalable aura permis de faire évoluer mon point de vue sur la génération procédurale, ainsi que la réflexion sur le sujet.

En effet, le sujet peut laisser penser que la génération procédurale est utilisée pour remplacer un designer, ou qu'un algorithme est employé pour faire la même chose qu'un designer. Or, la génération de contenu par algorithme ne remplace pas un designer, il s'agit d'un outil à son service.

Dans le cas où la génération procédurale est employée afin de générer d'importantes quantités de contenu rapidement, un ou plusieurs designers seront chargés de donner les règles que cet algorithme devra suivre, et devront effectuer de nombreux ajustements tout au long du développement afin de s'assurer que le contenu généré correspond à leurs attentes.

Ainsi, même si le designer ne crée pas directement les niveaux ou autres éléments, il reste un acteur primordial dans le développement de l'algorithme chargé de le faire, en ayant pour rôle d'établir très clairement les possibilités et limites du système chargé de produire ce contenu.

Un algorithme ne pourra donc générer du contenu engageant et captivant uniquement si le ou les designers responsables de sa conception en sont eux-mêmes capables.

### **8.2 Les forces de la génération procédurale**

La réalisation de ce prototype, bien qu'imparfait, nous aura permis de voir que la génération procédurale peut être employée afin de proposer du contenu adapté au joueur, avec pour but d'être engageant et captivant. La possibilité de faire les niveaux

autour du profil du joueur en temps réel est un outil puissant pour ajuster le rythme et le défi, caractéristiques importantes pour rendre un jeu engageant et captivant.

De plus, même sans cette composante, il y a de nombreux exemples de jeux employant de la génération procédurale ayant rencontré un succès critique et commercial, tels que Darkest Dungeon, Dead Cells ou FTL : Faster Than Light, ou plus généralement les roguelikes.

D'autres titres hors de ce genre utilisent la génération procédurale avec un réel succès, tel que No Man's Sky, qui affichait un temps de jeu moyen de 45 h pour les joueurs de sa précédente mise à jour (Bailey, 2019). Le succès de ces jeux montre que le contenu généré par un algorithme peut être engageant et captivant, certains joueurs leur accordant plusieurs centaines d'heures.

Aussi, le caractère émergent de certaines classes d'approches pour la génération procédurale, telles que la grammaire ou l'optimisation, permet à cette approche de générer des solutions que même les designers ayant conçu ses algorithmes n'avaient pas prévu.

### 8.3 Les limites de la génération procédurale

Cependant, la génération de contenu par un ordinateur a ses limites. La principale d'entre elles provient de l'incapacité pour un algorithme de créer du contenu avec une intention et un sens de façon constante. La créativité est encore un aspect qu'aucune machine n'a réussi à répliquer, et le game design est décrit par certains comme étant autant une science qu'un art.

Le caractère émergent de certaines solutions permet d'émuler la créativité humaine, mais un être humain est encore nécessaire pour produire quelque chose de nouveau. Comme le dit John Smith, Manager à IBM Research (IBM, Date inconnue) :

*« It's easy for AI to come up with something novel just randomly. But it's very hard to come up with something that is novel and unexpected and useful. »*

Aussi, la question de la « définition mathématique d'amusant » (traduit de l'anglais « mathematical definition of fun »), comme le disent Smith et Togelius, est aujourd'hui encore débattue, certains pensant qu'elle existe, d'autres pensent que non (Smith et al., 2015).

Ainsi, aujourd'hui, un algorithme est encore incapable de générer de façon constante des solutions créatives et originales sans l'intervention d'un être humain dans le processus de conception de cet algorithme.

## 8.4 Les options contre ces limites

Je l'ai mentionné brièvement lors de mon état de l'art, mais l'intelligence artificielle a le potentiel de changer la donne. En effet, quelques années auparavant, avoir un ordinateur capable de distinguer un cancer chez un patient semblait impossible, et aujourd'hui cette approche est à l'essai et montre un réel potentiel.

Cependant, la reconnaissance de paterne ne suffit pas encore pour créer du contenu complexe qui soit engageant et captivant, et pour cela il faudra d'abord des intelligences artificielles capables de distinguer parmi des mécaniques de jeux existants celles qui sont engageantes et captivantes de celles qui ne le sont pas.

Selon les mots de Jason Toy, président de Somatic (IBM, date inconnue) :

*« Can we take what humans think is beautiful and creative and try to put that into an algorithm? I don't think it's going to be possible for quite a while. »*



## 9. Conclusion

La question initiale était de savoir s'il était possible ou non de générer du contenu engageant et captivant à partir d'un algorithme, et si ce contenu pouvait être au niveau de contenu fait main par un designer. Dans cette étude, on a donc vu que la génération procédurale permet effectivement de créer du contenu avec lequel les joueurs pourront avoir envie d'interagir et pour longtemps.

Aujourd'hui, ce contenu sera le plus souvent des niveaux, des scénarios ou certains éléments tels que des sons ou des textures, les règles et systèmes de gameplay étant encore difficiles à créer pour des machines. Les algorithmes sont doués pour suivre une structure, mais sont encore incapables d'en créer de manière constante.

Mais les progrès dans le domaine de l'intelligence artificielle laissent percevoir la possibilité d'avoir des jeux entièrement générés de façon procédurale, sans intervention d'un être humain. Les experts ayant des opinions très divergentes sur le sujet, il est impossible de dire avec certitude lorsque cela arrivera, mais le consensus est que cette durée se compte tout de même en années.

Ainsi, un algorithme est capable de générer du contenu qu'un joueur trouvera engageant et captivant, parfois autant que du contenu qu'un designer aura réalisé lui-même, et bien qu'actuellement le type de contenu pouvant être généré reste relativement limité, les récentes avancées dans le domaine de l'intelligence artificielle ouvrent d'intéressantes perspectives sur ce point pour l'avenir.

## Bibliographie

BAILEY, Dustin, 2019. "The average No Man's Sky player now plays the game for 45 hours". *PCGamesn* [en ligne]. 10 juillet 2019 [Consulté le 23 juillet 2021]. Disponible à l'adresse : <https://www.pcgamesn.com/no-mans-sky/hours-played>

BAKKES, Sander, ROIJERS, Diederik, et TRAICHIOIU, Mircea, 2015. *Grammar-based Procedural Content Generation from Designer-provided Difficulty Curves* [en ligne]. 28 Avril 2015 [Consulté le 21 juillet 2021]. Disponible à l'adresse : [https://www.researchgate.net/publication/275539559\\_Grammar-based\\_Procedural\\_Content\\_Generation\\_from\\_Designer-provided\\_Difficulty\\_Curves](https://www.researchgate.net/publication/275539559_Grammar-based_Procedural_Content_Generation_from_Designer-provided_Difficulty_Curves)

BARTLE, Richard, 1996. *HEARTS, CLUBS, DIAMONDS, SPADES: PLAYERS WHO SUIT MUDS* [en ligne]. 1996 [Consulté le 03 septembre 2021]. Disponible à l'adresse : <https://mud.co.uk/richard/hcds.htm>

BEARBOG, Date inconnue. Kraid's Lair Entrance. *Metroid.Fandom.com* [en ligne]. Date inconnue [Consulté le 07 août 2021]. Disponible à l'adresse : [https://metroid.fandom.com/wiki/Kraid%27s\\_Lair?file=Kraid%2527s\\_Lair\\_Entrance.png](https://metroid.fandom.com/wiki/Kraid%27s_Lair?file=Kraid%2527s_Lair_Entrance.png)

BIDARRA, Rafael, 2016. Level generation in Spelunky. *ResearchGate.net* [en ligne]. Octobre 2016. [Consulté le 07 août 2021]. Disponible à l'adresse : [https://www.researchgate.net/figure/Level-generation-in-Spelunky-Adapted-from-10\\_fig3\\_309279824](https://www.researchgate.net/figure/Level-generation-in-Spelunky-Adapted-from-10_fig3_309279824)

BREDA UNIVERSITY OF APPLIED SCIENCES, 2021. *Everything procedural - Conference on procedural content generation for games* [en ligne]. [Consulté le 29 juin 2021]. Disponible à l'adresse : <https://www.everythingprocedural.com/>

BROWN, Mark, 2018. How to Keep Players Engaged (Without Being Evil) [enregistrement vidéo]. Youtube [en ligne]. 6 avril 2018. [Consulté le 24 juillet 2021]. Disponible à l'adresse : [https://www.youtube.com/watch?v=hbzGO\\_Qonu0](https://www.youtube.com/watch?v=hbzGO_Qonu0)

BROWN, Mark, 2019. Roguelikes, Persistence and Progression | Game Maker's Toolkit [enregistrement vidéo]. *YouTube* [en ligne]. 28 janvier 2019. [Consulté le 30 juin 2021]. Disponible à l'adresse : <https://www.youtube.com/watch?v=G9FB5R4wVno>

BROWN, Mark, 2020. This Psychological Trick Makes Rewards Backfire [enregistrement vidéo]. *Youtube* [en ligne]. 13 septembre 2020. [Consulté le 24 juillet 2021]. Disponible à l'adresse : <https://www.youtube.com/watch?v=K0M1PuQaE8s>

BUCKLEW, Brian, et GRINBLAT, Jason, 2019. Math for Game Developers: End-to-End Procedural Generation in Caves of Qud [enregistrement vidéo]. *YouTube* [en ligne]. 23 juin 2021. [Consulté le 28 juin 2021]. Disponible à l'adresse : <https://www.youtube.com/watch?v=jV-DZqdKlnE&t=2s>

DEBYL, Penny, 2018. The Theory of Noise : An Overview of Perlin Noise [enregistrement vidéo]. *YouTube* [en ligne]. 27 mai 2018. [Consulté le 02 juillet 2021]. Disponible à l'adresse : <https://www.youtube.com/watch?v=H6FhG9VKhJg>

COMPTON, Kate, 2017. Practical Procedural Generation for Everyone [enregistrement vidéo]. *YouTube* [en ligne]. 31 mai 2017. [Consulté le 29 juin 2021]. Disponible à l'adresse : <https://www.youtube.com/watch?v=WumyLEa6bU>

CD Projekt Red, 2020. Cyberpunk 2077 - Dans les coulisses : JALI [enregistrement vidéo]. *Youtube* [en ligne]. novembre 2020. [Consulté le 16 juillet 2021]. Disponible à l'adresse : [https://www.youtube.com/watch?v=fa3\\_Mfqu8KA](https://www.youtube.com/watch?v=fa3_Mfqu8KA)

EVOLUTIONARY GAMES, 2014. Galactic Arms Race sur Steam. *Store.Steampowered.com* [en ligne]. 31 mai 2014 [Consulté le 21 août 2021]. Disponible à l'adresse : [https://store.steampowered.com/app/249610/Galactic\\_Arms\\_Race/?l=french](https://store.steampowered.com/app/249610/Galactic_Arms_Race/?l=french)

GIESEN, Fabian, 2012. Metaprogramming for madmen. *The ryg blog* [en ligne]. 08 avril 2012. [Consulté le 05 juillet 2021]. Disponible à l'adresse : <https://fgiesen.wordpress.com/2012/04/08/metaprogramming-for-madmen/>

GYGAX, Gary et ARNESON, Dave, 1974. *Dungeons & Dragons*. Lake Geneva : TSR, Inc.

HAGGIS, Mata, 2017. Storytelling Tools to Boost Your Indie Game's Narrative and Gameplay [enregistrement vidéo]. *YouTube* [en ligne]. 23 juin 2017. [Consulté le 17 juillet 2021]. Disponible à l'adresse : <https://www.youtube.com/watch?v=8fXE-E1hjKk>

HAIRBALL, 2020. 3D All-Stars Tip: How to Long Jump in Super Mario Sunshine. *Super Mario 128 Central* [en ligne]. 19 septembre 2020 [Consulté le 16 août 2021]. Disponible à l'adresse : <https://www.sm128c.com/super-mario-sunshine-long-jump-0208>

HARTE, Thomas, 2007. The flight segment of the game Elite for the BBC Micro. *Wikipedia.org* [en ligne]. 20 août 2007 [Consulté le 05 septembre 2021]. Disponible à l'adresse : [https://en.wikipedia.org/wiki/File:BBC\\_Micro\\_Elite\\_screenshot.png](https://en.wikipedia.org/wiki/File:BBC_Micro_Elite_screenshot.png)

HASTINGS, Erin J., GUHA, Ratan K., et STANLEY, Kenneth O., 2009. *Evolving Content in the Galactic Arms Race Video Game* [en ligne]. 2009. [Consulté le 16 août 2021]. Disponible à l'adresse : [http://eplex.cs.ucf.edu/papers/hastings\\_ciq09.pdf](http://eplex.cs.ucf.edu/papers/hastings_ciq09.pdf)

HENDRIKX, Mark, MEIJER, Sebastiaan, VAN DER VELDEN, Joeri, et IOSUP, Alexandru, 2011. *Procedural Content Generation for Games: A Survey* [en ligne]. février 2013 [Consulté le 01 juillet 2021]. Disponible à l'adresse :

[https://www.researchgate.net/publication/262327212\\_Procedural\\_Content\\_Generation\\_for\\_Games\\_A\\_Survey](https://www.researchgate.net/publication/262327212_Procedural_Content_Generation_for_Games_A_Survey)

HESS, Brian, 2018. Plot and Parcel: Procedural Level Design in XCOM 2 [enregistrement vidéo]. *YouTube* [en ligne]. 26 février 2020. [Consulté le 05 juillet 2021]. Disponible à l'adresse : <https://www.youtube.com/watch?v=5jrQ5rDI4dk>

HILLER, Lejaren A. Jr., et ISAACSON, Leonard M, 1959. *Experimental Music composition with an electronic computer* [en ligne]. 2e éd. New York : McGraw-Hill ISBN 978-0-313-22158-3. Disponible à l'adresse : <https://archive.org/details/experimentalmusi00hill/mode/2up>

HWANG, Rachel, 2017. SHAPE GRAMMARS procedural generation techniques for virtual cities [en ligne]. 2017. [Consulté le 07 août 2021]. Disponible à l'adresse : [https://cis700-procedural-graphics.github.io/files/shape\\_grammar\\_2\\_7\\_17.pdf](https://cis700-procedural-graphics.github.io/files/shape_grammar_2_7_17.pdf)

IBM, Date inconnue. The quest for AI creativity. *IBM.com* [en ligne]. Date inconnue [Consulté le 12 septembre 2021]. Disponible à l'adresse : <https://www.ibm.com/watson/advantage-reports/future-of-artificial-intelligence/ai-creativity.html>

INTERACTIVE DATA VISUALIZATION, 2017. *SpeedTree* [en ligne]. 2020 [Consulté le 02 juillet 2021]. Disponible à l'adresse : <https://store.speedtree.com/games/>

INVENTIV-IT, 2019. INTELLIGENCE ARTIFICIELLE FAIBLE : DE QUOI S'AGIT-IL ?. *Inventiv-it.fr* [en ligne]. 15 mai 2019. [Consulté le 21 août 2021]. Disponible à l'adresse : <https://inventiv-it.fr/intelligence-artificielle-faible-de-quoi-sagit-il/>

JISHENAZ, 2013. XboxController. *Wikipedia.org* [en ligne]. 7 septembre 2013 [Consulté le 28 août 2021]. Disponible à l'adresse : [https://commons.wikimedia.org/wiki/File:Xbox\\_Controller.svg](https://commons.wikimedia.org/wiki/File:Xbox_Controller.svg)

KAPPEN, Dennis L., 2016. Categorization of Intrinsic and Extrinsic Motivations based on the Kaleidoscope of Effective Gamification. *ResearchGate.net* [en ligne]. Janvier 2016 [Consulté le 07 août 2021]. Disponible à l'adresse : [https://www.researchgate.net/figure/Categorization-of-Intrinsic-and-Extrinsic-Motivations-based-on-the-Kaleidoscope-of\\_fig1\\_289130418](https://www.researchgate.net/figure/Categorization-of-Intrinsic-and-Extrinsic-Motivations-based-on-the-Kaleidoscope-of_fig1_289130418)

KASTBJERG, Emil, SCHEDL, David, TOGELIUS, Julian, et YANNAKAKIS, Georgios N., 2011. *What is Procedural Content Generation? Mario on the borderline* [en ligne]. janvier 2011. [Consulté le 01 juillet 2021]. Disponible à l'adresse : [https://www.researchgate.net/publication/228620622\\_What\\_is\\_Procedural\\_Content\\_Generation\\_Mario\\_on\\_the\\_borderline](https://www.researchgate.net/publication/228620622_What_is_Procedural_Content_Generation_Mario_on_the_borderline)

KAZEMI, Darius, 2019. "How to effectively use procedural generation in games". *Gamasutra* [en ligne]. 10 avril 2019. [Consulté le 29 juin 2021]. Disponible à l'adresse : [https://www.gamasutra.com/view/news/340190/How\\_to\\_effectively\\_use\\_procedural\\_generation\\_in\\_games.php](https://www.gamasutra.com/view/news/340190/How_to_effectively_use_procedural_generation_in_games.php)

LAROUSSE, 2021. Définitions : engageant, engageante - Dictionnaire de français Larousse. *Larousse.fr* [en ligne]. Date inconnue. [Consulté le 17 juillet 2021]. Disponible à l'adresse : <https://www.larousse.fr/dictionnaires/francais/engageant/29508>

LAROUSSE, 2021. Définitions : captiver - Dictionnaire de français Larousse. *Larousse.fr* [en ligne]. Date inconnue. [Consulté le 17 juillet 2021]. Disponible à l'adresse : <https://www.larousse.fr/dictionnaires/francais/captiver/13021>

LAVENDER, Becky, et THOMPSON, Tommy, 2015. *The Zelda Dungeon Generator: Adopting Generative Grammars to Create Levels for Action-Adventure Games* [en ligne]. 23 avril 2015 [Consulté le 05 juillet 2021]. Disponible à l'adresse : [http://beckylavender.co.uk/wp-content/uploads/2017/11/ZDG\\_Dissertation.pdf](http://beckylavender.co.uk/wp-content/uploads/2017/11/ZDG_Dissertation.pdf)

LOPEZ, Mike, 2008. Gameplay Fundamentals Revisited: Harnessed Pacing & Intensity. *Gamedeveloper.com* [en ligne]. 12 novembre 2008. [Consulté le 03 août 2021]. Disponible à l'adresse : <https://www.gamedeveloper.com/design/gameplay-fundamentals-revisited-harnessed-pacing-intensity>

MACH, Michal, 2017. Physics Animation in Uncharted 4 : A Thief's End [enregistrement vidéo]. *YouTube* [en ligne]. 26 décembre 2018. [Consulté le 05 juillet 2021]. Disponible à l'adresse : <https://www.youtube.com/watch?v=7S-vuoKgR4>

MALETZ, David, 2012. « Four Tricks to Improve Game Balance. » *Gamedeveloper.com* [en ligne]. 14 septembre 2012. [Consulté le 03 août 2021]. Disponible à l'adresse : [https://www.gamasutra.com/blogs/DavidMaletz/20120913/177683/Four\\_Tricks\\_to\\_Improve\\_Game\\_Balance.php?print=1](https://www.gamasutra.com/blogs/DavidMaletz/20120913/177683/Four_Tricks_to_Improve_Game_Balance.php?print=1)

MCKENDRICK, Innes, 2017. Continuous World Generation in No Man's Sky [enregistrement vidéo]. *YouTube* [en ligne]. 9 août 2017. [Consulté le 29 juin 2021]. Disponible à l'adresse : <https://www.youtube.com/watch?v=sCRzxEEcO2Y>

NIERHAUS, Gerhard, 2009. *Algorithmic Composition - Paradigms of Automated Music Generation*. Vienne : Springer Vienna. ISBN 978-3-211-77539-6.

PECKHAM, Matt, Date inconnue. "NO MAN'S SKY IS WILDLY AMBITIOUS, UTTERLY VAST AND A HUGE CHALLENGE TO THE VIDEO GAME INDUSTRY'S STATUS QUO". *Time* [en ligne]. Date inconnue. [Consulté le 05 juillet 2021]. Disponible à l'adresse : <https://time.com/no-mans-sky/>

RAMIROMAGALHAES, 2013. Character theory chart. *Wikipedia.org* [en ligne]. 13 mars 2013 [Consulté le 03 septembre 2021]. Disponible à l'adresse : [https://en.wikipedia.org/wiki/File:Character\\_theory\\_chart.svg](https://en.wikipedia.org/wiki/File:Character_theory_chart.svg)

POLYMORPHIC GAMES, 2019. Project Hastur sur Steam. *Store.Steampowered.com* [en ligne]. 23 mai 2019 [Consulté le 07 août 2021]. Disponible à l'adresse : [https://store.steampowered.com/app/800700/Project\\_Hastur/](https://store.steampowered.com/app/800700/Project_Hastur/)

RICHTER, Felix, 2020. Gaming: The Most Lucrative Entertainment Industry By Far. *Statista* [en ligne]. 22 septembre 2020. [Consulté le 02 Juillet 2021]. Disponible à l'adresse : <https://www.statista.com/chart/22392/global-revenue-of-selected-entertainment-industry-sectors/>

ROGUEBASIN, 2013. *Berlin Interpretation* [en ligne]. [Consulté le 30 juin 2021]. Disponible à l'adresse : [http://www.roguebasin.com/index.php?title=Berlin\\_Interpretation](http://www.roguebasin.com/index.php?title=Berlin_Interpretation)

ROSEN, David, 2014. Animation Bootcamp: An Indie Approach to Procedural Animation [enregistrement vidéo]. *YouTube* [en ligne]. 21 octobre 2017. [Consulté le 05 juillet 2021]. Disponible à l'adresse : <https://www.youtube.com/watch?v=LNidsMesxSE&t=126s>

RIOT GAMES, 2018. So You Wanna Make Games?? | Episode 8: Sound Design [enregistrement vidéo]. *YouTube* [en ligne]. 13 décembre 2018. [Consulté le 05 juillet 2021]. Disponible à l'adresse : <https://www.youtube.com/watch?v=KcorlwJscFA>

SCHILLER, Pierre, 2020. @3D\_director. Magical ICE ocean details. Fully procedural... *Post Twitter* [en ligne]. 05 octobre 2020, 3:20. [Consulté le 12 juillet 2021]. Disponible à l'adresse : [https://twitter.com/3D\\_director/status/1312925504530505730](https://twitter.com/3D_director/status/1312925504530505730)

SHAKER, Noor, TOGELIUS, Julian, et NELSON, Mark J., 2016. *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer. ISBN 978-3-319-42714-0.

SMITH, Gillian, STURTEVANT, Nathan, et TOGELIUS, Julian, 2015. Making Things Up: The Power and Peril of PCG [enregistrement vidéo]. *YouTube* [en ligne]. 29 mars 2018. [Consulté le 05 juillet 2021]. Disponible à l'adresse : <https://www.youtube.com/watch?v=B11RIHZsmGE>

SpeedTree, 2012. SpeedTree Tutorial: Modeler Basics [enregistrement vidéo]. *Youtube* [en ligne]. 17 avril 2012. [Consulté le 02 juillet 2021]. Disponible à l'adresse : [https://www.youtube.com/watch?v=sbmQhdQ\\_2VI](https://www.youtube.com/watch?v=sbmQhdQ_2VI)

SPUFFORD, Francis, 2003. "Masters of their universe". *The Guardian* [en ligne]. 18 octobre 2003. [Consulté le 05 juillet 2021]. Disponible à l'adresse : <https://www.theguardian.com/books/2003/oct/18/features.weekend>

STALBERG, Oskar, 2018. EPC2018 - Oskar Stalberg - Wave Function Collapse in Bad North [enregistrement vidéo]. *YouTube* [en ligne]. 11 juillet 2018. [Consulté le 29 juin 2021]. Disponible à l'adresse : <https://www.youtube.com/watch?v=0bcZb-SsnrA&list=PLPdLkAMVHwZ7Z2OK3X7IZoAOHat4EdEaS&index=5>

STARDEW VALLEY, 2016. Farm day. *Stardewvalley.net* [en ligne]. 2016 [Consulté le 12 juillet 2021]. Disponible à l'adresse : [https://www.stardewvalley.net/presskit/Stardew%20Valley/images/SDV\\_iPad\\_01.png](https://www.stardewvalley.net/presskit/Stardew%20Valley/images/SDV_iPad_01.png)

STEAMCHARTS, 2020. No Man's Sky. *Steamcharts* [en ligne]. 24 juillet 2021 [Consulté le 24 juillet 2021]. Disponible à l'adresse : <https://steamcharts.com/app/275850>

STOLFI, Jorge, 2013. A "white noise" image. *Wikipédia : l'encyclopédie libre* [en ligne]. 12 février 2013. [Consulté le 02 juillet 2021]. Disponible à l'adresse : [https://en.wikipedia.org/wiki/White\\_noise#/media/File:White-noise-mv255-240x180.png](https://en.wikipedia.org/wiki/White_noise#/media/File:White-noise-mv255-240x180.png)

SUMMERVILLE, Adam, SHWETA, Philip, et MATEAS, Michael, 2015. *Monte Carlo Tree Search to Guide Platformer Level Generation* [en ligne]. 2015. [Consulté le 04 septembre 2021]. Disponible à l'adresse : <https://www.aaai.org/ocs/index.php/AIIDE/AIIDE15/paper/view/11569/11396>

THOMPSON, Tommy, 2020. Director AI for Balancing In-Game Experiences | AI 101 [enregistrement vidéo]. *Youtube* [en ligne]. 6 avril 2018. [Consulté le 24 juillet 2021]. Disponible à l'adresse : <https://www.youtube.com/watch?v=Mnt5zxb8W0Y>

YEUNG, Simon, 2012. Inverse Kinematics (two joints) for foot placement. *Gamedeveloper.com* [en ligne]. 20 janvier 2012. [Consulté le 24 juillet 2021]. Disponible à l'adresse : <https://www.gamedeveloper.com/programming/inverse-kinematics-two-joints-for-foot-placement>

ZUBEK, Robert, 2020. *Elements of Game Design*. Cambridge : MIT Press. ISBN 9780262043915

## Annexe 1 : Fiche de Test

### Instructions

Ce test devrait être assez rapide, merci de participer

Pour choisir une réponse, supprimez les autres, par exemple :

Avez-vous déjà joué à un jeu de plateforme en 3D ? Par exemple Mario 64/Sunshine/Galaxy/Odyssey, A Hat in Time, Banjo Kazooie ?	
Oui	

Je vous demanderai de bien vouloir jouer au moins 5 niveaux. Libre à vous de continuer au-delà si le jeu vous plaît.

Prenez connaissance des questions jusqu'à la mention "JOUEZ", puis n'oubliez pas de lire les questions suivantes après votre session !

### Votre profil

Nom et prénom

Avez-vous déjà joué à un jeu de plateforme en 3D ? Par exemple Mario 64/Sunshine/Galaxy/Odyssey, A Hat in Time, Banjo Kazooie ?	
Oui	Non

Quel est votre temps de jeu moyen par semaine ?
Moins de 3h
Entre 3h et 10h
Plus de 10h



Quelle est votre tranche d'âge ?
Moins de 15 ans
Entre 15 et 30 ans
Plus de 30 ans

Votre expérience avec les aventures procédurales de Beep-O :

Globalement, les niveaux étaient :
Trop courts
De bonne durée
Trop longs


Globalement, les niveaux étaient :
Trop faciles
Un défi convenable
Trop difficiles

Sur les 5 premiers niveaux, j'en ai réalisé plusieurs à la main : combien d'après vous ?				
1	2	3	4	5

# JOUEZ

Avez-vous ressenti une augmentation de difficulté d'un niveau à l'autre :
Pas du tout
Un peu
Suffisamment
Beaucoup trop

Nombre de niveaux joués :
5 ou moins
Entre 5 et 10
Entre 10 et 20
Plus de 20

Le screenshot de fin, appuyez sur start et prenez une capture d'écran :


Merci d'avoir participé !

## Annexe 2 : Résultats des Tests

Profil				Avis				Données				
Nom & Prénom	Expérience	Jeu par semaine	Tranche d'âge	Durée des Niveaux	Difficulté des Niveaux	Niveaux fait main	Augmentation de difficulté	Niveaux	Temps de jeu (s)	Pièces (%)	Morts	Temps/lvl
Jean-Marc Clerc	Non	Moins de 3h	Plus de 30 ans	De bonne durée	Un défi convenable	1	Un peu	2	221	0,86	3	110,5
Nadine Clerc	Non	Moins de 3h	Plus de 30 ans	Trop courts	Un défi convenable	1	Suffisamment	3	234	0,83	3	78
Tim Degardin	Non	Entre 3h et 10h	Moins de 15 ans	De bonne durée	Trop faciles	1	Pas du tout	12	612	0,12	23	51
Morgane Linder	Oui	Entre 3h et 10h	Entre 15 et 30 ans	Trop courts	Trop faciles	1	Un peu	9	469	0,56	2	52,11111111
Loirella Linder-Pigliapochi	Oui	Plus de 10h	Plus de 30 ans	De bonne durée	Un défi convenable	5	Suffisamment	15	1206	0,23	20	80,4
Désirée Linder	Oui	Plus de 10h	Entre 15 et 30 ans	De bonne durée	Trop faciles	3	Un peu	7	326	0,76	1	46,57142857
Cyrielle Clerc	Oui	Entre 3h et 10h	Entre 15 et 30 ans	De bonne durée	Un défi convenable	1	Un peu	6	300	0,91	3	50
Kristopher Degardin	Oui	Entre 3h et 10h	Entre 15 et 30 ans	De bonne durée	Trop faciles	3	Un peu	10	542	0,96	2	54,2
François Bouladou	Oui	Plus de 10h	Entre 15 et 30 ans	De bonne durée	Trop faciles	3	Un peu	24	1608	0,34	4	67