Chapitre 2

Génération de musique

Ce chapitre est divisé en trois sections. La première présente quelques algorithmes potentiels pour la composition de musique pour les jeux vidéo. La deuxième présente une sélection d'outils de composition. Plusieurs des outils présentés permettent la composition algorithmique et certains visent spécialement l'industrie du jeu vidéo. Enfin, nous discuterons des limitations de chacun des algorithmes et des outils mentionnés de sorte à introduire les chapitres à venir.

2.1 Algorithmes de composition

Nous débutons cette section en présentant l'un des premiers algorithmes de composition automatique. Il s'agit des dés musicaux introduits par *Johann Philipp Kirnberger* [27, 34] vers la fin du 18e siècle. Puis, nous présenterons la composition à l'aide des chaînes de Markov qui sont un processus stochastique très utilisé en composition algorithmique. Ensuite, nous présenterons les L-systèmes qui sont un type particulier de grammaires avec lequel nous avons fait des expériences. Nous poursuivrons en introduisant les automates cellulaires et les algorithmes génétiques puisqu'ils sont très populaires dans le domaine de la composition algorithmique. Enfin, nous ferons une présentation détaillée des automates finis puisqu'ils sont à la base du modèle que nous présenterons au chapitre 3 (plus spécifiquement notre modèle est basé sur les automates finis probabilistes et sur les automates étendus). De plus, ils ont la propriété intéressante de pouvoir généraliser plusieurs des modèles déjà présentés.

2.1.1 Dés musicaux

Les jeux de dés musicaux figurent parmi les premiers algorithmes de composition répertoriés comme tels. Ils furent très populaires vers la fin du 18e siècle comme en témoignent la vingtaine d'autres jeux de dés musicaux qui furent publiés suite à celui de *Kirnberger*. Parmi ceux-ci, certains furent même attribués à des compositeurs célèbres tels que : *C.P.E. Bach*, *Haydn* et *Mozart*. D'après *Noguchi*[1] [47], il semble toutefois plausible que *Mozart* ne soit pas l'auteur des jeux de dés qui lui sont attribués. Il existe toutefois un authentique jeu musical autographié par *Mozart* et archivé à la Bibliothèque Nationale de Paris. Ce dernier est basé sur les prénoms.

Afin d'illustrer le fonctionnement de ces jeux de dés musicaux, prenons la méthode originale de *Kirnberger* [34] pour générer des polonaises, des menuets et des trios qui sont trois formes de pièces musicales. L'ouvrage original[2] de *Kirnberger* comportait six tableaux de correspondance, deux pour chaque type de pièces, et trois ensembles de segments musicaux, un pour chaque type de pièces. Grâce à ces outils, le joueur peut composer une pièce en procédant comme suit :

1. Pour chaque ligne du premier tableau :
   (a) Le joueur lance un dé à six faces ou, optionnellement, dans le cas des polonaises, deux dés à six faces (le livre original de *Kirnberger* présente deux correspondances possibles pour les polonaises [34]).
   (b) Le joueur trouve le numéro de segment indiqué à l'intersection de cette ligne et de la valeur indiquée par le ou les dés.
   (c) Enfin, il transcrit le segment musical correspondant à ce numéro.
2. Le joueur répète ensuite la procédure en utilisant le deuxième tableau.

Le lecteur pourra trouver un exemple d'application de la procédure en consultant [31], une traduction partielle[3] du manuscrit original qui ajoute deux exemples d'application, inexistants dans l'original.

[1] `http://www.asahi-net.or.jp/~rb5h-ngc/e/k516f.htm`
[2] `http://www.musikwissenschaft.uni-mainz.de/Musikinformatik/schriftenreihe/nr15/Kirnframe.html`
[3] `http://icking-music-archive.org/scores/kirnberger/menuet.pdf`

Pour les jeux de dés subséquents, la procédure est aussi simple puisqu'elle est généralement basée sur celle de *Kirnberger* à quelques exceptions près. Il va de soi que certains jeux ont introduit leurs propres variations dans la procédure, sans toutefois en changer la nature profonde. Il existe cependant une extension intéressante développée récemment par *Langston* [37] pour le jeu *Ballblazer* de *Lucasfilm Games*[4]. Dans cette extension, chaque mesure, avant d'être jouée par l'ordinateur (ou retranscrite), est filtrée par un algorithme qui peut effectuer les transformations suivantes sur celle-ci :

1. Doubler la durée de toutes les notes de la mesure,
2. Diminuer de moitié la durée de toutes les notes de la mesure, et
3. Supprimer certaines notes de la mesure en les transformant en silence ou en prolongeant la durée de la note précédente.

Pour déterminer les transformations à appliquer, l'algorithme de *Langston* utilise un ensemble de règles bien précis que le lecteur pourra trouver dans [37].

2.1.2 Chaînes de Markov

Il existe plusieurs types de chaînes de Markov. Toutefois, dans cette introduction, nous n'allons nous intéresser qu'aux chaînes de Markov homogènes à temps discret (ci-après : chaînes de Markov) puisque ce sont les plus utilisées en composition algorithmique. Le lecteur pourra trouver une description plus approfondie du sujet dans des livres comme ceux de *Foata et Fuchs* [16] et de *Isaacson et Madsen* [32].

Définition 2.1 (*Fonction totale*)
En mathématiques, une *fonction totale* est une fonction $f\colon X \to Y$ qui associe chaque élément de l'ensemble X (appelé domaine de la fonction) à *exactement* un élément de l'ensemble Y (appelé co-domaine de la fonction). Chaque élément du co-domaine peut avoir zéro, un ou plusieurs antécédents dans le domaine.

[4] Ancien nom de LucasArts.

2.1.2.1 Définition

Une chaîne de Markov est un processus stochastique à temps discret, c'est-à-dire une suite de variables X_n où $n \in \mathbb{N}$, respectant la propriété Markovienne (équation 2.1).

$$Pr(X_{n+1} = j \mid X_n = i, X_{n-1} = i_{n-1}, \ldots, X_0 = i_0) = Pr(X_{n+1} = j \mid X_n = i) \qquad (2.1)$$

En d'autres mots, l'état du processus au temps $(n+1)$ ne dépend que de son état au temps n, mais non pas de ses états antérieurs. On dit d'un tel processus qu'il est une chaîne de Markov de premier ordre puisque seul le dernier état visité conditionne l'état présent et qu'il respecte la propriété de chaîne, c'est-à-dire que le nombre d'états est fini. Formellement, les chaînes de Markov sont définies comme suit :

Soit S un ensemble fini d'états, $\Pi \in \mathbb{R}^{|S|}$ un vecteur stochastique, et $\mathcal{P}$ une fonction totale de $S \times S$ dans $\mathbb{R}$. Le triplet $\langle S, \mathcal{P}, \Pi \rangle$ représente alors une chaîne de Markov ayant S comme ensemble d'états possibles, Π pour loi initiale, et $\mathcal{P}$ comme matrice de transition.

La matrice de transition spécifie la probabilité de transition $Pr(j \mid i)$ entre chaque paire d'états $\langle i, j \rangle \in S^2$. Elle possède deux dimensions et doit satisfaire les conditions suivantes :

- Pour toute paire $\langle i, j \rangle$, on a $0 \leq Pr(j \mid i) \leq 1$
- Pour tout $i \in S, \sum_{j \in S} Pr(j \mid i) = 1$

On qualifiera une matrice de transition d'homogène dans le temps si les probabilités de transition ne sont pas dépendantes du temps n. La figure 2.1 donne un exemple de matrice de transition non-homogène. Nous ne considérerons toutefois pas ce genre de matrices de transition puisqu'elles ne sont pratiquement jamais utilisées en composition algorithmique.

	a	**b**
a	$1/n$	$1 - 1/n$
b	1	0

FIG. 2.1 – Matrice de transition non-homogène

La loi initiale, quant à elle, est représentée par un vecteur stochastique $\Pi \in \mathbb{R}^{|S|}$ et spécifie la probabilité que la chaîne démarre dans chaque état.

De façon formelle, la loi initiale devient :

$$\sum_{i \in S} \Pi_i = 1 \tag{2.2}$$

$$\Pi_i = Pr(X_0 = i) \tag{2.3}$$

La figure 2.2 représente un exemple de chaîne de Markov appliqué à la composition.

$$S = \{Do, Ré, Mi\}$$

$$\mathcal{P} = \begin{array}{|c|c|c|c|} \hline & \textbf{Do} & \textbf{Ré} & \textbf{Mi} \\ \hline \textbf{Do} & 0.25 & 0.35 & 0.40 \\ \hline \textbf{Ré} & 0.00 & 0.50 & 0.50 \\ \hline \textbf{Mi} & 0.30 & 0.20 & 0.50 \\ \hline \end{array}$$

$$\Pi = \begin{array}{|c|c|c|} \hline \textbf{Do} & \textbf{Ré} & \textbf{Mi} \\ \hline 0.67 & 0.33 & 0.00 \\ \hline \end{array}$$

FIG. 2.2 – Exemple de chaîne de Markov

2.1.2.2 Chaînes de Markov d'ordre k

Il est utile de définir une généralisation des chaînes de Markov qui permet de conditionner celles-ci par plusieurs états antérieurs. On nomme alors *ordre de la chaîne de Markov* le nombre d'états passés à prendre en compte pour déterminer son état futur. Ainsi donc, les événements dans une chaîne de premier ordre ne sont dépendants que de leur prédécesseur immédiat, ceux dans une chaîne de second ordre ne sont dépendants que de leur deux plus proches prédécesseurs, et ainsi de suite jusqu'à l'ordre k. Nous appellerons ces chaînes *chaînes de Markov d'ordre k* où k représente un nombre entier positif plus grand ou égal à un. On obtient les propriétés de celles-ci en généralisant les propriétés des chaînes de Markov de premier ordre. Ainsi donc, la propriété Markovienne devient :

$$\begin{aligned} &Pr(X_{n+k} = j \mid X_{n+(k-1)} = i_{n+(k-1)}, X_{n+(k-2)} = i_{n+(k-2)}, \ldots, X_0 = i_0) \\ &\quad = Pr(X_{n+k} = j \mid X_{n+(k-1)} = i_{n+(k-1)}, X_{n+(k-2)} = i_{n+(k-2)}, \ldots, X_n = i_n) \end{aligned} \tag{2.4}$$

$$S = \{Do, Ré\}$$

$$\mathcal{P} = \begin{array}{|c|c|c|}\hline & \textbf{Do} & \textbf{Ré} \\ \hline \textbf{Do Do} & 0.45 & 0.55 \\ \hline \textbf{Do Ré} & 0.25 & 0.75 \\ \hline \textbf{Ré Do} & 0.00 & 1.00 \\ \hline \textbf{Ré Ré} & 0.50 & 0.50 \\ \hline \end{array}$$

$$\Pi = \begin{array}{|c|c|c|c|}\hline \textbf{Do Do} & \textbf{Do Ré} & \textbf{Ré Do} & \textbf{Ré Ré} \\ \hline 0.31 & 0.11 & 0.27 & 0.31 \\ \hline \end{array}$$

FIG. 2.3 – Exemple de chaîne de Markov d'ordre deux

De plus, on remplace la matrice de transition par une matrice à $(k+1)$ dimensions qui doit respecter les mêmes propriétés qu'une matrice de transition de premier ordre :

1. Pour tout tuple $(i_0, \ldots, i_{k-1}, j) \in S^{k+1}$, on a $0 \leq Pr(j \mid i_{k-1}, \ldots, i_0) \leq 1$
2. Pour tout $(i_0, \ldots, i_{k-1}) \in S^k, \sum_{j \in S} Pr(j \mid i_{k-1}, \ldots, i_0) = 1$

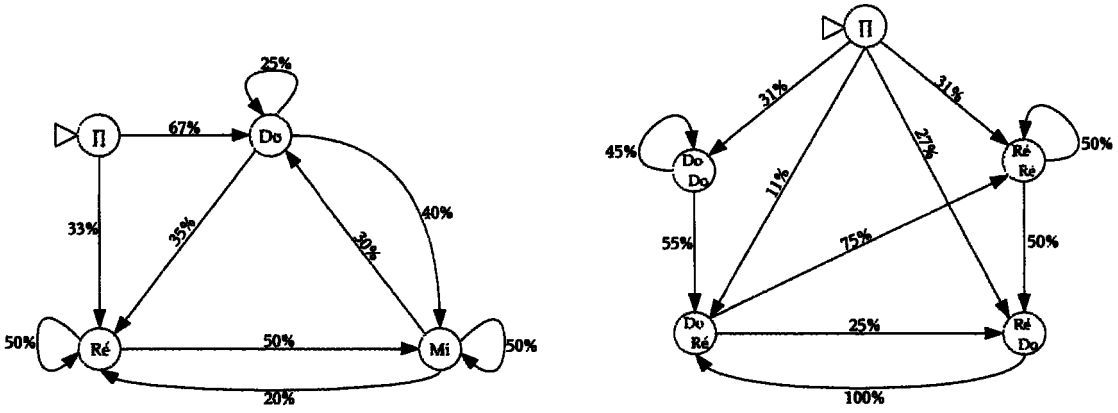
Enfin, on remplace la loi initiale par un vecteur de probabilités spécifiant la probabilité initiale de se trouver dans chaque tuple d'états à k dimensions. Formellement, ceci nous donne :

$$Pr(X_0 = i_0, \ldots, X_{k-1} = i_{k-1}) = \Pi_{(i_0, \ldots, i_{k-1})} \tag{2.5}$$

La figure 2.3 illustre une chaîne de Markov d'ordre deux. Toutefois, avant de s'y attarder, constatons que lorsque $k = 1$, les équations 2.4 et 2.5 sont équivalentes aux équations 2.1 et 2.3. Il en va de même pour les propriétés des matrices de transition d'ordre k versus les propriétés des matrices de transition de premier ordre. Ceci démontre que les chaînes d'ordre k généralisent bel et bien les chaînes de Markov de premier ordre.

2.1.2.3 Représentation sous forme de graphes

Il peut être utile de représenter une chaîne de Markov par un graphe orienté. Pour ce faire, il suffit de créer un sommet pour chaque ligne de la matrice de transition (donc pour chaque tuple $(i_1, \ldots, i_k) \in S^k$) puis, pour chaque tuple $(i_1, \ldots, j) \in S^{k+1}$, d'ajouter un arc étiqueté avec la probabilité de transition $Pr(X_{n+k} = j \mid X_{n+(k-1)} = i_{n+(k-1)}, X_{n+(k-2)} =$



(a) Chaîne de Markov d'ordre un

(b) Chaîne de Markov d'ordre deux

FIG. 2.4 – Exemple de chaînes de Markov exprimées sous forme de graphes

$i_{n+(k-2)}, \ldots, X_n = i_n$) entre les sommets correspondants aux tuples $(i_1, \ldots, i_k)$ et $(i_2, \ldots, j)$. Bien entendu, il n'est pas nécessaire de représenter les arcs étiquetés avec une probabilité de transition nulle. Quant à la loi initiale, on peut soit la laisser sous forme de vecteur aux côtés du graphe, soit la représenter sous forme d'un sommet supplémentaire Π dans le graphe. De façon générale, nous croyons qu'il est préférable de l'intégrer au graphe. Les figures 2.4(a) et 2.4(b) reprennent les exemples des figures 2.2 et 2.3 sous forme de graphes.

2.1.2.4 Applications en musique

Un artifice fréquemment utilisé en composition algorithmique est la génération de nombres aléatoires. Toutefois, les distributions aléatoires équiprobables ne sont que très rarement désirables en musique. Il est plus souvent utile d'utiliser des distributions de probabilité biaisées afin d'obtenir un *hasard contrôlé*. De même, les distributions aléatoires où tous les événements sont indépendants ne sont pas toujours désirables en musique. Il plus souvent désirable de *conditionner* les événements futurs en fonction d'un ou plusieurs événements passés, ceci afin d'imposer une structure à la musique générée. Ceci nous amène tout naturellement à considérer les chaînes de Markov qui répondent parfaitement à ces deux objectifs.

Ceci dit, il existe plusieurs façons d'exploiter les chaînes de Markov en musique. On peut, par exemple, définir l'espace d'états S comme un ensemble de notes. On y associera

ensuite une matrice de transition et un vecteur de probabilités initiales de premier ordre. Dans le cas le plus simple, chaque note aura une durée uniforme [62]. Ceci correspond à l'exemple de la figure 2.2. Il est aussi possible d'utiliser un tableau de correspondance ou une deuxième chaîne de Markov pour associer une durée différente à chaque note [62]. Enfin, on peut, sans changer l'espace d'états, utiliser une chaîne de Markov d'ordre supérieur à un, ce qui aura pour effet de faire converger la distribution de probabilité vers une forme fixe (souvent une pièce originale ayant été analysée).

Bien sûr, nous ne sommes pas obligés de nous limiter à cette définition de l'espace d'états ou même à l'utilisation d'une seule chaîne de Markov. Par exemple, dans [1], le compositeur *Charles Ames* décrit un modèle où quatre chaînes de Markov sont utilisées en parallèle, chacune contrôlant un paramètre différent de la composition. Entre autres, il a défini les paramètres suivant : la durée des notes, la probabilité de remplacer une note par un silence, le registre des notes (sept régions d'une octave) et le degré chromatique des notes (déplacement dans la gamme chromatique). Ce modèle lui permet d'obtenir des compositions beaucoup plus complexes qu'il est possible d'obtenir en utilisant une seule chaîne de Markov dont l'espace d'états correspond directement aux notes de la composition.

Dans le même article, le compositeur décrit les chaînes de chaînes, c'est-à-dire les chaînes de Markov dont l'espace d'états contient des sous-chaînes de Markov qui, optionnellement, peuvent être d'un ordre différent que celui de la chaîne parente. Il va sans dire que, dans ce modèle, l'espace d'états des sous-chaînes doit contenir un état *FIN* pour permettre au processus de retourner à la chaîne parente. De plus, il est impératif que cet état *FIN* soit atteignable à partir de tous les autres états. On remarque que ceci crée une forme de structure hiérarchique et pourrait permettre de modéliser les différents niveaux de structures d'une pièce musicale.

Quant aux manières de construire la matrice de transition, il en existe aussi plusieurs. L'une des plus populaires est de générer la matrice de transition automatiquement en analysant un ensemble de mélodies de départ. Nous avons nous-mêmes construit un tel analyseur qui prend en entrée une pièce au format MIDI, la divise en paires d'accords puis construit une matrice de transition d'ordre k en analysant les relations entre ces paires d'accords. Toutefois,

ce n'est pas la seule méthode. Il est tout autant envisageable de laisser un compositeur définir lui-même les probabilités de transition entre les états. On peut aussi songer à combiner les deux méthodes.

2.1.3 Grammaires (L-systèmes)

Introduits en 1968 par le biologiste *Aristid Lindenmayer* afin de modéliser le développement des plantes, les L-systèmes sont tout d'abord des systèmes de récriture [51]. En d'autres mots, les L-systèmes sont un type de grammaire. Ils ont d'ailleurs beaucoup en commun avec les grammaires formelles de *Chomsky* [9]. Toutefois, la différence majeure qui les distinguent des grammaires de *Chomsky* repose sur la méthode de dérivation utilisée pour arriver de la chaîne de caractères initiale à la chaîne de caractères finale. Dans les grammaires de *Chomsky*, on dérive chaque symbole récursivement jusqu'à ce qu'il n'y ait plus aucune dérivation possible. Par contraste, dans les L-systèmes, on dérive tous les symboles en parallèle pendant un certain nombre d'itérations jusqu'à l'obtention de la chaîne de caractères désirée. Les figures 2.5 et 2.6 illustrent cette différence en comparant un L-système et une grammaire de *Chomsky* reconnaissant tous deux le langage $a^n b^n c^n : n \geq 1$ et dérivant tous deux le mot *aaabbbccc*.

Définition 2.2 (*Dérivation*)
En théorie des langages, on appelle *dérivation* l'action d'appliquer une ou plusieurs productions $\alpha \rightarrow \chi$ sur un mot w afin de le transformer en un mot w' où la partie gauche de la production (α) est remplacée par la partie droite de la production (χ). Le processus de *dérivation* peut varier selon les types de grammaires.

$$\begin{aligned} \omega &\rightarrow abc \\ a > b &\rightarrow aa \\ b > c &\rightarrow bbc \end{aligned}$$

abc
aabbcc
aaabbbccc

FIG. 2.5 – L-système sensible au contexte. Les symboles à droite des > représentent le contexte à droite de la production. Les symboles du contexte de la production ne sont jamais remplacés.

$$
\begin{aligned}
S &\rightarrow abc|aSBc \\
cB &\rightarrow Bc \\
bB &\rightarrow bb
\end{aligned}
$$

$$
S \rightarrow aSBc \rightarrow aaSBcBc \rightarrow aaabcBcBc \rightarrow aaabBccBc \rightarrow aaabBcBcc \rightarrow aaabBBccc \rightarrow aaabbBccc \rightarrow aaabbbccc
$$

FIG. 2.6 – Grammaire de Chomsky sensible au contexte. Les productions peuvent être constituées de plusieurs symboles. Lorsqu'une production est appliquée, tous les symboles de gauche sont remplacés par ceux de droite.

L'intérêt des L-systèmes ne consiste toutefois pas en leur capacité à décrire des langages puisque, dû au fait qu'elles peuvent décrire beaucoup plus de langages, les grammaires de *Chomsky* s'acquittent beaucoup mieux de cette tâche. L'intérêt principal des L-systèmes consiste plutôt en leur capacité à modéliser des phénomènes aussi variés que le développement des plantes [51], la musique [40,50] ou encore la modélisation de géométrie complexe [28]. Afin d'arriver à ces résultats, plusieurs extensions aux L-systèmes de base ont été développées. Nous n'introduirons ici que la théorie essentielle sur les L-systèmes et nous inciterons le lecteur intéressé à approfondir le sujet à consulter [51].

2.1.3.1 Les L-systèmes sans contexte

Les L-systèmes sans contexte (ou 0L-systèmes) constituent la classe la plus limitée de L-systèmes. Ils ne couvrent qu'une infime partie des langages possibles. Formellement, ils sont définis comme suit :

Soit Σ un alphabet fini, $\omega \in \Sigma^+$, et P un sous-ensemble de $\Sigma \times \Sigma^*$ tel que pour chaque $\alpha \in \Sigma$, il existe exactement un $(\alpha, \chi) \in P$. Le triplet $L = \langle \Sigma, \omega, P \rangle$ représente alors un 0L-système ayant ω pour axiome et P pour ensemble de productions. Pour chaque production $(\alpha, \chi) \in P$, on écrit $\alpha \rightarrow \chi$.

Le procédé de dérivation s'effectue par itérations. À chaque itération du procédé de dérivation, on remplacera chaque symbole α par le conséquent χ de la production qui lui est associée. Ceci nous mène à la définition de dérivation directe :

Si $\phi = \alpha_1\alpha_2 \ldots \alpha_m$ et que $\psi = \chi_1\chi_2 \ldots \chi_m$ et qu'il existe des productions $\alpha_k \rightarrow \chi_k$ pour $k = 1 \ldots m$, on dit que ψ est une dérivation directe de ϕ et on écrit $\phi \Rightarrow \psi$. Si $\phi = \psi$ ou s'il existe un séquence de dérivation directes $\Delta_1 \Rightarrow \Delta_2 \Rightarrow \ldots \Rightarrow \Delta_m$ tel que $\phi = \Delta_1$ et $\psi = \Delta_m$, on dit que ψ est une dérivation de ϕ et on écrit $\phi \overset{*}{\Rightarrow} \psi$. Si, de surcroît, $\phi \neq \psi$, on écrit $\phi \overset{+}{\Rightarrow} \psi$.

On peut utiliser les L-systèmes sans contexte pour modéliser certaines structures fractales simples telles la courbe de Koch ou certains modèles de plantes peu complexes. L'approche présentée par *Prusinkiewicz* pour la musique est basée sur les L-systèmes sans contexte [50].

2.1.3.2 Les L-systèmes avec contexte

L'appellation « L-systèmes avec contexte » peut référer à plusieurs concepts différents. Nous présentons ici le concept le plus général, les IL-systèmes qu'on appelle aussi (k,l)-systèmes en référence aux contextes gauche et droit qui sont, respectivement, de longueur k et de longueur l. Notons que, à l'instar de *Prunsinkiewicz* et contrairement à la définition classique des (k,l)-systèmes, nous permettons à des productions portant des contextes de longueur différentes de co-exister dans le même IL-système. Pour ce faire, il faut supposer que s'il existe plusieurs productions portant le même prédécesseur, alors la production ayant le plus long contexte a priorité sur les autres. La figure 2.5 est un exemple de dérivation à partir d'un tel IL-système. Pour les définir, on peut modifier la définition des 0L-systèmes comme suit :

Soit Σ un alphabet fini, $\omega \in \Sigma^+$, et P un sous-ensemble de $\Sigma^*\Sigma\Sigma^* \times \Sigma^*$ tel que pour chaque $\alpha \in \Sigma$, il existe au moins un $(\phi, \alpha, \psi, \chi) \in P$. Le triplet $L = \langle \Sigma, \omega, P \rangle$ représente alors un IL-système ayant ω pour axiome et P pour ensemble de productions. Pour chaque production $(\phi, \alpha, \psi, \chi) \in P$, on écrit $\phi < \alpha > \psi \rightarrow \chi$ où ϕ représente le contexte gauche de longueur k et ψ représente le contexte droit de longueur l.

Le procédé de dérivation s'en trouve ainsi modifié puisqu'il faut maintenant vérifier si α est bel et bien entouré par ϕ et ψ avant de le remplacer par χ. On note, que contrairement à ce qui se passe avec les grammaires de *Chomsky*, ϕ et ψ restent inchangés ici.

2.1.3.3 Les L-systèmes stochastiques

Les L-systèmes que nous avons considérés jusqu'à maintenant sont tous parfaitement déterministes. Il s'agit d'une limitation gênante pour la modélisation de musique car, dans ce cas, le seul moyen d'obtenir un résultat différent avec un même L-système est de changer l'axiome de départ. Ceci fait en sorte qu'on obtient une structure musicale complètement différente après n dérivations. Ceci est problématique car il est souvent désirable de conserver la même structure musicale mais avec des variations mineures. C'est pourquoi nous introduisons ici les 0L-systèmes stochastiques[5] qui permettent d'avoir plusieurs productions ayant le même antécédent mais avec une probabilité associée.

Soit Σ un alphabet fini, $\omega \in \Sigma^+$, P un sous-ensemble de $\Sigma \times \Sigma^*$ tel que pour chaque $\alpha \in \Sigma$, il existe une ou plusieurs productions $(\alpha, \chi) \in P$, et Π: $P \rightarrow (0, 1]$ une fonction associant l'ensemble de productions à un ensemble de probabilités. Le quadruplet $L = \langle \Sigma, \omega, P, \Pi \rangle$ représente alors un 0L-système stochastique ayant ω pour axiome, P pour ensemble de productions et Π pour distribution de probabilités. On suppose que pour chaque symbole $\alpha \in \Sigma$ la somme des probabilités Π sur les productions ayant α comme antécédent est égale à 1.

Ceci modifie le procédé de dérivation en ce sens que le successeur d'une production dépend maintenant de la distribution de probabilités Π. La figure 2.7 donne un exemple de 0L-système stochastique.

$$\omega \longrightarrow XXYY$$
$$X \xrightarrow{0.5} XXYY$$
$$X \xrightarrow{0.25} CEG$$
$$X \xrightarrow{0.25} GBD$$
$$Y \xrightarrow{0.5} CDC$$
$$Y \xrightarrow{0.5} GAG$$

FIG. 2.7 – Un exemple de 0L-système stochastique

[5]Notons qu'il existe aussi des IL-systèmes stochastiques.

2.1.3.4 Applications en musique

L'utilisation d'un L-système pour la composition musicale implique généralement la recherche d'un résultat possédant une certaine structure puisque ceux-ci sont aussi utilisés pour la modélisation de plantes, de fractales et d'autres formes hautement structurées [50, 69]. Ils sont donc parfaitement adaptés à la composition note à note. Ainsi, ils sont utilisés par plusieurs auteurs pour générer des formes musicales une note à la fois. De plus, il existe souvent une représentation graphique de la forme musicale en question. Par exemple, *Prusinkiewicz* propose une méthode établissant une correspondance entre une courbe générée par un 0L-système dans un plan cartésien et une forme musicale [50]. Dans sa méthode, la fréquence des notes correspond aux coordonnées en y de chaque segment de la courbe et la durée des notes aux coordonnées en x de chaque segment de la courbe. Les figures 2.8, 2.9 et 2.10, extraites de [50] illustrent le résultat de l'application de la méthode de *Prusinkiewicz* à la courbe d'Hilbert [29, 42]. *Worth et Stepney*, quant à eux, ont cherché à établir une correspondance entre des images de plantes générées à l'aide de L-systèmes et des formes musicales [69].

2.1.4 Algorithmes génétiques

Introduits par *John Holland* en 1975, les algorithmes génétiques sont des algorithmes d'optimisation stochastiques basés sur la survie du plus fort. Ceux-ci fonctionnent en générant une population initiale de solutions candidates. Puis, à chaque itération, ils sélectionnent les individus les plus aptes à se reproduire afin de produire la génération suivante de solutions candidates. Occasionnellement, une nouvelle solution est introduite de façon aléatoire afin de diversifier la population [20].

Pour fonctionner, les algorithmes génétiques requièrent une représentation génétique du problème à optimiser et une fonction de mérite[6] (« fitness function ») pour évaluer les solutions. Ces deux préalables doivent être définis avant le démarrage de l'algorithme et sont toujours dépendant du domaine d'optimisation. On appelle chromosome le résultat de l'encodage du problème sous une forme appropriée pour un algorithme génétique. Dans

[6] Aussi appelée fonction objectif

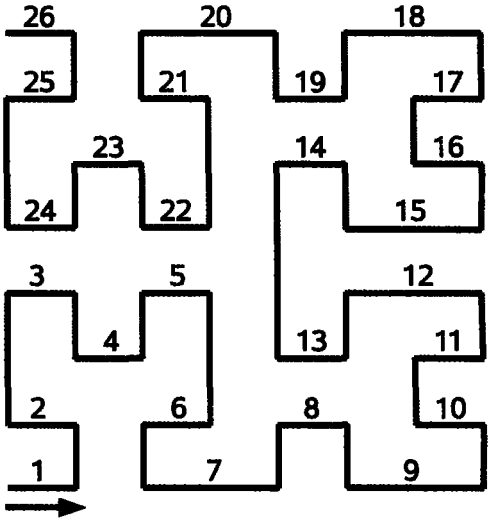


FIG. 2.8 – Traversée de la courbe d'Hilbert

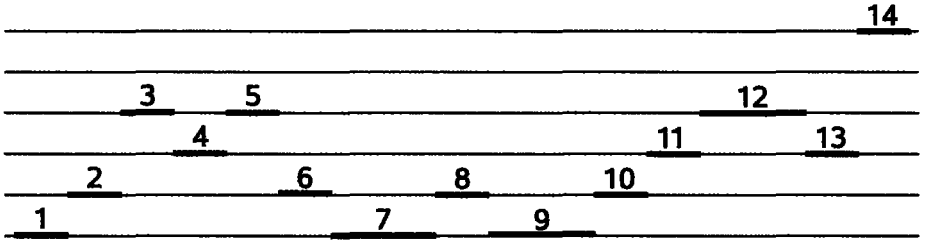


FIG. 2.9 – La partition associée en notation « piano roll »

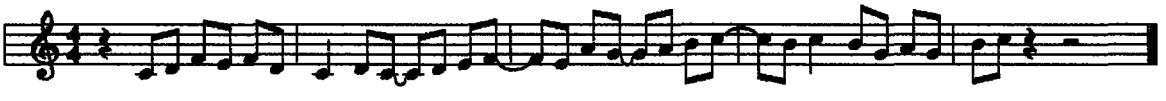


FIG. 2.10 – La partition associée en notation standard

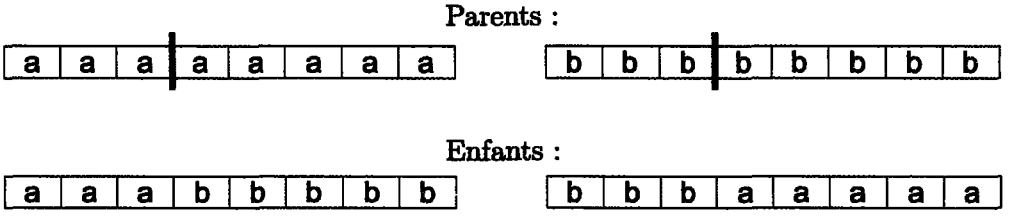


FIG. 2.11 – Processus de recombinaison

l'algorithme de base, tel que défini par *Holland*, cet encodage prend la forme d'une séquence de bits que l'algorithme pourra ensuite modifier individuellement [19]. On appelle ces bits des gènes.

La production d'une nouvelle génération de solution candidates commence par la sélection des individus les plus aptes à se reproduire. Pour ce faire, on évalue d'abord le mérite de la génération en cours avec la fonction de mérite puis on sélectionne, de façon aléatoire, un certain nombre de couples d'individus pour servir de parents à la génération suivante. Les individus ayant le plus haut mérite se verront attribués une meilleure chance d'être sélectionnés [67].

Les parents de la prochaine génération étant sélectionnés, on applique maintenant l'opérateur de recombinaison. Celui-ci est défini par la sélection d'un gène au hasard sur le chromosome définissant les individus et par l'échange, entre les deux parents, de ce gène et de tous les gènes situés après celui-ci sur le chromosome. Pour éviter le cas où les deux chromosomes seraient échangés sans recombinaison, on peut définir une position minimale sur le chromosome à partir de laquelle se fait la sélection de la position d'échange. La figure 2.11 illustre ce procédé.

Enfin, on applique l'opérateur de mutation aux enfants résultant de l'opérateur de recombinaison. Celui-ci est défini par le remplacement aléatoire, avec une probabilité faible (de l'ordre de 0.001), de chaque gène par un bit aléatoire. On peut aussi définir l'opérateur de mutation comme une perturbation où l'on remplace la valeur des gènes en question par leur inverse binaire. Peu importe la définition choisie, la probabilité de mutation doit rester faible sinon l'algorithme génétique s'en trouvera réduit à une recherche complètement aléatoire [61].

Il existe plusieurs variations sur la définition canonique des algorithmes génétiques. Par exemple, on peut utiliser un encodage constitué de nombres entiers ou réels plutôt que d'une chaîne de bits pour les chromosomes. On peut aussi définir des opérateurs de sélection, de recombinaison et de mutation plus sophistiqués. Par exemple, on peut implémenter la recombinaison en plusieurs points puisque parfois c'est l'interaction entre les gènes du début et de la fin d'un chromosome qui font que celui-ci est aussi bien adapté à survivre. Si l'encodage utilisé est constitué de nombres réels, on peut implémenter la recombinaison en effectuant la moyenne arithmétique des deux parents ou encore on peut implémenter des opérateurs spécifiques au domaine du problème à optimiser. Les possibilités sont multiples.

2.1.4.1 Applications en musique

Il existe plusieurs façons d'appliquer les algorithmes génétiques dans le monde de la musique. Ces méthodes sont souvent très différentes et n'ont en commun que le fait qu'elles sont basées sur un algorithme génétique. Par exemple, la méthode de *Biles* [4] utilise un algorithme génétique adapté pour le temps réel et dont la fonction de mérite correspond à une évaluation faite en temps réel par l'utilisateur. Pour ce faire, leur système, *GenJam*, incrémente le mérite d'une mesure ou bien d'une phrase (le système utilise deux populations : une de mesures et une de phrases) lorsque l'utilisateur tape « g » (pour good) au clavier et décrémente le mérite lorsque l'utilisateur tape « b » (pour bad) au clavier. Nécessairement, le système offre à l'utilisateur un temps de réaction en faisant en sorte que son entrée affecte la dernière mesure (ou phrase) jouée et non celle en train d'être jouée. Une autre approche possible est celle utilisée par *Spector et Alpern* [59] qui utilisent la programmation génétique [35] pour faire évoluer des programmes pouvant produire une réponse à un appel (terminologie tirée du jazz). Les deux consistent en une mesure en temps $\frac{4}{4}$.

2.1.5 Automates cellulaires

Introduits par *Stanislaw Ulam et John von Neumann* dans le but d'étudier les systèmes auto-réplicants, les automates cellulaires sont des structures mathématiques qui peuvent exhiber des comportements très complexes [64,68]. Entre autres, on a remarqué chez certains automates

des comportements cycliques, d'auto-organisation, ou encore de propagation de motifs [43]. D'autres encore ont démontré une capacité à simuler des machines de Turing [3]. On appelle ces comportements propriétés émergentes du système car ils ne font pas explicitement partie de la définition du modèle et ne peuvent pas être expliqués par les seules composantes de celui-ci. Ces propriétés des automates cellulaires font en sorte que ceux-ci présentent un intérêt certain pour la composition de musique.

2.1.5.1 Définition

Comme les chaînes de Markov, les automates cellulaires sont des systèmes dynamiques à temps discret, c'est-à-dire qu'ils sont définis par une fonction de la forme $x_{t+1} = f(x_t)$. On les définit à partir d'une grille infinie de dimension arbitraire (réseau de dimension n) divisée en cellules pouvant chacune prendre un état choisi parmi un ensemble fini. L'état d'une cellule x au temps $t+1$ ne dépend que de l'état des cellules de son voisinage au temps t. Ce voisinage est constitué d'un nombre fini de cellules du réseau spécifiées de façon relative à la cellule x et peut inclure celle-ci. Lorsque chaque cellule possède la même règle de mise à jour et que cette règle ne dépend pas du temps, on dit que l'automate est homogène dans le temps et l'espace. Dans les lignes qui vont suivre, nous ne considérerons que ce type d'automates cellulaires qui sont formellement définis comme suit :

Définition 2.3 (*Réseau*)
En géométrie et en théorie des groupes, on définit un *réseau* de $\mathbb{R}^n$ comme un sous-groupe additif discret de $\mathbb{R}^n$ qui engendre $\mathbb{R}^n$. Tout *réseau* de $\mathbb{R}^n$ peut être engendré en formant les combinaisons linéaires à coefficients entiers d'une base de $\mathbb{R}^n$. On peut visualiser un *réseau* comme une grille régulière dans l'espace $\mathbb{R}^n$.

Soit n la dimension de l'automate, R un réseau infini sur $\mathbb{R}^n$, m la taille du voisinage d'une cellule, S un ensemble fini non-vide d'états, V un ensemble de taille m de coordonnées dans $\mathbb{Z}^n$ tel que $\forall i \in V; \forall x \in R : (x+i) \in R$, et $\delta : S^m \rightarrow S$. Le quadruplet $A = \langle R, S, V, \delta \rangle$ représente alors un automate cellulaire où V représente l'ensemble des coordonnées relatives des cellules voisines et δ la fonction de transition locale d'une cellule x.

On définit une configuration C par une application de $\mathbb{Z}^n$ dans S, soit $C\colon \mathbb{Z}^n \to S$. Une configuration au temps t évolue vers une autre configuration au temps $t+1$ selon la relation définie par : $\forall x \in R : C_{t+1}(x) = \delta(C_t(x+V_1), \ldots, C_t(x+V_m))$. On appelle cette relation fonction de transition globale de l'automate. On dénote la configuration initiale de l'automate par C_0.

Pour illustrer cette évolution des configurations d'un automate cellulaire, prenons un automate cellulaire à deux dimensions dont le voisinage est défini par l'ensemble : $V = \{\{-1,-1\},\{0,-1\},\{1,-1\},\{-1,0\},\{1,0\},\{-1,1\},\{0,1\},\{1,1\}\}$. On peut représenter ce voisinage par la figure 2.12.

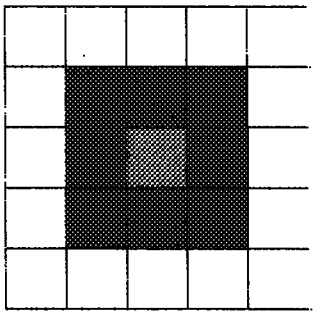


FIG. 2.12 – Exemple de voisinage

Posons aussi la fonction de transition suivante pour l'automate :

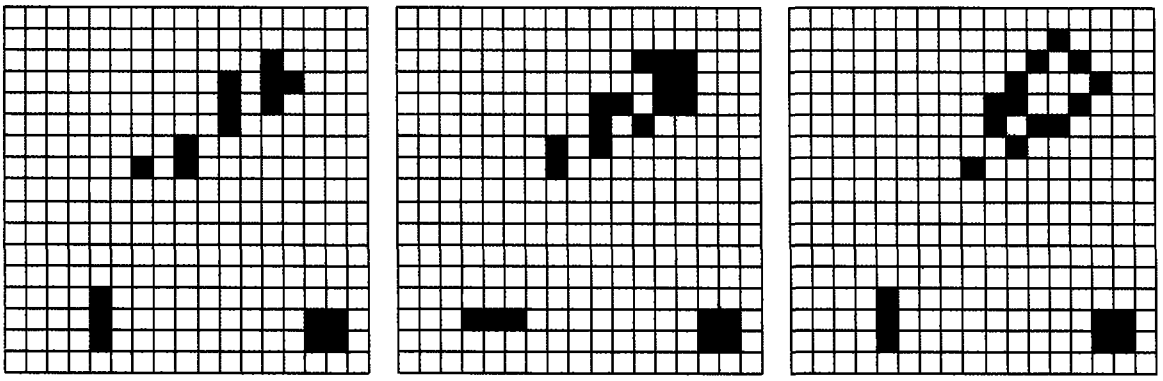
$$\forall x \in R : C_{t+1}(x) = \begin{cases} C_t(x) & si \sum_{i \in V} C_t(x+i) = 2, \\ 1 & si \sum_{i \in V} C_t(x+i) = 3, \\ 0 & dans\ tous\ les\ autres\ cas. \end{cases} \tag{2.6}$$

Ceci correspond aux règles du « jeu de la vie » de *Conway* [11]. Maintenant, supposons que la configuration initiale de l'automate est telle que représentée à la figure 2.13(a). Dans ce cas, les deux configurations suivantes seront telles que représentées aux figures 2.13(b) et 2.13(c).

2.1.5.2 Automates cellulaires sur une grille finie

La définition précédente implique l'utilisation d'un réseau infini. Toutefois, en pratique, on utilise généralement une grille finie car cela facilite la visualisation de l'automate. De plus,

il est nécessaire d'utiliser une grille finie si l'on veut simuler des automates cellulaires sur ordinateur. Il nous faut donc trouver un moyen de créer des automates cellulaires de tailles finies.



(a) Configuration initiale (temps 0) (b) Configuration au temps 1 (c) Configuration au temps 2

FIG. 2.13 – Exemple d'évolution des configurations d'un automate cellulaire

Une première intuition pour créer un automate cellulaire de taille finie consiste à n'appliquer la fonction de transition qu'aux cellules appartenant à la grille. Malheureusement, cela est insuffisant car, même si elle est locale à chaque cellule, la fonction de transition peut, en cherchant à évaluer les cellules voisines, faire référence à des cellules en dehors de la grille (*cellules-bordures*). C'est pourquoi il nous faut aussi trouver une fonction de correspondance entre un système de coordonnées infini et un système de coordonnées fini. À cette fin, il existe plusieurs possibilités : [21]

1. Traiter la grille comme un tore de dimension n (évaluer les coordonnées modulo *taille de la grille*).
2. Utiliser une copie miroir de la grille pour les cellules-bordures. Il s'agit d'une variante de la méthode 1.
3. Fixer l'état des cellules-bordures à une valeur constante.

Les méthodes les plus simples et les plus utilisées sont les méthodes 1 et 3. La méthode 2 est plus complexe car il faut déterminer l'orientation de la grille miroir dans laquelle tombe la cellule-bordure puisqu'il faut ensuite projeter pour obtenir la cellule correspondante de la grille principale. La figure 2.14 illustre ce problème en deux dimensions.

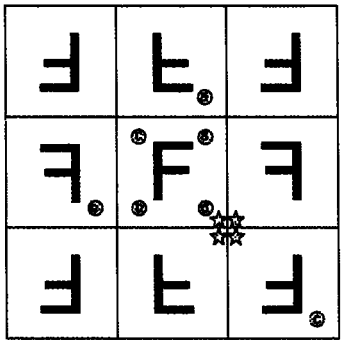


FIG. 2.14 – Grilles miroirs et projection de cellules

Toutefois, la projection des cellules n'est pas le problème le plus important de la méthode 2. En effet, on constate qu'avec cette méthode, trop de cellules deviennent leurs propre voisines, comme les cellules représentées par des petites étoiles sur la figure 2.14. C'est pourquoi elle est peu utilisée.

Définition 2.4 (*Comportement émergent*)
Un *comportement émergent* est un comportement complexe imprévu lors de la conception d'un système mais néanmoins effectué par celui-ci et qui découle de plusieurs comportements plus simples introduits dans le système lors de sa conception.

2.1.5.3 Applications en musique

Dû à leurs comportements émergents, les automates cellulaires représentent un important outil de génération de motifs et de séquences pour les artistes [5]. En particulier, les automates cellulaires présentant les propriétés de comportement cyclique, d'auto-organisation et de propagation de motifs ont été étudiés par *McAlpine et al.* [41] et *Miranda* [43] dans le contexte de la composition musicale. Ceux-ci ont implémenté un système de composition basé sur le « jeu de la vie » de *Conway* [11] et l'automate cellulaire cyclique de *Griffeath* [24]. De façon spécifique, à chaque étape, leur système, CAMUS 3D, utilise le « jeu de la vie » pour sélectionner les enchaînements d'accords à jouer et l'automate cyclique pour sélectionner les instruments avec lesquels seront joués ces accords. Pour ce faire, on parcourt la grille du « jeu de la vie » de gauche à droite, de haut en bas, de l'avant vers l'arrière. Lorsqu'une cellule

vivante est rencontrée, on ajoute à la partition un accord constitué d'un quadruplet de notes sélectionnées de la façon suivante :

1. On sélectionne une note fondamentale de façon aléatoire,
2. On examine les coordonnées cartésiennes x, y et z de la cellule,
3. On définit la deuxième note du quadruplet comme la note x demi-tons au dessus de la fondamentale,
4. On définit la troisième note du quadruplet comme la note y demi-tons au dessus de la deuxième note,
5. On définit la quatrième note du quadruplet comme la note z demi-tons au dessus de la troisième note,
6. Enfin, on sélectionne une façon de jouer cet accord (ordre de déclenchement et durée des notes) de façon aléatoire.

2.1.6 Automates finis

On remarquera, en examinant les figures 2.4(a) et 2.4(b), qu'une chaîne de Markov représentée par un graphe ressemble beaucoup à un automate probabiliste. En fait, nous montrerons à la section 2.1.6.3 que les chaînes de Markov peuvent très bien être représentées par un automate probabiliste. Mais avant tout, nous devons définir les différents types d'automates finis. Nos définitions ont été inspirées par *Sipser* [57], *Yvon et Demaille* [70], *Turakainen* [66], et *Cheng* [7].

Notons que les automates finis peuvent avoir plusieurs fonctions : reconnaissance de langages, production de langages, contrôle de systèmes numériques. Nos premières définitions concerneront les automates reconnaissant des langages, puis, à la section 2.1.6.4, nous traiterons des automates servant à la production de langages : les transducteurs.

2.1.6.1 Automates finis déterministes

Un automate fini déterministe est un automate où, pour chacun de ses états, il n'existe qu'une et une seule transition par symbole de l'alphabet d'entrée. Ceci signifie

que, pour chaque combinaison d'état et de symbole de l'alphabet d'entrée, il n'existe pas d'ambiguïté quant au prochain état que visitera l'automate. Formellement, ils sont définis comme suit :

Soit S un ensemble fini d'états, $S_0 \in S$, F un sous-ensemble de S, Σ un alphabet fini, et δ une fonction totale de $S \times \Sigma$ dans S. Le quintuplet $\langle S, S_0, F, \Sigma, \delta \rangle$ représente alors un automate fini déterministe ayant S_0 pour état initial, F pour états finaux, et δ comme fonction de transition. Cet automate peut alors être représenté par un graphe orienté dans lequel les nœuds correspondent aux états de l'automate et les arcs aux transitions de celui-ci. Les arcs sont étiquetés selon la fonction de transition tel que a est l'étiquette de l'arc (q, r) si et seulement si $\delta(q, a) = r; q, r \in S, a \in \Sigma$. L'état initial est représenté par un arc entrant sans origine et les états finaux par des arcs sortants sans destination. Alternativement, les états finaux peuvent être représentés par un double trait, comme sur la figure 2.15.

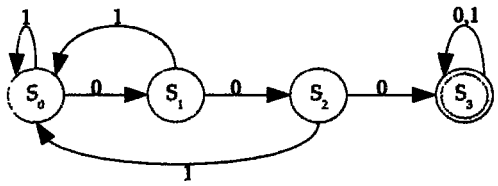


FIG. 2.15 – Automate fini déterministe

Ainsi donc, on définit un parcours dans l'automate par une séquence d'états $r_1\, r_2 \ldots r_n$. On dira alors qu'un automate reconnaît un mot $w \in \Sigma^*$ si et seulement s'il existe un parcours tel que $r_1 = S_0$, $r_n \in F$, et $\delta(r_i, w_i) = r_{i+1}$ où w_i représente le i_e caractère du mot w. On dira qu'un automate reconnaît un langage L si celui-ci est l'union de tous les mots reconnus par l'automate. Formellement, on peut définir le langage reconnu par un automate A par l'équation suivante :

$$L(A) = \{w \in \Sigma^* \mid \delta^*(S_0, w) \in F\} \tag{2.7}$$

où $\delta^*(q, w)$ représente une version récursive de la fonction de transition définie comme :

$$\delta^*(q, w) = \begin{cases} q & si\ w = \varepsilon, \\ \delta^*(\delta(q, a), u) & si\ w = au. \end{cases} \tag{2.8}$$

Par cette définition, on constate que la complexité de l'algorithme de reconnaissance d'un mot par un automate fini déterministe est linéaire, c'est-à-dire que pour reconnaître un mot de longueur $|w|$, l'automate prendra $|w|$ étapes. Ce sont donc des algorithmes très efficaces et c'est pourquoi ils sont très utilisés.

Définition 2.5 (*Fonction partielle*)
En mathématiques, une *fonction partielle* est une fonction $f\colon X \to Y$ qui associe chaque élément de l'ensemble X (appelé domaine de la fonction) à *au plus* un élément de l'ensemble Y (appelé co-domaine de la fonction). Il n'est pas nécessaire que chaque élément du domaine ait un successeur dans le co-domaine.

2.1.6.2 Automates finis non-déterministes

Les automates finis non-déterministes sont une généralisation des automates déterministes. Ils sont plus souples que ces derniers en ce sens qu'ils permettent à plusieurs transitions sortantes d'un état q de porter le même symbole. Ils permettent aussi l'utilisation de transitions spontanées (aussi appelées transitions epsilon) [57, 70].

Formellement, ils sont définis de manière similaire aux automates déterministes, c'est-à-dire comme un quintuplet $\langle S, S_0, F, \Sigma, \delta \rangle$ où S représente l'espace d'états de l'automate, S_0 l'état initial, $F \subseteq S$ les états finaux, et Σ l'alphabet de l'automate. Ils diffèrent toutefois des premiers par la fonction de transition δ qui est maintenant définie comme une fonction partielle de $S \times \{\Sigma \cup \varepsilon\}$ dans 2^S.

Pour définir $\delta^*(q, w)$ pour les automates non-déterministes, on pose d'abord $T \subseteq S$. On définit alors $E(T) = \bigcup_{i \geq 0} E_i(T)$ où :

$$E_i(T) = \begin{cases} T & si\ i = 0, \\ \bigcup_{q \in E_{i-1}(T)} \delta(q, \varepsilon) & si\ i > 0. \end{cases} \tag{2.9}$$

On peut alors définir $\delta^*(q, w)$:

$$\delta^*(q,w) = \begin{cases} E(\{q\}) & si\ w = \varepsilon, \\ \bigcup_{r \in \delta^*(q,a)} E(\delta(r,a)) & si\ w = ua. \end{cases} \tag{2.10}$$

Ceci nous permet de définir le langage L reconnu par un automate non-déterministe A :

$$L(A) = \{w \in \Sigma^* \mid \delta^*(S_0, w) \cap F \neq \emptyset\} \tag{2.11}$$

Grâce à ces dernières définitions, on peut affirmer que le processus de reconnaissance est beaucoup moins efficace lorsqu'il est effectué par un automate non-déterministe que par un automate déterministe. En effet, on constate que puisque la fonction de transition retourne plusieurs états possibles pour un même symbole et qu'il faut les considérer tous, alors l'automate possède en réalité $2^{|Q|}$ états. Ceci se voit plus facilement lorsqu'on applique le processus de conversion en automate déterministe qui est bien défini pour tous les automates finis non-déterministes [57]. Les détails de ce processus seraient toutefois trop poussés pour ce mémoire et nous invitons le lecteur intéressé à consulter des ouvrages tels que ceux de *Sipser* [57] et de *Yvon et Demaille* [70].

Le fait que les automates non-déterministes soient tous convertibles en automates déterministes montre qu'ils n'apportent rien de plus au niveau de la reconnaissance de langages. Toutefois, même s'ils n'apportent rien sur ce point, ils ont toutefois un avantage majeur : ils simplifient énormément la spécification d'automates en plus de faciliter leur compréhension. Ceci est particulièrement vrai lorsqu'on utilise les transitions spontanées qui sont des transitions étiquetées avec le mot vide (ε) et qui peuvent être empruntées sans consommer de symbole du mot à reconnaître.

2.1.6.3 Automates finis probabilistes

Comme les automates non-déterministes, les automates finis probabilistes sont une généralisation des automates déterministes. Ils remplacent la fonction de transition déterministe par une fonction de transition probabiliste et l'état initial par un vecteur de probabilités spécifiant la probabilité initiale de se trouver dans chaque état. En ce sens, ils sont aussi une généralisation des chaînes de Markov. Formellement, ils sont définis comme suit :

Soit S un ensemble fini d'états, $\Pi \in \mathbb{R}^{|S|}$, F un sous-ensemble de S, Σ un alphabet fini, et $\mathcal{P}$ une fonction totale de $S \times \Sigma \times S$ dans $\mathbb{R}$. Le quintuplet $\langle S, \Pi, F, \Sigma, \mathcal{P} \rangle$ représente alors un automate fini probabiliste ayant Π pour loi initiale, F pour états finaux, et $\mathcal{P}$ comme matrice de transition.

La matrice de transition possède trois dimensions : $\mathcal{P} = Pr(j|i,a), (i,j) \in S^2, a \in \Sigma$ qui spécifie la probabilité de passer de l'état i à l'état j étant donné le symbole a. Pour être valide, elle doit respecter les deux propriétés suivante :

1. Pour tout triplet $\langle i, a, j \rangle$, on a $1 \geq Pr(j|i,a) \geq 0$
2. $\forall a \in \Sigma, \forall i \in S \sum_{j \in S} Pr(j \mid i, a) = 1$

À partir de ces définitions, on peut maintenant définir une fonction de transition pour les automates probabilistes : $\delta(\varpi, a) = \varpi \mathcal{P}_a$ où ϖ est un vecteur colonne représentant la probabilité courante de se trouver dans chaque état et $\mathcal{P}_a$ représente le plan $\mathcal{P}(*, a, *)$ de la matrice de transition. Nous pouvons maintenant définir la fonction de transition récursive :

$$\delta^*(\varpi, w) = \begin{cases} \varpi & si\ w = \varepsilon, \\ \delta^*(\delta(\varpi, a), u) & si\ w = au. \end{cases} \tag{2.12}$$

Ainsi, l'état de l'automate à l'instant $i+1$ est la multiplication du vecteur colonne ϖ_i avec le plan $\mathcal{P}(*, a, *)$ de la matrice de transition pour donner le vecteur colonne ϖ_{i+1}.

Enfin, pour définir le processus de reconnaissance avec les automates probabilistes, on modifie la notion d'acceptation car, contrairement à tous les autres types d'automates, il n'est pas possible d'affirmer avec certitude si l'automate se trouvera dans un état final. Un mot d'un langage L sera donc accepté si la probabilité de se trouver dans un état final est supérieure à un point de coupure $0 \leq \nu < 1$. Le point de coupure est spécifique à chaque langage. On dit alors de L qu'il s'agit d'un *langage stochastique* puisqu'il est reconnu par un automate fini probabiliste [66] et on notera :

$$L(A, \eta) = \{w \in \Sigma^* \mid \delta^*(\pi, w), F > \eta\} \tag{2.13}$$

Il est maintenant facile de voir que les chaînes de Markov sont des cas particulier des automates finis probabilistes. En effet, considérons un espace d'états S, une matrice de transition $\mathcal{P}$ et une loi initiale Π partagées par une chaîne de Markov $M = \langle S, \mathcal{P}, \Pi \rangle$ et un automate probabiliste $A = \langle S, \Pi, F, \Sigma, \mathcal{P} \rangle$. Posons aussi $F = S$, $\Sigma = \{a\}$, $\omega = a^n$ où ω est un mot de longueur n à faire reconnaître par l'automate. Dans ce cas, la séquence d'états visités par l'automate est la même que la séquence d'états $X_0, \ldots, X_n$ visités par la chaîne de Markov (si l'on suppose que la même série de nombres « pseudo-aléatoires » est utilisée pour les deux).

2.1.6.4 Transducteurs

On appelle transducteurs la classe d'automates effectuant la traduction d'un langage $L(A)$ vers un langage $L(B)$. Il en existe deux types fonctionnellement équivalents : le premier type est appellé machine *machine de Moore* et associe la sortie aux états de l'automate, le second est appelé *machine de Mealy* et associe la sortie aux transitions de l'automate. Dans ce mémoire, nous nous intéresserons à ceux du premier type qui sont formellement définis comme suit :

Soit S un ensemble fini d'états, $S_0 \in S$, F un sous-ensemble de S, Σ un alphabet d'entrée fini, Γ un alphabet de sortie fini, δ une fonction totale de $S \times \Sigma$ dans S, et G une fonction totale de S dans $\{\Gamma \cup \varepsilon\}$. Le septuplet $\langle S, S_0, F, \Sigma, \Gamma, \delta, G \rangle$ représente alors un transducteur ayant S_0 pour état initial, F pour états finaux, δ comme fonction de transition, et G comme fonction de sortie. On nomme ce type d'automates transducteurs puisqu'ils traduisent d'un langage à un autre. On utilise aussi le nom machines de Moore [44] pour les différencier d'un deuxième type de transducteurs.

La transduction du langage L_1 vers le langage L_2 s'opère en même temps que le processus de reconnaissance du langage L_1. Ces langages sont définis de la même manière qu'à la section 2.1.6.1. De même, le processus de reconnaissance fonctionne exactement de la même manière. Toutefois, la différence ici est que lorsque l'automate visite un état (y compris l'état initial S_0), il applique la fonction G et écrit le symbole spécifié par celle-ci

sur son ruban de sortie. Pour finir, si l'automate se trouve dans un état final à la fin du processus de reconnaissance, il retourne le mot sur son ruban de sortie. Sinon, il retourne le mot vide.

On remarque que la définition précédente est basée sur les automates finis déterministes. Il n'est toutefois pas difficile d'adapter celle-ci pour accommoder des transducteurs non-déterministes ou encore des transducteurs probabilistes. Pour ce faire, il suffit d'ajouter Γ et G aux modèles en question.

2.1.6.5 Automates étendus

Les automates étendus sont un type particulier de transducteurs. Ils ont été introduits par *Cheng et Krisnakumar* [7] afin de pallier au problème d'explosion combinatoire du nombre d'états dans les automates traditionnels (voir figure 2.16). Pour ce faire, ils ajoutent à ceux-ci un ensemble de registres distincts des états. Les opérations sur ces registres sont modélisées dans les transitions. Il en résulte des automates munis de mémoires tampons. Formellement, ils sont définis comme suit :

Soit S un ensemble fini d'états, $S_0 \in S$, F un sous-ensemble de S, Σ un alphabet d'entrée fini, Γ un alphabet de sortie fini, $V = V_1 \times \ldots \times V_n$, C un ensemble de fonctions C_i tel que $C_i \colon V \rightarrow \{0,1\}$, ϕ un ensemble de transformations ϕ_i tel que $\phi_i \colon V \rightarrow V$, δ est une fonction partielle de $S \times C \times \Sigma$ dans $S \times \phi$, et G est une fonction totale de S dans Γ. On dit alors que le décuplet $\langle S, S_0, F, \Sigma, \Gamma, V, C, \phi, \delta, G \rangle$ représente un automate étendu ayant S_0 pour état initial, F pour états finaux, V pour ensemble de registres, C pour fonctions d'activations, ϕ pour fonctions de mise à jour, δ comme fonction de transition, et G comme fonction de sortie. Chaque registre V_i prend une valeur appartenant à l'ensemble des entiers $\mathbb{N}$. Notons qu'il n'est pas nécessaire que la taille de V soit finie. En ce sens, il n'est pas obligatoire qu'un automate étendu particulier ait un automate fini équivalent [36].

Afin de décrire le fonctionnement des automates étendus, nous allons utiliser $\vec{x}$ pour dénoter une configuration particulière des registres d'un l'automate tel que $\vec{x}_i \in V_i$. Ainsi, si q et r sont deux états de l'automate, a est un symbole appartenant à l'alphabet d'entrée Σ,

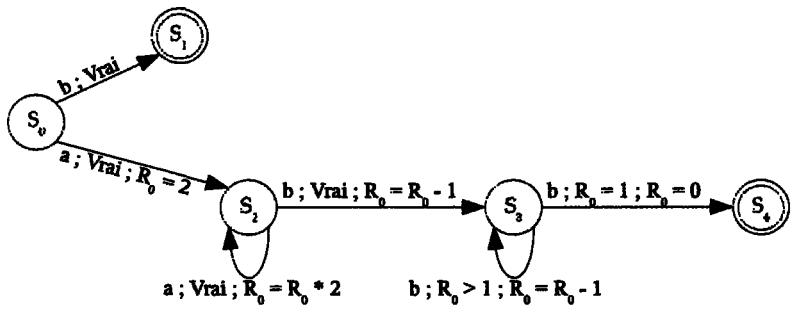


FIG. 2.16 – Automate étendu acceptant le langage $a^n b^{2^n}$ qui ne peut être représenté par un automate traditionnel

$f \in C$ est une fonction d'activation, et $u \in \phi$ est une fonction de mise à jour, alors la transition $\delta(q, f, a) = (r, u)$ s'interprète comme suit :

Si l'automate est dans la configuration $\langle q, \vec{x} \rangle$, soit dans l'état q avec le vecteur de variables $\vec{x}$ tel que $f(\vec{x}) = 1$ et qu'il lit le caractère a sur le ruban d'entrée, alors il passera dans la configuration $\langle r, \vec{y} \rangle$, tel que $\vec{y} = u(\vec{x})$. On note que s'il n'existe aucune transition sortant de l'état q tel que $f(\vec{x}) = 1$, l'automate s'arrête sans retourner le contenu du ruban de sortie.

Le papier de *Cheng et Krishnakumar* oublie toutefois de mentionner que si l'on veut avoir un automate étendu parfaitement déterministe, il faut supposer que pour toute paire $\langle q, a \rangle \in S \times \Sigma$, toutes les transitions sortant de l'état q après lecture du symbole a portent des fonctions d'activation $f \in C$ mutuellement exclusives. Dans le cas contraire, on obtient un automate étendu non-déterministe où ni l'état de l'automate ni la valeur des registres ne sont déterminés. Nous reviendrons sur le problème des fonctions d'activation non-mutuellement exclusives au chapitre 3.

2.2 Outils de composition

Dans cette section, nous parlerons de quelques outils de composition de musique interactive et/ou algorithmique[7] existants. Certains de ces outils sont très utilisés dans l'industrie du jeu vidéo, d'autres moins.

[7]Comme la ligne qui sépare la musique interactive de la musique algorithmique est très mince, nous les couvrirons en un seul bloc.

Il existe plusieurs façons d'envisager les outils pour la composition de musique algorithmique. On peut, par exemple, laisser l'utilisateur programmer lui-même ses algorithmes à l'aide d'un langage de programmation général. À l'inverse, on peut fournir à l'utilisateur différents types d'interfaces graphiques, chacune imposant une abstraction différente sur le processus de composition. Dans cette section, nous couvrirons une gamme d'outils de composition de musique algorithmique, sélectionnés non seulement pour leur appartenance à chacun de ces différents groupes d'outils mais aussi pour leur importance dans le monde académique ou dans l'industrie du jeu vidéo.

Le premier outil que nous présentons ici, *Common Music* [62], est en fait un ensemble d'extensions au langage Lisp permettant le traitement de structures musicales. En ce sens, il appartient au groupe des outils qui laissent l'utilisateur programmer lui-même ses algorithmes à l'aide d'un langage de programmation général. Il permet d'effectuer le rendu de ces structures dans divers formats de fichiers musicaux. Ainsi, un compositeur voulant utiliser *Common Music* doit tout d'abord apprendre quelques notions de programmation. Le premier tiers du livre de *Taube* [62] est d'ailleurs consacré à l'apprentissage du langage Lisp et des extensions fournies par *Common Music*. Une autre lacune de *Common Music* provient de son incapacité à répondre à des stimuli externes en temps réel lors du rendu d'algorithmes musicaux. Il ne s'agit donc pas d'un outil de composition de musique interactive. En contrepartie, le système possède de nombreux points positifs dont sa grande flexibilité, son extensibilité, et sa portabilité. En effet, comme Lisp est un langage de programmation très général, *Common Music* peut tout aussi bien être utilisé pour la composition d'œuvres complètement tonales et déterministes que pour la composition d'œuvres incorporant des éléments de microtonalité et de hasard [49]. De plus, un programmeur suffisamment compétent peut étendre le système avec de nouvelles fonctionnalités. Il serait ainsi envisageable de rendre Common Music interactif en lui permettant de répondre à des stimuli externes, l'environnement d'un jeu vidéo par exemple. Enfin, puisqu'il existe un interpréteur Lisp sur pratiquement toutes les plateformes, le système est très portable.

Le deuxième outil que nous considérons, *OpenMusic* [13], est un environnement de programmation visuelle complet pour la musique algorithmique. Il fournit à l'utilisateur

une interface graphique à travers laquelle celui-ci peut rapidement concevoir de nouveaux algorithmes musicaux sans passer par l'apprentissage du langage Lisp. Plus particulièrement, l'utilisateur peut composer des pièces en plaçant des « patches », c'est-à-dire des algorithmes construits à partir de fonctions primitives, sur une maquette qui est en fait une sorte de « super-patch » avec une dimension temporelle. La construction de « patches » se fait en sélectionnant des composantes à partir d'une bibliothèque de fonctions fournie par le logiciel, en spécifiant leurs paramètres, et en les reliant l'une à l'autre. Les « patches » peuvent aussi posséder des paramètres d'entrée/sortie et être appelées récursivement. On voit immédiatement qu'il s'agit d'un outil extrêmement puissant et à peine moins flexible que *Common Music*. Par contre, il est un peu moins extensible que *Common Music* car, bien qu'il soit possible d'utiliser directement Lisp pour écrire de nouvelles bibliothèques de fonctions pour *OpenMusic*, il n'est pas possible de s'écarter trop du cadre imposé par celui-ci. Par exemple, il serait impossible de faire en sorte que OpenMusic puisse répondre à des stimuli externes en temps réel. Ce qu'il perd en flexibilité et en extensibilité, il le gagne toutefois en utilisabilité.

Définition 2.6 (*Utilisabilité*)
D'après la norme ISO 9241-11 :1998(F), l'*utilisabilité* est définie comme « le degré selon lequel un produit peut être utilisé, par des utilisateurs identifiés, pour atteindre des buts définis avec efficacité, efficience et satisfaction, dans un contexte d'utilisation spécifié ».

Le troisième outil que nous présentons, *Max* [52], est un autre environnement de programmation visuelle pour la musique algorithmique, prisé par les compositeurs de musique contemporaine, électronique et informatique. Contrairement à *OpenMusic* toutefois, il a été conçu dès le départ pour la composition de musique interactive. En effet, la possibilité d'opérer le logiciel en temps réel à l'aide d'un instrument MIDI connecté à un ordinateur de contrôle[8] fut implémentée très tôt dans le cycle de développement de *Max* [54]. Comme *OpenMusic*, il utilise aussi le paradigme de « patches » pour la composition. Par contre, ses « patches »

[8] À l'époque, les ordinateurs personnels n'étaient pas assez puissants pour permettre la synthèse musicale en temps réel. De ce fait, le coeur de *Max* devait tourner sur une station de travail spécialisée dans le traitement de signaux. Celle-ci n'était toutefois pas adaptée pour la réception d'un flux MIDI en entrée/sortie. Il fallait donc utiliser un autre ordinateur pour convertir le flux MIDI dans un format mieux adapté.

sont évaluées en temps réel. De plus, elles peuvent contenir deux types d'objets : les objets normaux et les objets signaux (aussi appelés objets « tildes »). Ces derniers ont été introduits dans *Max/FTS* et sont évalués en continu à la fréquence d'échantillonnage (44100 Hz dans les versions modernes de *Max*), contrairement aux objets normaux qui ne sont évalués que lorsqu'ils reçoivent un message d'un autre objet. L'implémentation originale des objets signaux était toutefois très limitée et contraignante [53].

Considérons ensuite le logiciel *Pure Data* [53] qui se veut un successeur à *Max*. Il a été conçu par *Miller Puckette*[9] afin de remanier certains aspects de *Max* qu'il considérait problématiques, tout en conservant les forces de ce dernier. Entre autres, les faiblesses suivantes ont été identifiées dans *Max* [53] :

1. Difficulté de stocker des structures de données complexes
2. Difficulté d'intégrer des signaux non-audio (ex. : vidéo, spectres audio) dans le système d'*objets tildes* rigide de *Max/FTS*
3. Difficultés causées par l'obligation de maintenir deux copies de chaque structure de données (une pour l'édition et une pour l'accès en temps réel)
4. Incohérence dans l'implémentation de la relation entre le processus graphique et le processus en temps réel sur les différentes plateformes (Mac vs ISPW)[10]

Mis à part les modifications apportées au logiciel pour suppléer aux quatres limitations identifiées ci-dessus, *Pure Data* fonctionne d'une manière très similaire à *Max* et est en partie interopérable avec celui-ci. Ceci est dû au fait qu'il conserve le paradigme d'interconnexion d'objets (« patches ») et de passage de messages entre ceux-ci utilisé par *Max*.

Les quatres outils que nous venons de décrire ont tous, à priori, été développés par la communauté scientifique afin d'étudier la musique algorithmique et/ou interactive en tant qu'entité indépendante. Ils ne visent donc pas spécifiquement l'industrie du jeu vidéo et sont parfois difficilement acceptés par celle-ci à cause de leur manque d'interactivité (ex. : *Common*

[9] L'auteur original de *Max*.

[10] L'acronyme signifie IRCAM Signal Processing Workstation. Il s'agit d'une plateforme spécialisée dans le traitement de signaux utilisée par l'IRCAM au cours des années 1990s.

Music et *OpenMusic*) et/ou de leur complexité (ex. : *Common Music*, *Max* et *Pure Data*). Il existe toutefois des outils commerciaux qui visent spécifiquement l'industrie du jeu vidéo. Il est toutefois beaucoup plus difficile d'obtenir de l'information aussi précise sur ces outils que sur ceux ayant pris racine dans la communauté académique. Les seuls auxquels nous avons donc pu avoir accès sont *Wwise* et *FMOD Designer*. Puisque ces outils sont très complexes, les paragraphes qui vont suivent se limiteront à l'aspect « composition de musique interactive » de ceux-ci.

Wwise [2] est un outil d'organisation des ressources sonores pour les jeux vidéo, c'est-à-dire qu'il permet de définir les relations entre les états d'un jeu et les événements sonores de celui-ci. De ce fait, il permet déjà d'associer des événements musicaux aux différents états du jeu. Mais, puisque la musique interactive est une forme de ressource sonore très spécialisée, il possède aussi un ensemble de fonctions spécifiques à l'intégration de musique dans les jeux vidéo. En fait, il propose à l'utilisateur d'organiser sa musique en une hiérarchie de sélecteurs (« switch » en anglais), de listes d'écoute (« playlist » en anglais), de segments de musique et de pistes de musique. La figure 2.17 illustre cette hiérarchie. Les sélecteurs sont des conteneurs très généraux qui peuvent aussi bien contenir des sous-sélecteurs que des liste d'écoutes ou encore des segments de musique. Ils sont utilisés par *Wwise* pour définir les relations entre leurs descendants et les états du jeu. Les segments de musique sont beaucoup moins flexibles et ne peuvent contenir que des pistes de musique organisées en une ou plusieurs couches, c'est-à-dire qu'une même piste peut superposer plusieurs fichiers sonores simples pour former un tout plus complexe. Quant aux listes d'écoute, elles servent à spécifier les enchaînements de segments qui produiront les différents thèmes musicaux du jeu. Elles peuvent être organisées en une hiérarchie de groupes séquentiels et aléatoires. Enfin, afin d'atténuer les transitions entre les différents segments de musique, *Wwise* propose toute une panoplie d'outils pour la création de transitions musicales (« segue » en anglais).

FMOD Designer [15] est un autre outil d'organisation des ressources sonores pour les jeux vidéo. Ses versions récentes proposent aussi des fonctions spécifiques à l'intégration de musique interactive dans les jeux vidéo. À cet effet, il propose des fonctionnalités similaires à

FIG. 2.17 – Interface de Wwise

Wwise, à quelques différences près. Plus particulièrement, *FMOD Designer* permet à l'utilisateur d'associer des thèmes musicaux à un ensemble de signaux (« cues » en anglais) provenant du jeu. Ces signaux correspondent aux états de *Wwise*. Comme les listes d'écoutes de *Wwise*, ces thèmes musicaux sont constitués de segments de musique ordonnés dans un ordre précis. Toutefois, contrairement à *Wwise*, et de façon similaire à l'outil que nous présenterons au chapitre 4, *IMTool*, ces segments sont ordonnés en créant des liens entre eux et en spécifiant un ordre de priorité sur ces liens ainsi qu'une condition d'activation pour ceux-ci. Il s'agit donc d'un parfait exemple d'automate étendu avec priorités (mais non-probabiliste).

2.3 Discussion

Dans ce chapitre, nous avons passé en revue différents algorithmes applicables à la composition de musique algorithmique et interactive. Certains de ces algorithmes sont plus adaptés que d'autres à l'utilisation en temps réel et sont utilisés dans des systèmes tels que *Max*, *Pure Data* et *Wwise* alors que d'autres ne sont utilisés que dans les solutions hors-ligne comme *Common Music* et *OpenMusic*. Le tableau 2.1 présente les algorithmes suivant cette classification.

Temps réel		Hors-ligne	
Dés musicaux	Section 2.1.1	Grammaires (L-systèmes)	Section 2.1.3
Chaînes de Markov	Section 2.1.2	Algorithmes génétiques	Section 2.1.4
Automates finis	Section 2.1.6		
Automates cellulaires	Section 2.1.5		

TAB. 2.1 – Classification des algorithmes de composition

Ce premier critère de classification nous permet déjà de réduire le nombre d'algorithmes à considérer. Il nous faut toutefois prendre en compte d'autres critères tels que la flexibilité, la variabilité et l'utilisabilité si l'on veut produire une solution robuste au problème de la musique interactive pour les jeux vidéo.

Définition 2.7 (*Variabilité*)

Selon la 8e édition du dictionnaire de l'académie française, la *variabilité* est définie comme « la disposition habituelle à varier » de certaines choses. Le dictionnaire du trésor de la langue française informatisé[11] [6] ajoute qu'il s'agit du « caractère de ce qui est variable ».

Suivant ces trois critères, on peut éliminer les automates cellulaires car ils souffrent d'un problème majeur d'utilisabilité. Celui-ci découle du fait qu'il est difficile de prévoir le comportement d'un automate en se basant sur sa règle d'évolution locale. Il est donc difficile de trouver des configurations initiales qui donneront des résultats intéressants. Il est aussi difficile de déterminer une règle d'évolution locale pouvant guider un automate cellulaire dans une direction musicale particulière. De même, il est difficile d'établir une correspondance entre les configurations d'un automate et la combinaison espace musical/état d'un jeu. En contrepartie, tous ces facteurs inconnus font que l'on peut dire des automates cellulaires qu'ils offrent beaucoup de variabilité et de flexibilité, bien qu'il s'agisse aussi d'un algorithme complètement déterministe.

Les algorithmes restants ne souffrent en aucun cas de problèmes d'utilisabilité. Il nous faudra donc utiliser d'autres critères pour discriminer entre eux. On peut ainsi éliminer d'emblée les automates finis déterministes et non-déterministes car ils n'offrent ni flexibilité ni variabilité. On peut aussi éliminer les dés musicaux car, bien qu'ils offrent une excellente variabilité, ils n'offrent que très peu de flexibilité. Rappelons-nous en effet que les jeux de dés musicaux prennent généralement la forme d'un ensemble de variations sur les mesures d'une pièce donnée.

De même, on peut éliminer les chaînes de Markov car il s'agit d'un modèle où l'on observe directement les probabilités de transition entre les notes de musique. Elles possèdent donc une excellente variabilité mais très peu de flexibilité et aucun mécanisme pour l'intégration avec l'environnement des jeux vidéo. De plus, elles peuvent être très imprévisibles car il est difficile de prévoir le résultat d'une distribution de probabilités précise lorsque les notes sont directement observables. C'est pourquoi on préfèrera les automates probabilistes qui spécifient

[11] `http://www.cnrtl.fr/portail/`

plutôt les probabilités de transition entre un ensemble d'états non-observables. Selon cette définition, on remarque qu'il existe un lien très fort entre les automates probabilistes et les modèles de Markov de telle sorte qu'on puisse transformer l'un en l'autre sans trop de difficultés. Ils représentent donc une généralisation des chaînes de Markov. Malgré cette amélioration au niveau de l'utilisabilité, les problèmes de flexibilité et d'impossibilité d'intégration avec l'environnement d'un jeu vidéo subsistent.

Enfin, il y a les automates étendus qui n'apportent qu'une solution partielle à notre problème. En effet, bien qu'ils offrent une excellente flexibilité et des mécanismes d'intégration avec l'environnement d'un jeu vidéo, ils n'offrent que très peu de variabilité, excepté celle causée par les changements dans l'environnement du jeu.

En résumé, aucun des algorithmes que nous avons étudié dans ce chapitre ne convient parfaitement à l'utilisation dans un jeu vidéo. Le tableau 2.2 illustre les problèmes que nous avons trouvés à chacun des algorithmes. On constate ainsi qu'il n'existe que trois algorithmes qui ne souffrent pas de problèmes multiples, c'est-à-dire les automates probabilistes, les automates étendus et les automates cellulaires. De ceux-ci, nous ne retiendrons que les deux premiers car les automates cellulaires souffrent d'un problème d'utilisabilité. Nous introduirons donc, au prochain chapitre, un nouvel algorithme combinant les forces des automates probabilistes et des automates étendus tout en remédiant à leurs faiblesses respectives.

Algorithmes	Flexibilité	Variabilité	Utilisabilité
Dés musicaux	X		
Chaînes de Markov	X		X
Automates finis déterministes	X	X	
Automates finis non-déterministes	X	X	
Automates finis probabilistes	X		
Automates étendus		X	
Automates cellulaires			X

TAB. 2.2 – Problèmes des algorithmes de composition en temps réel

En ce qui concerne les outils que nous avons passé en revue, il est clair que notre choix se limite aux outils conçus pour la génération de musique en temps réel, soit : *Max*, *Pure Data* et *Wwise*. Nous avons toutefois déterminé qu'aucun d'entre eux ne correspondait parfaitement à nos besoins. En effet, comme nous l'avons déjà mentionné, il s'avère que l'interface de *Max* et de *Pure Data* peut sembler très complexe pour un compositeur non-initié à la musique algorithmique. De plus, le développement non-commercial de *Max* a cessé depuis 2004. Quant à *Wwise*, bien qu'il réponde à tous nos critères et qu'il s'agisse d'un outil accepté dans l'industrie du jeu vidéo, il ne possède pas toute la flexibilité que nous désirons quant à la communication entre la piste de musique interactive et l'état du jeu. De plus, il s'agit d'un outil commercial et nous n'avons pas accès à son code source, ce qui nous empêche d'étendre ses fonctionnalités. Nous avons donc décidé de développer notre propre outil que nous introduirons au chapitre 4.