Université du Québec à Chicoutimi

Mémoire présenté à
l'Université du Québec à Chicoutimi
comme exigence partielle
de la maîtrise en informatique

offerte à

l'Université du Québec à Chicoutimi
en vertu d'un protocole d'entente
avec l'Université du Québec à Montréal

par

JEAN-MICHEL GILBERT

Algorithmes de génération de musique pour application dans les jeux vidéo

Automne 2008

Bibliothèque
Paul-Émile-Boulet

UQAC
Université du Québec
à Chicoutimi

Mise en garde/Advice

Afin de rendre accessible au plus grand nombre le résultat des travaux de recherche menés par ses étudiants gradués et dans l'esprit des règles qui régissent le dépôt et la diffusion des mémoires et thèses produits dans cette Institution, **l'Université du Québec à Chicoutimi (UQAC)** est fière de rendre accessible une version complète et gratuite de cette œuvre.

Motivated by a desire to make the results of its graduate students' research accessible to all, and in accordance with the rules governing the acceptation and diffusion of dissertations and theses in this Institution, the **Université du Québec à Chicoutimi (UQAC)** is proud to make a complete version of this work available at no cost to the reader.

L'auteur conserve néanmoins la propriété du droit d'auteur qui protège ce mémoire ou cette thèse. Ni le mémoire ou la thèse ni des extraits substantiels de ceux-ci ne peuvent être imprimés ou autrement reproduits sans son autorisation.

The author retains ownership of the copyright of this dissertation or thesis. Neither the dissertation or thesis, nor substantial extracts from it, may be printed or otherwise reproduced without the author's permission.

RÉSUMÉ

Ce mémoire porte sur les algorithmes de génération de musique et leurs applications dans les jeux vidéo. Nous avons choisi ce domaine d'application car, en tant que média interactif complexe, les jeux vidéo représentent un défi de taille pour les compositeurs de musique. Nous commencerons donc par une introduction générale au domaine d'application pour poursuivre avec une présentation ciblée de la problématique. Ensuite, nous passerons en revue une série d'algorithmes de génération de musique issus de la littérature scientifique et basés sur différents modèles mathématiques et nous expliquerons pourquoi aucun d'entre eux ne convient à nos besoins. Nous passerons aussi en revue un éventail d'outils de composition et de mise en séquence de musique interactive. Parmi ces outils, certains ciblent spécifiquement l'industrie du jeu vidéo. Nous expliquerons aussi pourquoi ceux-ci ne correspondaient pas à nos besoins.

Afin de répondre à nos besoins, la troisième partie de ce mémoire présente un nouveau modèle d'automates que nous avons introduit pour la première fois à la conférence Futureplay 2007 : les automates étendus probabilistes. Ce modèle combine la flexibilité des automates étendus et la variabilité des automates probabilistes. Nous décrirons ces modèles d'automates en détails au cours de ce mémoire.

En dernière partie du mémoire, nous présenterons notre outil de composition, introduit aux côtés de notre modèle d'automates étendus probabilistes : IMTool. Cet outil permet à l'utilisateur d'exploiter toute la puissance des automates étendus probabilistes dans un contexte de composition et de mise en séquence de musique pour un jeu vidéo (ou pour toute autre application interactive). En ce sens, celui-ci peut associer des séquences musicales aux états de l'automate, créer des conditions de transition, créer des fonctions de mise-à-jour et associer des probabilités aux transitions. Les conditions de transition et les fonctions de mise-à-jour sont des propriétés des automates étendus et seront expliquées en détails plus loin dans ce mémoire.

Remerciements

En premier lieu, je tiens à remercier mon directeur, Yves Chiricota, pour son soutien, sa patience et le temps qu'il m'a consacré pendant toute la durée du projet. Ses conseils judicieux et son esprit critique ont permis de mener à terme un projet qu'il a accepté de reprendre en plein milieu même si celui-ci avait très mal commencé. J'aimerais aussi spécialement remercier le professeur François Lemieux pour son aide précieuse dans la compréhension des formalismes reliés aux modèles d'automates probabilistes et étendus. Sans son aide, il m'aurait été difficile de passer de la définition intuitive à la définition formelle pour mon modèle. Ensuite, je voudrais aussi remercier tous les autres membres du corps professoral de l'Université du Québec à Chicoutimi qui m'ont encouragé au cours de ce projet et tout spécialement le professeur Abdenour Bouzouane qui m'encourageait souvent lorsqu'il venait prendre des nouvelles de son propre étudiant, mon ami Patrice Roy.

Je tiens aussi à remercier mes parents qui m'ont supporté pendant tout le long de mes études. Aussi, je veux remercier mes amis qui m'ont conseillé et supporté pendant toute la rédaction de mon mémoire : Bruno Bouchard, Martin Charlton, Patrice Roy, Steeve Turcotte et Nathalie Tremblay. De même, je voudrais remercier mes collègues du Groupe de Recherche en Informatique (GRI) de l'Université du Québec à Chicoutimi : Sébastien Noël, Jérôme Lambert, Pierre Delisle, Mohamed Anouar Taleb, Ismaël Coulibali et Arnaud Zinflou avec qui j'ai pu avoir plusieurs conversations intéressantes.

Enfin, je voudrais remercier le Conseil de recherches en sciences naturelles et en génie du Canada (CRSNG) qui, grâce à leur support financier, ont rendu ce projet possible.

Table des matières

Liste des figures

Liste des tableaux

CHAPITRE 1

INTRODUCTION

1.1 La musique et le multimédia

On entend de la musique partout : au cinéma, à la télévision, à la radio, dans les jeux vidéo... Mais à quoi sert-elle ? Il y a bien sûr plusieurs réponses possibles à cette question. Toutefois, des études ont démontré que la trame sonore dans une œuvre multimédia a un effet direct sur la réaction d'un sujet à celle-ci. Plus particulièrement, la trame sonore affecte directement la dimension de *force* de la réaction (faible à forte) et la dimension d'*activité* de la réaction (de passive à active) [38]. Une autre étude a confirmé que lorsque le message véhiculé par les composantes visuelle et auditive d'une œuvre multimédia divergent, celui véhiculé par la composante auditive finit toujours par l'emporter [10]. La publicité pour le jeu *Gears of War*, sorti en 2006, a exploité cet effet à merveille en superposant une musique tranquille et apaisante par dessus une imagerie violente.

Les façons d'utiliser la musique varient selon les médias et les objectifs. Les chercheurs en psychologie de la musique définissent différents modèles de communication pour la musique selon les types de médias [10, 38]. Toutefois, cela dépasse de beaucoup le cadre de ce mémoire pour lequel nous ne ferons qu'une distinction : musique non-interactive (ou linéaire) et musique interactive. Mais peu importe son type, on peut dire que la musique ajoute une dimension affective à une œuvre qu'il serait difficile d'aller chercher sans celle-ci. Selon *Lipscomb* [38], la musique est aussi reliée à la dimension *évaluative* (bonne à mauvaise) de la réaction d'un sujet envers une œuvre. En fait, celui-ci stipule que cette réaction serait entièrement déterminée

par la congruence entre les composantes auditive et visuelle selon une mesure établie par une composante comparative, quelle qu'elle soit.

1.1.1 Le problème des médias interactifs

À l'exception des jeux vidéo, toutes les formes de médias que nous avons présentées à la section précédente ont une durée prédéterminée. En ce sens, il est aussi possible de prédéterminer la quantité de musique qu'il sera nécessaire de composer pour mettre ceux-ci en musique.

À l'inverse, les médias interactifs n'ont pas de durée prédéterminée. En fait, pour la plupart, l'ordre d'exécution des scènes présentées à l'utilisateur n'est même pas prédéterminé. Il est aussi possible que certaines scènes soient répétées plusieurs fois — on peut penser à une séquence de menus — et que d'autres ne soient jamais présentées à l'utilisateur ; ce qui arrive souvent avec les médias interactifs complexes tels que les livres interactifs et les jeux vidéo. À cause de cela, il est difficile de prévoir la quantité de musique nécessaire pour créer une expérience musicale appropriée pour un média interactif. En effet, si on utilise trop peu de musique, l'auditeur risque de se fatiguer à cause de la répétition. Par contre, si on utilise trop de musique pour le temps de visionnement d'un utilisateur moyen, on risque non seulement de payer pour la composition de musique qui ne sera jamais entendue mais aussi de diluer l'expérience musicale et donc l'effet émotif potentiel de la trame sonore. Ceci est dû au fait que la répétition et l'établissement de thèmes et de contre-thèmes forts et mémorables sont aussi une partie importante de l'expérience musicale.

Pour résoudre ce problème, l'industrie du jeu vidéo[1] a adopté plusieurs solutions au fil des années. Une technique extrêmement primitive mais populaire au début des années 80 consistait à ne pas composer sa propre musique et à faire jouer en boucle une pièce classique ou folklorique pendant tout le jeu. Par exemple, *Black Hawk* [63] sorti en 1984 utilisait le thème de *La chevauchée des Walkyries* de *Wagner* pendant toute la durée du jeu. Le seul moment de

[1] Cette industrie sera notre intérêt principal au cours de ce mémoire car les jeux vidéo comptent parmi les médias interactifs les plus coûteux à produire. Le coût de production de certains jeux rivalise même avec le coût de production de certains films Hollywoodiens.

répit laissé au joueur était lorsque le jeu coupait la musique. Une technique similaire, tout aussi primitive et populaire pendant la même période consistait à composer un seul thème musical qu'on laissait ensuite jouer en boucle. Le jeu *Northstar* [23] sorti en 1988 est un des nombreux exemples de cette époque à avoir utilisé cette technique. La technique la plus évoluée qui a pris naissance à cette époque et qui est encore populaire aujourd'hui consiste à composer un thème pour chaque changement de scène majeur (l'unique point à prendre en considération dans le cas d'un livre interactif) et/ou pour tout autre élément important dans le cas d'un jeu vidéo.

Cette dernière technique est encore populaire aujourd'hui car elle est simple et fournit un bon rapport qualité/prix. De plus, avec la montée en puissance des processeurs modernes et l'augmentation des capacités de stockage, il est aujourd'hui possible de stocker des pistes de musique intermédiaires, en utilisant par exemple une piste par instrument, et de refaire le mélange dynamiquement tout en appliquant des effets spéciaux sur chaque piste individuelle. Cette technique a été utilisée avec succès par plusieurs jeux, notamment *The Mark of Kri* [58], et on peut facilement voir qu'elle fournit un bien meilleur rapport qualité/prix car elle permet de créer du tout nouveau matériel en récupérant du matériel existant. Concrètement, cette technique permet de changer dynamiquement le volume d'un seul violon, d'ajouter de la distorsion à une guitare et de changer la partition jouée par un piano ; tout ça sans toucher aux autres instruments du mélange[2]. La méthode que nous allons présenter plus loin dans ce mémoire partage le même but que cette technique : créer du nouveau matériel à partir de matériel existant.

Enfin, il existe une autre famille de solutions : celles basées sur la composition algorithmique (voir définition 1.1). La popularité de ces techniques a cru et décru au fil des années. Par exemple, au milieu des années 80s, lorsqu'il fallait être compositeur *et* programmeur pour pouvoir tirer un peu de musique des ordinateurs et des consoles de jeux,

[2]Remarquons que le fait de changer le volume et/ou la partition d'un instrument sans toucher aux autres était déjà possible auparavant en utilisant des séquences MIDI. Toutefois, l'ajout de distorsion doit absolument se faire à l'aide de filtres DSP (acronyme anglais pour Digital Signal Processing, ce qui signifie Traitement de Signaux Numériques en français).

certains compositeurs étaient prompts à expérimenter avec la composition algorithmique. Par exemple, la musique du jeu *Ballblazer* [39], composée par *Peter Langston* [37], utilise une variante, recherchée indépendamment, de la technique des dés musicaux que nous aborderons plus en détails au chapitre 2. Par contre, au milieu des années 90s, l'utilisation de ces techniques a presque totalement cessé avec l'obsession des trames sonores « comme à la radio » pour ne réapparaître que récemment dans les séquenceurs les plus sophistiqués où elle se combine avec la dernière technique présentée au paragraphe précédent. Il n'en demeure pas moins que ces techniques ont plusieurs limitations que nous verrons à la section suivante. De plus, elles sont difficiles à maîtriser et elles réduisent le côté humain de la composition musicale, ce qui fait qu'elles demeurent encore taboues auprès de plusieurs musiciens.

Définition 1.1 (*Composition algorithmique*)
Les techniques de *composition algorithmique* sont des méthodes de composition qui impliquent que la musique est, en tout ou en partie, composée par un algorithme, que celui-ci soit informatisé ou non, paramétrable par un compositeur ou non.

1.2 Problématique

Les méthodes de composition que nous avons présentées à la dernière section ont toutes des avantages et des inconvénients. Certaines ont été adaptées des médias linéaires, c'est-à-dire non-interactifs ; d'autres ont été spécialement conçues pour les médias interactifs. La ligne qui sépare les deux est très mince et dépend de l'usage que l'on prévoit faire de la musique. Si l'on prévoit utiliser une musique précomposée, soit manuellement ou à l'aide d'un algorithme, de façon plus ou moins statique, en n'effectuant des changements qu'aux points importants de l'œuvre interactive, il s'agit d'une technique adaptée à partir d'un média linéaire, notamment le cinéma. Par contre, si l'on prévoit recomposer à chaque instant la musique selon les actions de l'utilisateur, il s'agit d'une technique spécialement conçue pour un média interactif. Cela ne signifie toutefois pas qu'on soit obligé de se limiter à de la musique purement algorithmique ; on peut aussi utiliser de la musique hybride comme le cas du jeu *The Mark of Kri* [58].

Pour ce qui est des méthodes adaptées des médias linéaires, les avantages sont clairs : elles sont faciles à utiliser, faciles à comprendre et offrent souvent un bon rapport qualité/prix. Elles ont toutefois comme inconvénients de ne pas être toujours adaptées à ce qui est affiché à l'écran et peuvent, à long terme, fatiguer l'utilisateur à cause de la répétition. Il nous apparaît clair que ces méthodes, bien qu'elles présentent un certain attrait de par leur simplicité, ne sont pas à la hauteur lorsqu'il s'agit de fournir une trame sonore complètement interactive.

Quant aux méthodes de composition algorithmique, elles présentent certains défis, généralement par rapport à l'un de ces cinq facteurs : utilisabilité, performance, variabilité, flexibilité, esthétisme du résultat et souvent par rapport à plusieurs de ceux-ci. Par exemple, bien que le résultat obtenu grâce aux algorithmes génétiques a des chances d'être plutôt bon (rappelons-nous qu'il s'agit d'un algorithme d'optimisation), les algorithmes génétiques sont difficiles à paramétrer pour un utilisateur moyen et ont besoin de plusieurs centaines d'itérations avant de *peut-être* converger sur un bon résultat. D'un autre côté, on peut penser à utiliser des algorithmes fractals qui sont généralement performants et plus ou moins utilisables (dépendamment de l'algorithme). Par contre, le résultat de ces algorithmes est souvent très imprévisible et plus souvent qu'autrement déplaisant à l'oreille. On peut aussi penser à utiliser des structures auto-réplicantes comme les automates cellulaires qui, bien paramétrées, ont le potentiel de générer des motifs ressemblant à de la musique. Toutefois, ils sont difficiles à utiliser. Aussi, on peut penser à utiliser différents types d'automates finis qui ont tous leurs propres limites. Remarquons que, dans le cas des automates, l'esthétisme du résultat final dépend largement de la façon dont on associe les configurations de ceux-ci à l'espace musical. Nous couvrirons ces sujets plus en détail au chapitre 2. Enfin, on peut utiliser une méthode hybride qui génère la structure de la musique de façon algorithmique mais qui utilise des segments de musique précomposés. C'est la méthode qui a été utilisée dans *The Mark of Kri* [58] et c'est la méthode que nous nous proposons d'employer dans ce mémoire.

À cet effet, nous présenterons aux chapitres 3 et 4 une architecture logicielle que nous avons précédemment présentée [8] à la conférence FuturePlay 2007, une conférence internationale sur le futur des jeux vidéo. Cette architecture logicielle, que nous décrirons

au chapitre 3, est basée sur un nouveau type d'automates qui combine la variabilité des transducteurs probabilistes (section 2.1.6.3) et la flexibilité des automates étendus (section 2.1.6.5) dont nous parlerons au chapitre 2. Ensuite, au chapitre 4, nous introduirons *IMTool*, un outil que nous avons développé afin de permettre aux compositeurs d'utiliser nos automates étendus probabilistes pour créer de la musique interactive en chaînant des segments de musique plus courts par des règles (pour créer de la structure) et par des transitions aléatoires (pour créer de la variabilité). Il va de soi qu'avec une telle architecture, il est possible de générer plus de types de musique qu'avec un algorithme difficilement paramétrable.

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 $(i_0, \ldots, i_{k-1}) \in S^k, \sum_{j \in S} Pr(j \mid i_{k-1}, \ldots, i_0) = 1$

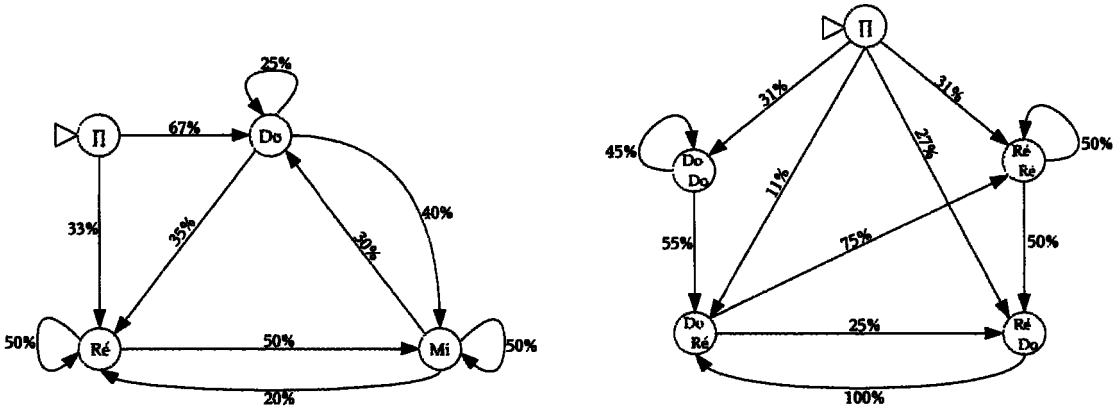
Enfin, on remplace la loi initiale par un vecteur de probabilités spécifiant la probabilité initiale de se trouver dans chaque tuple d'états à k dimensions. Formellement, ceci nous donne :

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

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

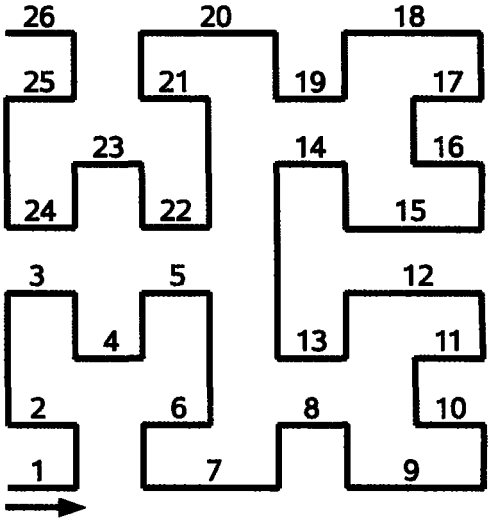


FIG. 2.8 – Traversée de la courbe d'Hilbert

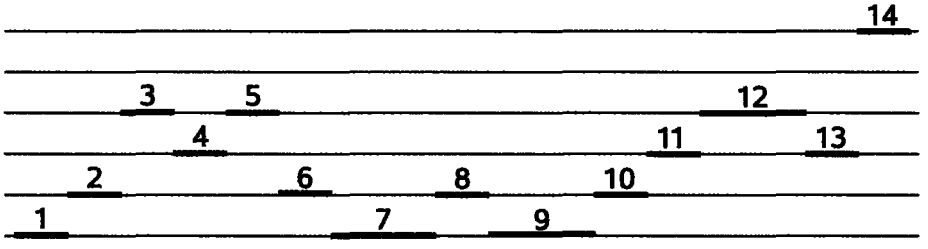


FIG. 2.9 – La partition associée en notation « piano roll »

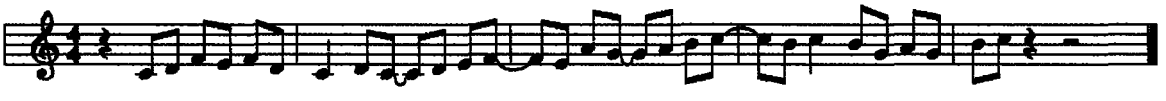


FIG. 2.10 – La partition associée en notation standard

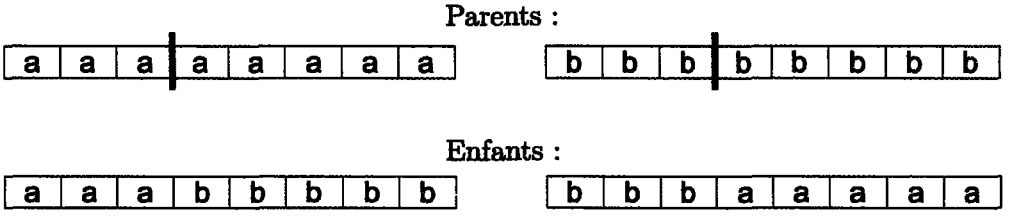


FIG. 2.11 – Processus de recombinaison

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.

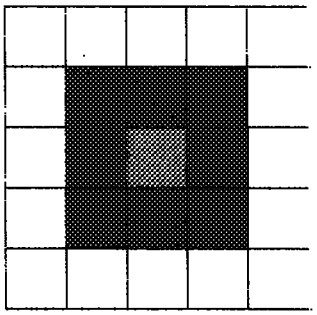


FIG. 2.12 – Exemple de voisinage

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

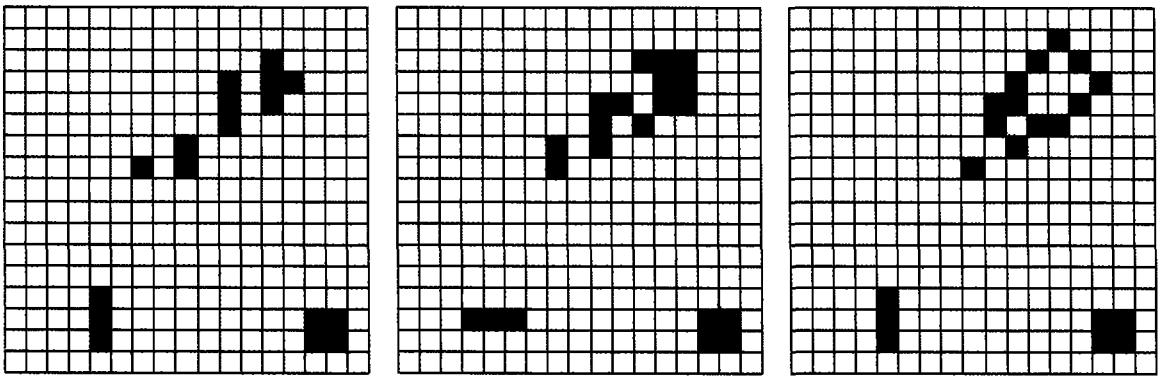
$$\forall x \in R : C_{t+1}(x) = \begin{cases} C_t(x) & si \sum_{i \in V} C_t(x+i) = 2, \\ 1 & si \sum_{i \in V} C_t(x+i) = 3, \\ 0 & dans\ tous\ les\ autres\ cas. \end{cases} \tag{2.6}$$

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.

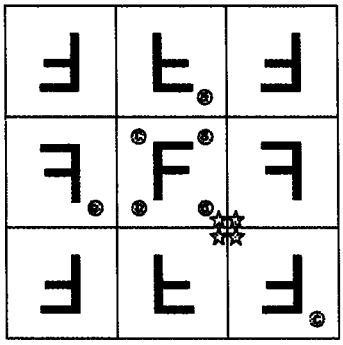


FIG. 2.14 – Grilles miroirs et projection de cellules

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.

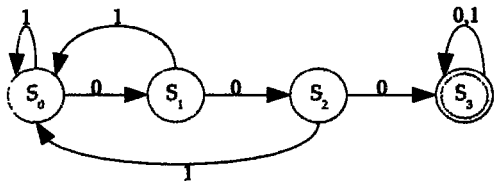


FIG. 2.15 – Automate fini déterministe

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,

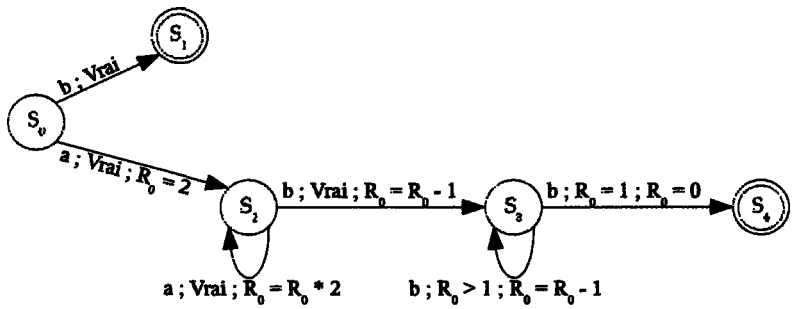


FIG. 2.16 – Automate étendu acceptant le langage $a^n b^{2^n}$ qui ne peut être représenté par un automate traditionnel

$f \in C$ 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.

Chapitre 3

Automates étendus probabilistes

Nous avons vu au chapitre précédent plusieurs algorithmes de génération de musique. Toutefois, aucun d'entre eux ne s'est avéré totalement satisfaisant. Certains sont trop difficiles à paramétrer, d'autres n'offrent pas assez de flexibilité à l'utilisateur, d'autres encore ne permettent pas de créer assez facilement des variations sur un même thème, et enfin d'autres ne sont pas adaptés à l'utilisation en temps réel. C'est afin de satisfaire tous ces critères que nous introduisons un nouvel algorithme de génération de musique. Dans la première partie de ce chapitre, nous présenterons la description formelle de cet algorithme. Ensuite, nous présenterons une implémentation de l'algorithme adaptée pour le jeu vidéo. Enfin, nous introduirons un exemple d'application que nous reprendrons dans les chapitres de mise en œuvre.

3.1 Retour sur les automates étendus

Au chapitre 2, nous avons souligné un problème existant avec le modèle original d'automates étendus tel que défini par *Cheng et Krishnakumar* [7]. Ce problème provient du fait qu'à moins de 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, les automates étendus sont non-déterministes. Ce problème a aussi été constaté par *Krishnakumar* [36] dans un article subséquent dans lequel il indique que les automates étendus sont non-déterministes.

Dans ce mémoire toutefois, nous préfèrerons une solution différente à ce problème, inspirée par *Huang et Lo* [30], qui consiste à déterminiser les automates étendus en introduisant

une valeur de priorité dans chaque fonction d'activation C_i, telles que définies au chapitre 2. Pour ce faire, nous redéfinirons celles-ci par $C_i : V \rightarrow \{0, p\}$ où $p \in \mathbb{N}^*$ est la valeur de priorité de la condition. Ceci signifie que chaque fonction C_i est une fonction booléenne où 0 représente faux et p représente vrai. Ainsi, le fonctionnement d'un automate étendu avec priorités se comprend de cette manière :

Soit q, r et s des états de l'automate, a un symbole appartenant à l'alphabet d'entrée Σ, $f, g \in C$ des fonctions d'activation, et $u, v \in \phi$ des fonctions de mise à jour (voir chapitre 2). Ainsi, si $\langle q, \vec{x} \rangle$ est la configuration courante de l'automate et que $\delta(q, f, a) = \langle r, u \rangle$ et $\delta(q, g, a) = \langle s, v \rangle$ sont deux transitions possibles considérant que le prochain symbole sur le ruban d'entrée est a, que $f(\vec{x}) \neq 0$, et que $g(\vec{x}) \neq 0$, alors l'automate passera dans la configuration $\langle r, u(\vec{x}) \rangle$ si et seulement si $f(\vec{x}) > g(\vec{x})$, sinon il passera dans la configuration $\langle s, v(\vec{x}) \rangle$. On peut définir $f(\vec{x}) = g(\vec{x}) \neq 0$ comme condition d'erreur afin d'éviter de simplement déplacer le problème du non-déterminisme.

Cette nouvelle définition permet de définir plus facilement des automates étendus avec un grand nombre de registres et un grand nombre de transitions sortant du même état.

3.2 Définition

Nous introduisons ici un nouveau type de transducteurs que nous appelons automates étendus probabilistes. Ils combinent la flexibilité des automates étendus et la diversité engendrée par les automates finis probabilistes, tels que vus au chapitre 2. Afin de simplifier la définition d'un automate étendu probabiliste, nous allons supposer que l'état de l'automate est connu à chaque instant t du processus (comme pour les automates étendus). Ainsi on peut les définir de manière analogue aux automates étendus avec priorités :

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, G est une fonction totale de S dans Γ, V un espace n-dimensionnel $V_1 \ldots V_n$, ϕ un ensemble de transformations ϕ_i tel que $\phi_i : V \rightarrow V$, et C un ensemble de fonctions C_i tel que $C_i : V \rightarrow mathbbN$. Soit aussi δ une fonction partielle de $S \times C$ dans $(\mathbb{R} \times \phi)^S$ tel que $(\mathbb{R} \times \phi)^S$ représente un vecteur stochastique augmenté d'une

fonction de mise à jour (appartenant à $\phi \cup Indéfini$) pour chaque état de destination possible. Formellement, un tel vecteur augmenté est défini comme un vecteur $|S|$-dimensionnel de paires ordonnées $\langle Pr(S_j) \in \mathbb{R}, \varphi_j \rangle$ tel que :

- $0 \leq Pr(S_j) \leq 1$,
- $\sum_{j=1}^{|S|} Pr(S_j) = 1$,
- $\varphi_j \in \phi$ si $0 < Pr(S_j) \leq 1$, et
- φ_j est indéfini sinon.

On dit alors que le décuplet $\langle S, S_0, F, \Sigma, \Gamma, V, C, \phi, \delta, G \rangle$ représente un automate étendu probabiliste où S_0 est état initial, F l'ensemble des états finaux, V l'ensemble de registres, C l'ensemble des fonctions d'activations, ϕ l'ensemble des fonctions de mise à jour, δ la fonction de transition, et G la fonction de sortie. On remarque immédiatement que la seule différence avec les automates étendus est que la fonction de transition prend maintenant en compte la probabilité de transition vers chaque état de l'automate. L'exemple suivant illustre le fonctionnement d'un automate étendu probabiliste.

Considérons l'automate étendu probabiliste à trois états (q_0, q_1, q_2) illustré à la figure 3.1. On remarque que l'ensemble C de ses fonctions d'activation est constitué des deux fonctions $f(\vec{x})$ et $g(\vec{x})$ et que l'ensemble ϕ de ses fonctions de mise à jour est constitué des trois fonctions $u(\vec{x}), v(\vec{x})$ et $w(\vec{x})$. De plus, l'état q_0 possède deux transitions : $\delta(q_0, f, a) = \{\langle 0, Indéfini \rangle, \langle 1, u \rangle, \langle 0, Indéfini \rangle\}$ et $\delta(q_0, g, a) = \{\langle 0.25, v \rangle, \langle 0, Indéfini \rangle, \langle 0.75, w \rangle\}$. Supposons maintenant que $\langle q_0, \vec{x} \rangle$ est la configuration initiale et que a est le prochain caractère sur le ruban d'entrée de l'automate. Dans ce cas, la configuration suivante de l'automate est déterminée comme suit :

- Si $f(\vec{x}) = g(\vec{x})$, la configuration suivante n'est pas définie,
- Si $f(\vec{x}) > g(\vec{x})$, l'automate passe dans la configuration $\langle q_1, u(\vec{x}) \rangle$ avec probabilité 100%,
- Sinon, l'automate passe dans la configuration $\langle q_0, v(\vec{x}) \rangle$ avec probabilité 25% ou dans la configuration $\langle q_2, w(\vec{x}) \rangle$ avec probabilité 75%.

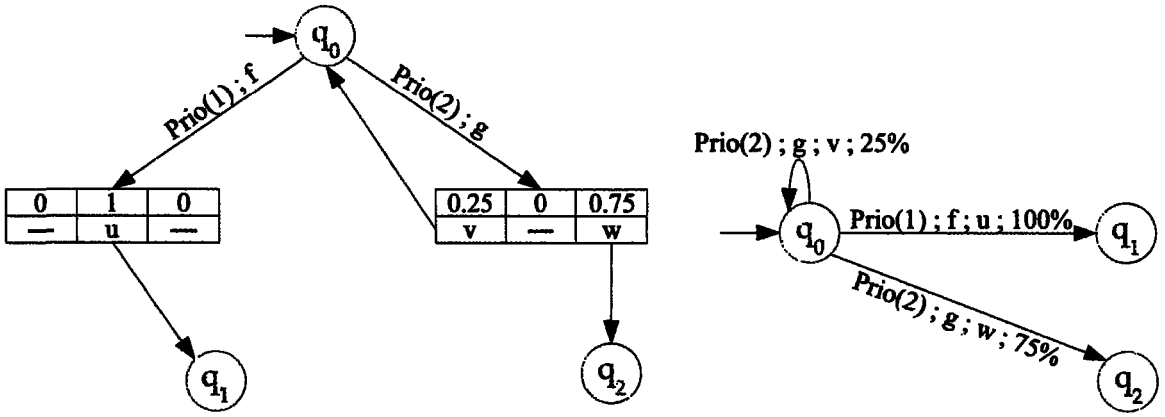


FIG. 3.1 – Deux représentations pour le modèle d'automates étendus probabilistes. Celle de gauche est conforme au modèle mais moins intuitive que celle de droite, qui sera donc utilisée pour toutes les figures subséquentes dans ce mémoire.

Nous pouvons maintenant montrer que les automates étendus probabilistes sont une généralisation appropriée des automates finis probabilistes. Il suffit de poser un automate ayant $f(\vec{x}) = 1$ comme seule fonction d'activation et $\varphi(\vec{x}) = \vec{x}$ comme seule fonction de mise à jour. Les registres sont alors inutilisés et on obtient un transducteur probabiliste standard.

De même, nous pouvons montrer que les automates étendus probabilistes sont une généralisation appropriée des automates étendus avec priorités. Il suffit de poser un automate où l'on impose que les probabilités de transition soient 1 ou 0. Il s'ensuit que pour tout état $q \in S$ et pour toute condition $f \in C$, la fonction de transition $\delta(q, f)$ est déterministe puisqu'elle retourne un vecteur stochastique qui doit donc satisfaire la propriété suivante : $\sum_{j=1}^{|S|} Pr(S_j) = 1$. On obtient la définition d'un automate étendu avec priorités standard en remarquant que la fonction de mise à jour dans le vecteur stochastique augmenté n'est définie que pour l'état ayant une probabilité de 1.

3.2.1 Conditions de bonne formation d'un automate étendu probabiliste

Le modèle d'automates étendus probabilistes comporte un problème inhérent au modèle d'automates étendus. Il s'agit du cas où, sous certaines conditions, aucune transition partant d'un état $q \in S$ ne peut être activée ; ce qui est le cas pour l'état A dans la figure 3.2 lorsque $V_1 \leq 4$. On dit d'un tel état qu'il n'est pas complet puisqu'il existe certaines configura-

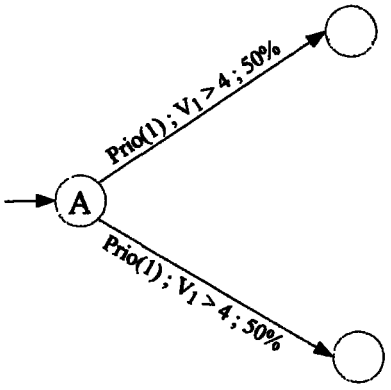


FIG. 3.2 – Aucune transition activable lorsque $V_1 \leq 4$

tions de l'automate où aucune transition sortant de l'état A n'est activable. Malheureusement, cette erreur est difficile à corriger car il n'existe pas de solution unique. De plus, il ne s'agit pas toujours d'une erreur ; certaines configurations de l'automate sont parfois impossibles et c'est pourquoi δ est une fonction partielle.

De plus, sans constituer une malformation à proprement parler, il est aussi possible de créer des transitions inactivables peu importe l'état de l'automate. Par exemple, sur la figure 3.3, on remarque que la transition A porte une contradiction logique comme fonction d'activation ($V_1 < V_1$). Quant aux transitions C et D, elles sont inactivables car elles sortent d'un état marqué comme état final (rappelons-nous que, dans notre implémentation, les états finaux amènent automatiquement l'automate à s'arrêter). Toutes ces transitions peuvent êtres supprimées sans risque de changer le comportement final de l'automate.

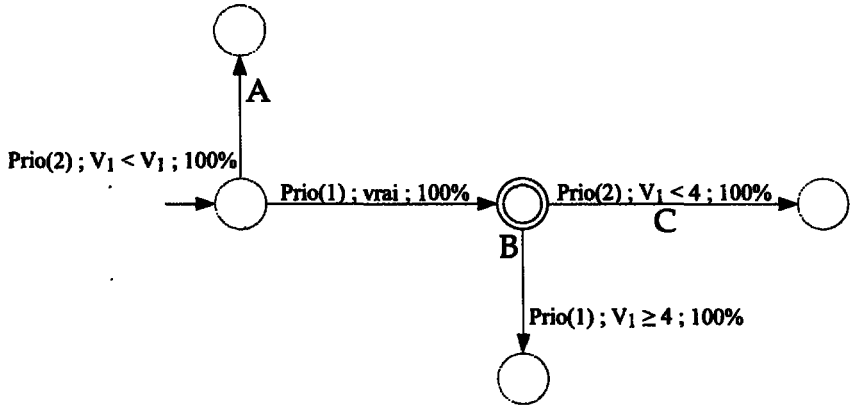


FIG. 3.3 – Transitions inactivables peu importe l'état de l'automate

3.3 Implémentation

Jusqu'ici, par souci de clarté, nous avons présenté le modèle sans faire référence aux jeux vidéo. Dans les lignes qui vont suivre, nous allons adapter celui-ci à une situation d'utilisation en temps réel, ce qui correspond au contexte des jeux vidéo. À cet effet, nous allons ajouter de nouveaux critères à ceux précédemment énumérés au chapitre 2 :

- Communication bi-directionnelle entre l'automate et un processus externe (ex. : jeu, outil de composition, etc.),
- Simplicité.

Définition 3.1 (*Simplicité*)
La *simplicité* est la qualité de ce qui est facile à comprendre ; le caractère de ce qui est facilement utilisable ou réalisable.[1] [6]

À priori, l'introduction de ces nouveaux critères peut sembler redondante puisque l'automate possède déjà un mécanisme lui permettant de communiquer avec des processus externes : ses rubans d'entrée et de sortie. Cette solution peut même sembler idéale par sa simplicité apparente. Toutefois, tel que montré par *Cheng et Krishnakumar* [7], il existe deux manières de traiter l'entrée :

1. En la consommant sur les transitions,
2. En la transférant dans les registres.[2]

Ceci est problématique puisque les deux méthodes n'offrent pas le même niveau de flexibilité. En effet, en transférant l'entrée dans un registre, la deuxième méthode nous permet d'appliquer des fonctions de mise à jour sur celle-ci. De plus, seule cette dernière méthode nous permet de garder la trace de plusieurs paramètres évoluant en parallèle dans l'environnement du jeu vidéo (le processus externe) sans ambiguïté. Remarquons toutefois qu'il nous faut pour cela multiplier les rubans d'entrées, c'est-à-dire qu'il nous faut un ruban par paramètre ou,

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

[2] Notons que cette deuxième option ne change en rien le comportement de la tête de lecture par rapport à la première option. L'article de *Cheng et Krishnakumar* [7] montre un exemple qui combine, en fait, les deux méthodes.

dans le cas limite, un ruban par registre de l'automate. Ceci vaut aussi pour les rubans de sortie si l'on veut permettre au jeu de récupérer le contenu des registres de cette manière. On en conclut donc que l'utilisation des rubans d'entrée et de sortie de l'automate pour la communication avec un processus aussi complexe qu'un jeu vidéo n'est pas une si bonne idée qu'il n'y paraît à l'origine. Nous proposerons plus loin une solution fonctionnelle à ce problème.

Mais avant, il nous faut montrer une propriété intéressante qu'acquièrent les automates étendus lorsque l'espace de calcul d'un ou de plusieurs registres est un sur-ensemble de l'alphabet d'entrée. Dans ce cas, il devient possible, en appliquant une transformation similaire à celle présentée à la figure 3.4, de remplacer toutes les transitions par des transitions spontanées. Bien sûr, cela introduit une quantité importante d'états artificiels et complexifie du même coup l'automate mais la solution que nous introduisons au prochain paragraphe sait conserver cette propriété désirable tout en corrigeant ce problème.

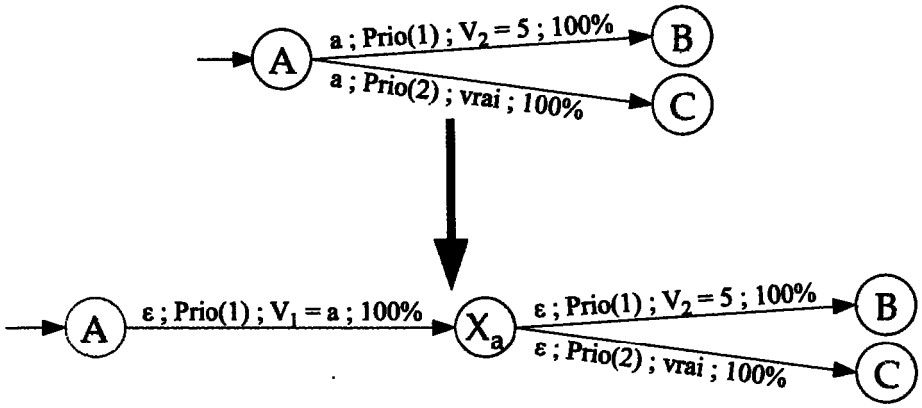


FIG. 3.4 – Suppression des transitions non-spontanées dans un automate étendu (probabiliste)

Notre solution accorde au jeu vidéo un accès direct aux registres de l'automate. Ainsi, elle s'affranchit du ruban d'entrée et des transitions non-spontanées d'une façon qui permet de réduire le nombre d'états artificiels. Enfin, bien que nous aurions aussi pu éliminer les états finaux, puisqu'il n'y a plus d'entrée à reconnaître, nous avons préféré leur accorder une nouvelle fonction : celle de points d'arrêts forcés (états forçant l'arrêt de l'automate).

Afin d'appliquer notre concept, nous avons implémenté un outil de composition et un moteur de musique (l'automate en réalité) entièrement découplés l'un de l'autre. Comme

chacun possède son état propre, il nous faut fréquemment mettre à jour chaque état à partir des informations pertinentes provenant de l'autre. Le fait d'avoir un accès direct aux registres nous a permis d'implémenter facilement ce processus grâce à deux fonctions seulement : `GetRegister` et `SetRegister` (voir chapitre 4).

Enfin, considérant que notre objectif final consiste à produire de la musique pour un jeu vidéo, il nous a fallu encoder la musique pour le ruban de sortie de l'automate. À cet effet, nous avons choisi de définir l'alphabet de sortie de l'automate (Γ) comme un sous-ensemble de $\mathbb{N}$ où chaque nombre correspond à un identificateur pour un segment musical stocké dans une structure externe. Bien entendu, ces segments musicaux devront, eux-aussi, être encodés dans un format utilisable par l'ordinateur. La section suivante compare deux formats d'encodage possible au niveau de la performance et de la flexibilité.

3.3.1 Encodage des segments musicaux

Nous présentons maintenant différentes options par rapport au format d'encodage pour les segments musicaux. Celui-ci affecte directement la performance et la flexibilité du système résultant, deux critères très importants pour l'industrie du jeu vidéo. Pour des fins de commodité, nous avons choisi de ne retenir que les deux formats d'encodage les plus populaires dans l'industrie du jeu aujourd'hui : le format MIDI[3] et le format WAV (voir annexe A pour une description du format MIDI).

L'encodage de la musique sous la forme d'une séquence d'échantillons sonores non compressés est certainement l'option la plus simple car il s'agit de la forme finale que la musique devra adopter avant d'être transmise à la carte de son. Puisque ce format représente l'information sonore à son plus bas niveau, c'est-à-dire comme une séquence d'échantillons, on peut facilement déduire qu'il s'agit du format le moins flexible. Ceci découle du fait que puisque toute l'information musicale de haut niveau est perdue, il devient difficile de modifier le flux musical de façon réaliste, même en appliquant des algorithmes de traitement de signaux complexes. En contrepartie, l'application d'effets spéciaux s'en trouve facilitée. Il existe bien

[3]De nos jours, le format MIDI est surtout populaire pour le développement de jeux pour cellulaires ainsi que comme format de travail pour la composition.

sûr des solutions partielles à ce problème comme l'enregistrement multi-pistes, soit la technique d'enregistrer chaque instrument de musique dans un fichier séparé et de reporter le mixage de ceux-ci au moment de l'exécution du jeu vidéo. De cette façon, le moteur de son du jeu vidéo peut appliquer des effets spéciaux sur chaque instrument indépendamment des autres et on récupère une partie de la flexibilité perdue. Certaines modifications, comme les changements de tempo ou de notes, demeurent difficiles.

Une séquence au format MIDI est formée d'une série de commandes à transmettre à un synthétiseur musical. Ces commandes sont fort simples et espacées dans le temps (ex. : activer note, désactiver note, changer instrument, etc). Ceci fait en sorte que ces séquences sont faciles à modifier en temps réel puisqu'il suffit de modifier le flux de commandes. Toutefois, ceci implique aussi qu'il y a une étape supplémentaire entre la lecture de la séquence MIDI et le mixage de la séquence avec les autres effets sonores utilisés par le jeu, soit la conversion, par un synthétiseur, de la séquence MIDI en séquence d'échantillons WAV. De ce fait, le problème du choix du type de synthétiseur s'impose. Ce problème est important car il affecte directement la performance du logiciel et la qualité du fichier WAV produit. De façon réaliste, deux options s'offrent à nous : utiliser le synthétiseur interne de l'ordinateur ou utiliser un synthétiseur logiciel.

La première option est la plus performante, plus encore que l'utilisation de musique échantillonnée. Ceci est dû au fait que le processeur central de l'ordinateur n'intervient pas dans cette option, même pas au niveau du mixage avec les effets sonores car le synthétiseur interne possède un canal dédié sur la carte de son, c'est-à-dire que la musique produite par celui-ci est jouée simultanément avec les effets sonores sans besoin d'être mélangée. Toutefois, la qualité des synthétiseurs internes peut varier d'un ordinateur à l'autre. De ce fait, il n'existe aucune garantie qu'une même séquence MIDI sera reproduite de la même façon sur différents ordinateurs, ce qui fait que cette solution est plus ou moins appropriée à notre contexte.

L'autre option, l'utilisation d'un synthétiseur logiciel, est plus intéressante pour les jeux car elle garantit une reproduction toujours fidèle de la musique d'un ordinateur à l'autre. De plus, elle offre beaucoup plus de flexibilité que la précédente puisque, dans ce cas, nous

sommes en contrôle de la manière dont le synthétiseur procède pour générer les instruments qui serviront à transformer la séquence MIDI en signal échantillonné. À cette fin, nous avons choisi d'utiliser Fluidsynth[4] [26], un synthétiseur implémentant la technique la plus utilisée de nos jours, soit la synthèse par table d'ondes qui consiste en la modulation et l'interpolation d'échantillons d'instruments réels échantillonnés afin d'obtenir les tonalités désirées [12,26]. Ceci implique que les banques d'instruments d'un tel synthétiseur sont entièrement personnalisables par le compositeur qui est alors libre de fournir ses propres banques d'échantillons. Pour ce faire, nous avons utilisé le format SoundFont (nommé par analogie aux polices de caractères en typographie). Il s'agit d'un format de fichier pouvant contenir jusqu'à plusieurs gigaoctets d'échantillons et pouvant décrire jusqu'à 16384 instruments en superposant ces échantillons et en leur appliquant différents paramètres (ex. : l'enveloppe de volume, l'enveloppe de modulation, la quantité de chorus, la quantité de réverbération, etc). Ces instruments pourront ensuite être utilisés par le synthétiseur pour effectuer le rendu des fichiers MIDI. Remarquons que les banques d'instruments personnalisées d'un compositeur ne sont pas obligées de respecter le standard « General MIDI » qui spécifie un jeu de 128 instruments (ou 256 dans le cas du standard « General MIDI 2 ») que tous les synthétiseurs doivent fournir afin que toutes les séquences MIDI puissent être reproduites de manière prévisible sur tous les synthétiseurs[5]. Remarquons toutefois que, puisque la transformation en temps réel du flux MIDI en signal échantillonné est une opération coûteuse en temps de processeur, il s'agit de l'option la moins performante. Elle n'est donc pas adaptée à l'utilisation dans un jeu vidéo. Toutefois, sa flexibilité la rend idéale pour l'utilisation dans un outil de composition. C'est ce que nous verrons au chapitre 4.

3.4 Exemples

Afin d'illustrer notre modèle, cette section présente quelque cas d'utilisation de musique interactive et/ou algorithmique tirés de jeux réels et explique comment on peut modéliser chacun d'eux à l'aide d'automates étendus probabilistes. Afin d'augmenter la clarté

[4] `http://www.nongnu.org/fluid/`

[5] Dans notre implémentation, la responsabilité incombe à l'utilisateur de fournir une banque d'instruments compatible « General MIDI » si tel est son souhait.

de nos figures, nous nous sommes concentrés sur des éléments spécifiques de la trame sonore des jeux concernés.

Exemple 3.1 On sait déjà que la plupart des jeux font en sorte que l'état du jeu influence la musique. Toutefois, une possibilité intéressante, souvent inexploitée dans les jeux, consiste à faire en sorte que la musique agisse sur l'état du jeu de façon rétroactive. Afin d'illustrer cet effet, prenons l'exemple de *New Super Mario Bros.* [46] dans lequel les ennemis font des pas de danse au rhythme de la musique. Une manière d'implémenter cet effet consiste à séparer chaque pièce de musique en deux segments : *avant pas de danse* et *pendant/après pas de danse* puis d'utiliser un automate étendu probabiliste comme celui représenté à la figure 3.5 dont le but de la fonction de mise à jour est d'indiquer au jeu qu'il est temps pour les ennemis d'effectuer leurs pas de danse. Le jeu remet ensuite le registre à 0.

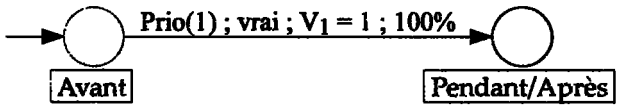


FIG. 3.5 – Rétroaction de la musique vers l'état du jeu

Exemple 3.2 Une technique pratiquement universelle dans tous les médias où la musique joue un rôle accompagnateur consiste à changer de thème musical à des repères bien précis. Dans les jeux, cela se traduit généralement par des changements de thèmes musicaux lorsqu'il se produit un changement majeur dans l'environnement du joueur. Par exemple, dans *Final Fantasy* [60], le joueur peut se déplacer sur un modèle réduit de la carte du monde mais il peut aussi visiter des temples, des villages, et bien d'autres endroits encore. De plus, au cours de ses explorations, il peut rencontrer des ennemis qu'il devra combattre. Pour modéliser ce cas avec un automate étendu probabiliste, il faut d'abord bien cerner le problème. On peut modéliser cela en définissant deux variables : une pour indiquer l'endroit où le joueur se trouve (V_1) et une autre pour indiquer s'il y a combat ou non.

À l'aide de certaines suppositions, généralement tirées du document de conception du jeu, un tableau comme le tableau 3.1 peut être directement traduit en automate par un

	Endroit (V_1)	Combat (V_2)
0	Carte du monde	Non
1	Temple	Oui
2	Village	—

TAB. 3.1 – Extrait des états possibles pour la musique de Final Fantasy

procédé de recouplement des variables. Pour *Final Fantasy*, on suppose qu'il n'est pas possible de se déplacer d'un endroit à l'autre lorsqu'un combat est en cours et qu'il ne peut y avoir de combat dans un village. La figure 3.6 illustre cette situation.

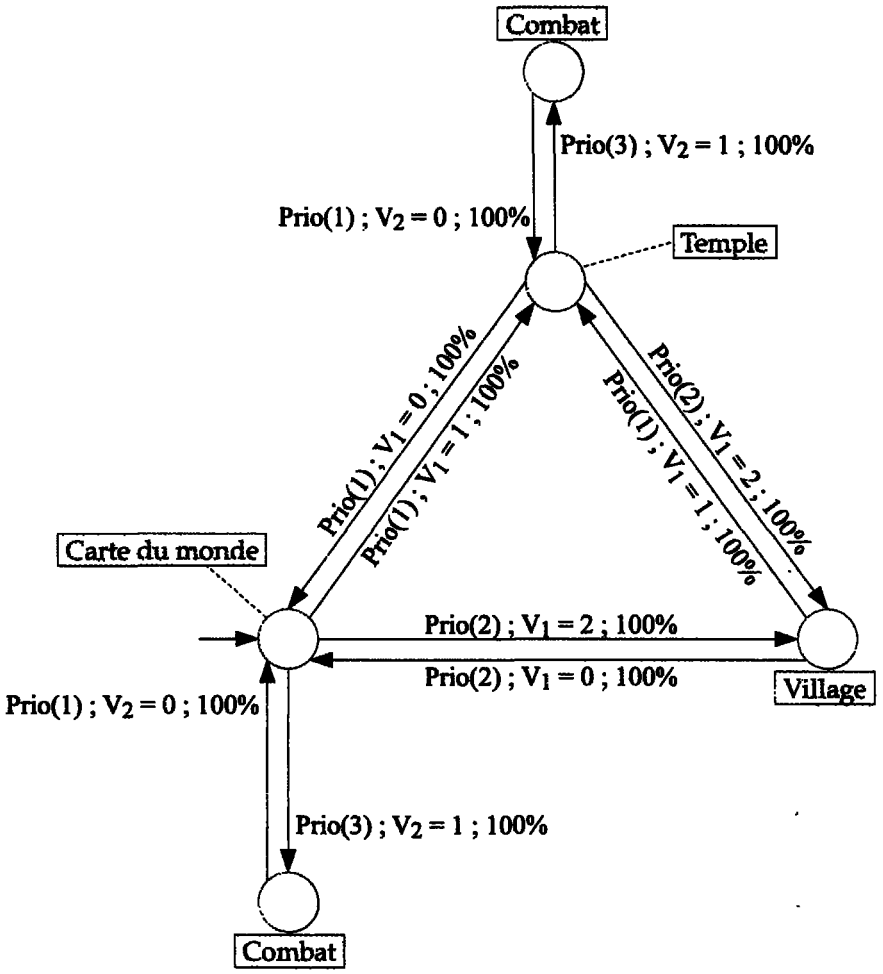


FIG. 3.6 – Changements de thèmes musicaux selon l'environnement du jeu

Exemple 3.3 Afin de combattre la fatigue de l'oreille causée par la répétition constante d'une même musique de fond, certains jeux introduisent des variations algorithmiques dans leur musique de fond. Par exemple, dans *Legend of Zelda : Ocarina of Time* [45], le thème de la « Prairie d'Hyrule » est composé de groupes de 8 mesures choisies aléatoirement parmi 12 possibilités. La figure 3.7 montre un automate étendu probabiliste effectuant la même tâche pour des groupes de 2 mesures choisies parmi 3 possibilités.

Bien entendu, ces trois exemples ne modélisent que des aspects isolés des trames sonores de ces trois jeux qui utilisent souvent une combinaison des techniques décrites et plus encore. Néanmoins, nous croyons que le fait que notre algorithme puisse modéliser une gamme aussi variée de techniques utilisées dans la création de trames sonores interactives et/ou algorithmiques pour les jeux vidéo suffit à prouver sa flexibilité.

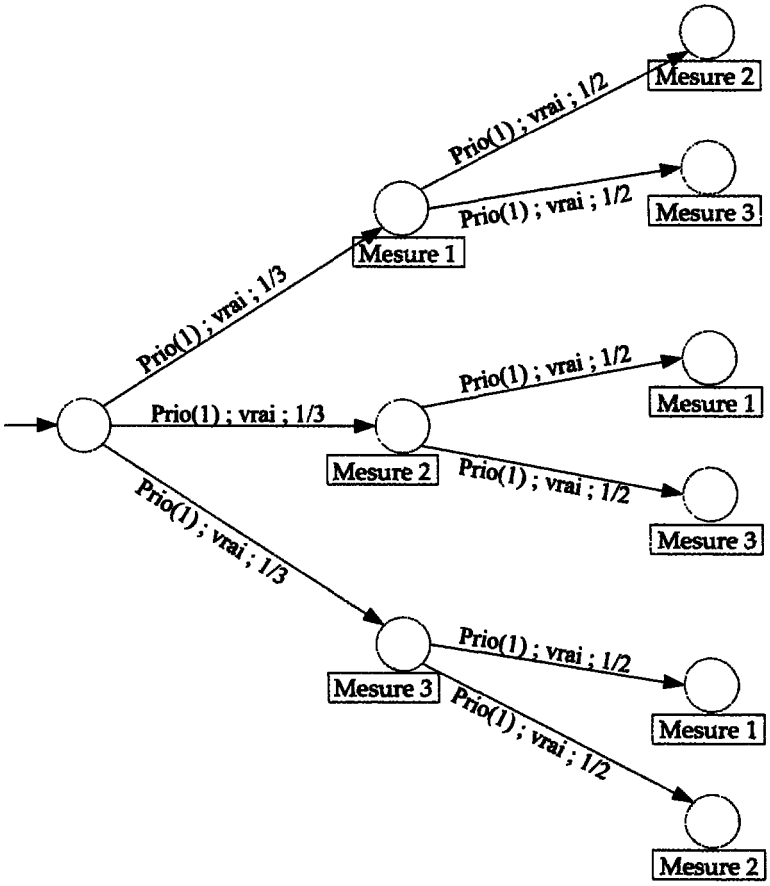


FIG. 3.7 – Permutation de mesures

Chapitre 4

Mise en œuvre

Dans ce chapitre, nous traiterons de la mise en œuvre de notre algorithme dans deux contextes applicatifs différents : un outil de composition et un jeu vidéo. Nous aborderons la question de la conception d'interfaces graphiques dans la première partie du chapitre puisqu'il s'agit d'un prérequis à la conception d'un outil de composition que nous avons nommé *IMTool*[1]. Puis, nous prendrons le point de vue d'un compositeur dont la tâche sera de paramétrer l'algorithme afin d'aborder l'outil de composition lui-même. Enfin, dans la dernière partie du chapitre, nous nous pencherons sur les spécificités reliées à la mise en œuvre de notre algorithme à l'intérieur d'un jeu vidéo. Dans cette dernière partie, nous tenterons aussi d'établir des comparatifs entre un jeu intégrant notre algorithme et un jeu de référence utilisant de la musique linéaire.

Définition 4.1 (*Cohérence*)
Selon *Van Dam et Foley* [17], la *cohérence* implique que tous les objets de l'interface utilisateur se comportent de manière similaire partout à travers l'interface et que les objets aux fonctionnalités similaires se ressemblent. De plus, l'interface doit être uniforme dans l'ensemble du système.

4.1 Les interfaces graphiques

Dans cette section, nous ferons une courte introduction à la conception d'interfaces graphiques. Plus particulièrement, nous nous pencherons sur différents critères à considérer lorsqu'on veut concevoir une interface permettant à l'utilisateur de performer. Par exemple,

[1] Le nom signifie « Interactive Music Tool ».

Foley et al. préconisent un critère de cohérence [17]. Toutefois, ces mêmes auteurs font aussi remarquer qu'il ne faut pas appliquer ce critère aveuglément au détriment de critères plus importants et ainsi risquer de surprendre l'utilisateur. Ceci concorde avec les remarques de *Kellogg* [33] et de *Grudin* [25] qui stipulent qu'une interface incohérente est parfois préférable car la cohérence est un concept inhéremment relatif et n'a pas de sens par elle-même en tant qu'objectif de conception d'interface [25,33]. On lui préfèrera généralement d'autres critères comme la simplicité d'utilisation, l'ergonomie, la rectification des erreurs, la clarté visuelle et l'esthétisme [17, 48, 65]. Certains auteurs proposent l'utilisation d'ensembles de critères totalement différents tels que la manipulation directe, la facilité de discrimination entre les différents éléments de l'interface et bien d'autres encore [14, 55, 56]. Par exemple, *Oren et Nayar* [48] suggèrent de prendre en compte vingt et un critères dont : l'apprentissage minimal, la mémorisation minimale, la simplicité, la familiarité, la séparation des préoccupations, la fonctionnalité, une relation restreinte avec les usagers, l'informativité, la perceptivité, l'habilité d'explication, l'expressivité, l'aspect esthétique et culturel, la fiabilité, la prédictibilité, la cohérence, la sécurité, le support assurance qualité intégré à l'interface, l'adaptabilité, la personnalisabilité, la maintenabilité et la portabilité. Après évaluation, nous avons retenu un ensemble de critères grâce auxquels nous avons conçu un outil de composition, *IMTool*, proposant une interface conviviale à l'utilisateur. Nous allons maintenant décrire plus en détails chacun des critères retenus.

Le premier critère que nous avons considéré est la *simplicité d'utilisation*, car une interface trop complexe à utiliser risque de décourager les utilisateurs potentiels par sa courbe d'apprentissage trop élevée. À l'opposé, une interface simple à utiliser ne nécessite que peu ou pas d'apprentissage de telle sorte qu'un utilisateur débutant peut immédiatement commencer à utiliser celle-ci et un utilisateur compétent peut rapidement se rappeler comment l'utiliser. Ce critère a guidé nos travaux de recherche, y compris l'algorithme de composition choisi.

Comme deuxième critère, nous avons retenu *l'ergonomie de l'interface*. Ce critère est important car une interface ergonomique accélère le travail de l'utilisateur en minimisant le nombre de mouvements répétitifs et de clics inutiles que doit effectuer l'utilisateur au cours de

sa session de travail. Ainsi donc, une interface faisant preuve d'une bonne ergonomie améliore aussi la simplicité d'utilisation d'un logiciel. Afin d'atteindre cet objectif dans *IMTool*, nous avons éliminé la majorité des boîtes de dialogues et rendu les outils de conception d'automates persistants (voir section 4.2).

Définition 4.2 (*Persistant*)
En conception d'interfaces graphiques, on dit qu'un outil est *persistant* s'il reste sélectionné après utilisation. Par constraste, un outil qui n'est pas *persistant* deviendra désélectionné après utilisation au profit d'un outil par défaut, quel qu'il soit.

Nous venons de voir qu'une interface simple et ergonomique permet une utilisation rapide et efficace d'un outil logiciel. Toutefois, l'utilisateur n'est pas infaillible et peut parfois commettre des erreurs dont certaines peuvent avoir des conséquences fâcheuses. Par exemple, dans *IMTool*, la suppression d'un état de l'automate entraîne la suppression de toutes les transitions entrantes et/ou sortantes de cet état. Il est donc important de permettre à l'utilisateur de *rectifier ses erreurs* en revenant en arrière sur ses dernières actions. C'est pourquoi nous avons implémenté un mécanisme robuste d'annulation des opérations.

Le critère suivant qui a orienté la conception de l'interface est la *clarté visuelle* car une interface visuellement claire permet à l'utilisateur de rapidement distinguer les différentes composantes de celle-ci. Il est possible de concevoir une interface claire en éliminant les éléments inutiles de l'interface, en groupant de façon logique les éléments restants et en s'assurant qu'il y ait un espacement suffisant entre les différents groupes [17].

Le critère suivant que nous avons pris en compte est la *rétroaction visuelle* car une bonne interface ne doit jamais forcer l'utilisateur à deviner l'état du programme. Dans *IMTool*, la rétroaction visuelle passe par des changements de couleurs lorsque l'utilisateur sélectionne différents éléments de l'automate ou lorsque le logiciel joue une séquence MIDI associée à un état.

Les deux critères précédents nous amènent à considérer *l'esthétisme* de l'interface. En effet, une étude par Tractinsky [65] a démontré que les interfaces esthétiquement plaisantes produisent une meilleure impression d'utilisabilité apparente au premier coup d'œil, ce qui

affecte aussi l'impression à long terme des utilisateurs sur l'utilisabilité réelle du système. Il est donc important de faire en sorte que l'interface soit la plus plaisante possible du point de vue esthétique. Dans *IMTool*, l'atteinte de cet objectif passe principalement par le choix des couleurs mais aussi par l'organisation de la fenêtre principale, ce qui fait aussi partie du critère de clarté visuelle.

Enfin, le dernier critère que nous avons considéré est la *manipulation directe*. Ceci signifie que l'utilisateur doit pouvoir manipuler directement l'objet d'intérêt [55, 56]. Toutefois, dans notre outil, ce critère n'est que partiellement respecté car la représentation par automates n'est qu'une représentation intermédiaire de la musique interactive. En effet, bien que nous y référions parfois en tant qu'outil de composition, rappelons-nous que notre outil demeure principalement un outil de paramétrage d'un algorithme de composition. Pour pallier à ce problème, nous avons implémenté dans *IMTool* des fonctions de prévisualisation et de déboguage du résultat telles que « Play », « Stop », « Pause », « Step » et « Play State ».

Nous venons de voir différents critères pour la conception d'interfaces. Nous nous sommes bien sûr limités aux critères que nous avons utilisés pour concevoir l'interface de notre outil. Dans les sections qui vont suivre, nous décrirons celui-ci en plus de détails et expliquerons le processus de conception d'une musique interactive à l'aide de cet outil.

4.2 IMTool

Au dernier chapitre, nous avons présenté notre implémentation d'un moteur de musique interactive basé sur les automates étendus probabilistes. Il s'agissait en fait de la première composante d'un système de musique interactive qui ne pourrait être complet sans un outil de composition. En effet, sans outil de composition, les musiciens auraient beaucoup de difficulté à créer du contenu pour un tel engin sans solliciter l'aide de programmeurs qui, vu l'envergure des jeux vidéo modernes, sont souvent déjà débordés. C'est pourquoi nous avons implémenté *IMTool* qui utilise le moteur décrit au précédent chapitre et qui est aussi utilisé, sous une autre forme, dans un jeu vidéo que nous avons implémenté. Nous reviendrons sur l'interface entre le jeu et le moteur à la section 4.3.3.

Le logiciel *IMTool* représente des musiques interactives sous la forme d'automates finis étendus probabilistes. Il permet de manipuler directement ces automates à l'aide d'outils de création d'automates et d'un panneau affichant les propriétés des états et des transitions sélectionnées. À l'aide d'un autre panneau, il permet aussi d'importer des séquences MIDI qui serviront d'alphabet de sortie aux automates ainsi que des Soundfonts qui serviront de banques d'instruments pendant l'étape de rendu des séquences MIDI. À l'exception de la fonctionnalité de rendu MIDI pour laquelle il utilise des bibliothèques provenant de tierces parties, le logiciel a été développé entièrement à partir de zéro.Pour effectuer le rendu des séquences MIDI, nous avons utilisé le synthétiseur *Fluidsynth*[2] [26] et la bibliothèque d'ordonnancement en temps réel *MidiShare*[3] [22], deux bibliothèques « Open Source » sous license « GNU Lesser General Public License v2 » [18].

La figure 4.1 montre la fenêtre principale de l'application. On y retrouve les principaux éléments de l'interface : une barre de menus, une barre d'outils, une barre d'état ainsi que les trois panneaux mentionnés précédemment. Avant de passer à une description plus détaillée de chacun des éléments de l'interface, remarquons les deux caractéristiques suivantes de celle-ci :

1. Chaque panneau est identifié par sa fonction.
2. Les boutons de la barre d'outils sont larges, colorés, espacés et également bien identifiés.

Grâce à ces particularités de l'interface, l'utilisateur peut rapidement se rappeler la fonction de chaque élément de l'interface puisqu'elle est constamment affichée à l'écran. De surcroît, il peut facilement repérer les différents boutons sur la barre d'outils grâce à leurs couleurs vives. Ces deux caractéristiques de l'interface augmentent non seulement la simplicité d'utilisation du logiciel mais aussi la clarté visuelle de l'interface.

Par convention, le menu principal de l'application contient les fonctions d'accès aux fichiers et les fonctions d'aide du logiciel. Il contient aussi les fonctions moins souvent effectuées à l'aide de la souris telles que les fonctions d'édition. Toutefois, les fonctions de création et de déboguage d'automates sont absentes des menus puisqu'elles sont constamment utilisées

[2] `http://www.nongnu.org/fluid/`
[3] `http://midishare.sourceforge.net/`

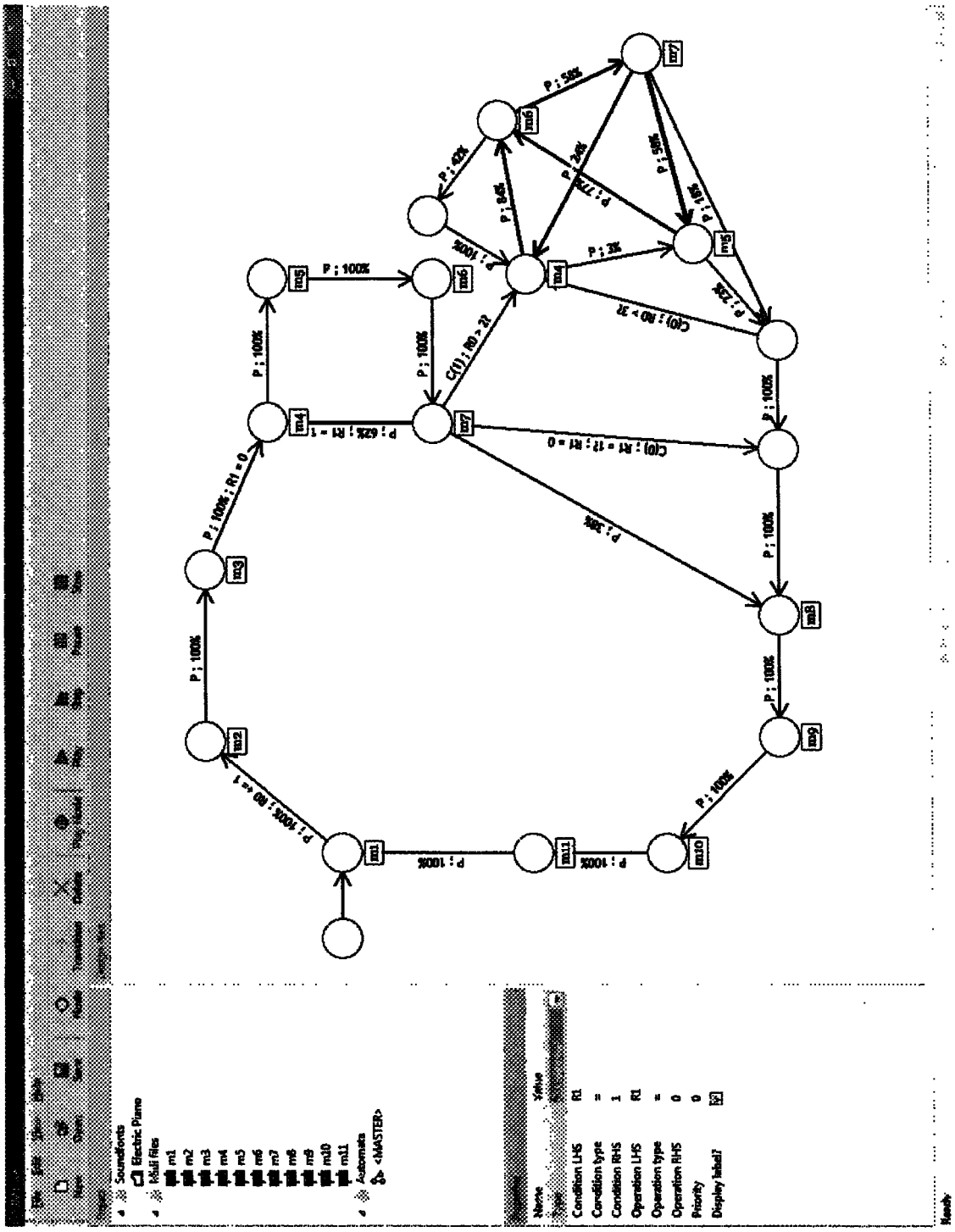


FIG. 4.1 – L'interface utilisateur

et déjà présentes dans la barre d'outils. De ce fait, ces fonctions ne feraient qu'encombrer les menus, ce qui diminuerait la clarté visuelle de l'application.

La barre d'outils est divisée en trois groupes d'outils. Le premier contient les opérations d'accès aux fichiers les plus fréquemment utilisées, soit *nouveau*, *ouvrir* et *enregistrer*. Le deuxième groupe se constitue des outils de création d'automates. Ceux-ci permettent de créer et de supprimer des états et des transitions dans la zone de conception. Ces outils sont persistants, c'est-à-dire qu'ils provoquent un changement de mode du logiciel. Nous verrons plus loin, lorsque nous traiterons de la zone de conception, ce que cela implique. Enfin, le dernier groupe contient les outils de prévisualisation et de déboguage qui permettent à l'utilisateur de corriger les problèmes dans sa musique avant qu'ils ne surviennent dans le jeu vidéo. Malheureusement, cette prévisualisation n'est fiable à 100% que si l'on suppose que le code du jeu vidéo ne contient pas de défauts majeurs au niveau du code chargé de mettre les registres du moteur de musique à jour. Par exemple, il peut arriver que le jeu ne fasse jamais la mise à jour de certains registres ou qu'il écrive des valeurs erronées dans les registres. C'est le programmeur du jeu, soit l'utilisateur de l'API d'*IMTool*, qui doit s'assurer de la cohérence à ce niveau.

La barre d'état permet à l'utilisateur de voir plus rapidement la condition d'activation d'une transition lorsqu'il n'y a pas assez d'espace dans la zone de conception pour afficher celle-ci au dessus de la flèche représentant la transition. Cela évite à l'utilisateur de devoir reconstituer mentalement la condition d'activation courante à partir de la fenêtre de propriétés. Cela lui évite aussi de devoir mémoriser celle-ci s'il prévoit la modifier.

Le panneau en bas à gauche sur la figure 4.1 représente la fenêtre de propriétés que nous venons de mentionner. Il s'agit d'une liste clé-valeur qui se met à jour automatiquement dès que l'utilisateur sélectionne un élément (par un clic du bouton gauche) dans l'explorateur de projet ou dans la zone de conception. Comme cette liste est toujours visible, l'utilisateur n'a pas à effectuer d'actions supplémentaires pour accéder aux propriétés d'un objet. Cela diminue le nombre de clics inutiles et répétitifs, ce qui fait partie du critère d'ergonomie de notre application. Grâce à cette liste, l'utilisateur peut modifier les propriétés de l'élément sélectionné à sa guise. Cet élément est mis à jour lorsqu'un autre objet est sélectionné ou lorsque le projet

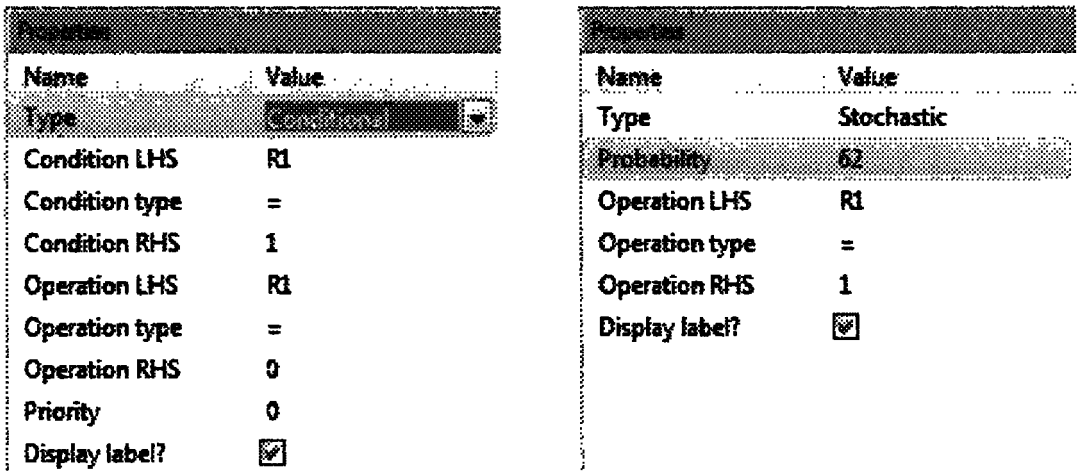
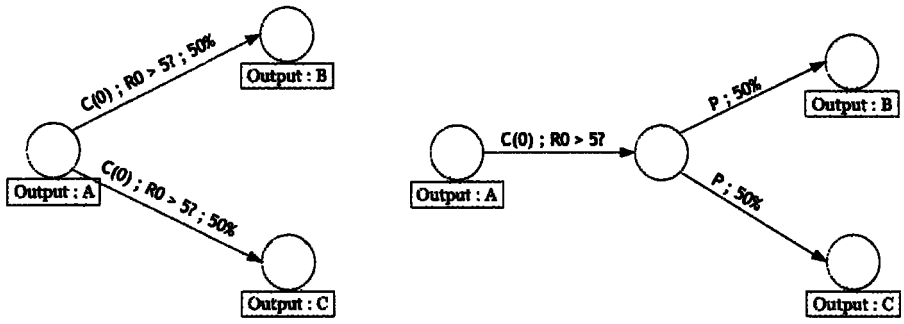


FIG. 4.2 – Fenêtre de propriétés

est sauvegardé. La figure 4.2 montre le panneau affichant les propriétés de deux transitions différentes. On remarque que la plupart des propriétés correspondent à celles que nous avons définies au chapitre précédent, bien que les noms utilisés aient été changés dans le but de rendre l'interface utilisateur plus conviviale. Par contre, la propriété *type* est spécifique à l'éditeur et a été introduite dans un but de prévention d'erreurs. En effet, le modèle d'automates finis étendus probabilistes est complexe et peut sembler difficile à aborder pour un utilisateur néophyte. Nous avons donc divisé les transitions en deux types : les transitions conditionnelles et les transitions stochastiques. Dans le cas des premières, nous fixons la probabilité de transition à 100% mais nous laissons l'utilisateur modifier librement la condition de transition ainsi que la priorité de transition. Pour les deuxièmes, nous faisons le contraire : nous fixons la condition de



(a) Sans « types »

(b) Avec « types »

FIG. 4.3 – La propriété « type » ne limite pas l'expressivité du modèle

transition à la tautologie booléenne VRAI et la priorité de transition à $-\infty$ mais nous laissons l'utilisateur modifier librement la probabilité de transition. On peut voir, en comparant les figures 4.3(a) et 4.3(b), que cette contrainte ne restreint pas l'expressivité du modèle.

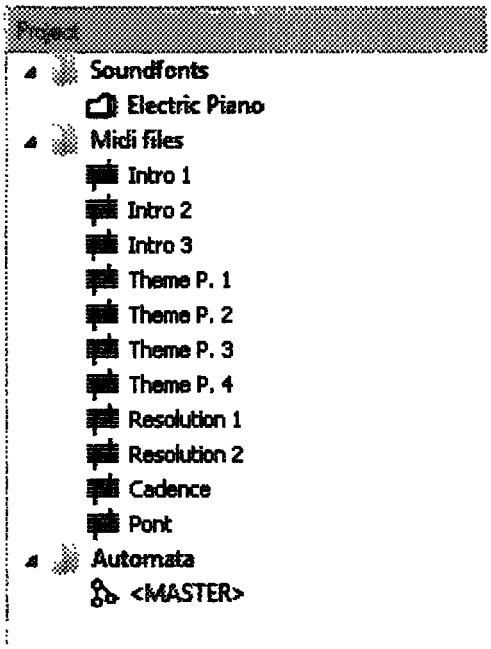


FIG. 4.4 – Explorateur de projet

La figure 4.4 illustre en gros plan l'explorateur de projet. On peut voir qu'il s'agit d'une liste des objets (fichiers) faisant partie du projet et que ces objets sont classés dans trois dossiers. Ces trois dossiers représentent les trois types d'objets qu'un projet peut contenir : des Soundfonts, des séquences MIDI et des automates[4]. L'ajout de fichiers dans le projet peut se faire par le menu principal de l'application ou directement dans ce panneau par un clic du bouton droit de la souris. Dans ce cas, un menu contextuel apparaît et propose non seulement l'ajout de fichiers mais aussi un ensemble de tâches relatives au point de chute du clic de souris. Il est par exemple possible d'écouter une séquence MIDI individuelle à partir de l'explorateur de projet. Quant au bouton gauche de la souris, il permet de sélectionner les objets déjà présents dans le projet et ainsi faire afficher leurs propriétés. Comme propriétés,

[4]Rappelons toutefois que les projets créés avec la version présente du logiciel ne peuvent pas contenir d'autres automates que l'automate maître(<MASTER> sur la figure).

les Soundfonts et les séquences MIDI possèdent tous deux un nom d'affichage et le chemin sur disque du fichier. Toutefois, les Soundfonts possèdent une propriété supplémentaire : un décalage de banque. Cette propriété permet d'utiliser plusieurs Soundfonts dans un même projet car la plupart des Soundfonts ont été conçues pour se charger dans la banque d'instruments 0, ce qui provoque un conflit lorsqu'on tente d'en utiliser plusieurs à la fois. Le synthétiseur que nous utilisons, *Fluidsynth*, propose donc de rémédier à ce problème en spécifiant un décalage de banque (« bank offset » en anglais) pour chaque Soundfont. Ce décalage est ajouté au numéro de banque de chaque instrument dans la Soundfont. Par exemple, si la Soundfont spécifie qu'un instrument devrait se charger dans la banque 4 et qu'on spécifie un décalage d'une position, *Fluidsynth* charge cet instrument dans la banque 5. Ceci amène toutefois le problème qu'il faut informer les fichiers MIDI que les Soundfonts ont été décalées. Ceci se fait grâce au message de changement de banque (voir annexe A).

La zone de conception (panneau de droite sur la figure 4.1) permet à l'utilisateur de manipuler directement un automate. À l'aide des outils précédemment mentionnés, il peut créer et supprimer des états et des transitions en cliquant dans ce panneau. Comme nous l'avons aussi déjà mentionné, ces outils sont persistants, c'est-à-dire que lorsque l'utilisateur clique sur le bouton correspondant dans la barre d'outils, le logiciel passe du mode *sélection* à l'un des modes suivant : *création états*, *création transitions*, *suppression éléments*. Il restera dans ce mode tant que l'utilisateur ne cliquera pas sur le bouton droit de la souris pour revenir au mode *sélection*. Cette mesure d'ergonomie permet à l'utilisateur de travailler plus rapidement par rapport à une première version du logiciel qui n'implémentait pas cette fonctionnalité. Ce panneau implémente aussi la majorité de la rétroaction visuelle de notre logiciel. En effet, lorsque l'utilisateur clique sur un élément de l'automate celui-ci change de couleur en même temps que les propriétés de celui-ci s'affichent dans la fenêtre de propriétés. De plus, lorsque le logiciel est en mode prévisualisation et que l'automate traverse un état auquel est associé une séquence MIDI, il colore cet état en rouge dans la zone de conception pendant que la séquence MIDI joue.

Enfin, avant d'expliquer le processus de composition d'une musique interactive, nous allons d'abord parler du mode de prévisualisation et de déboguage du logiciel, illustré à la

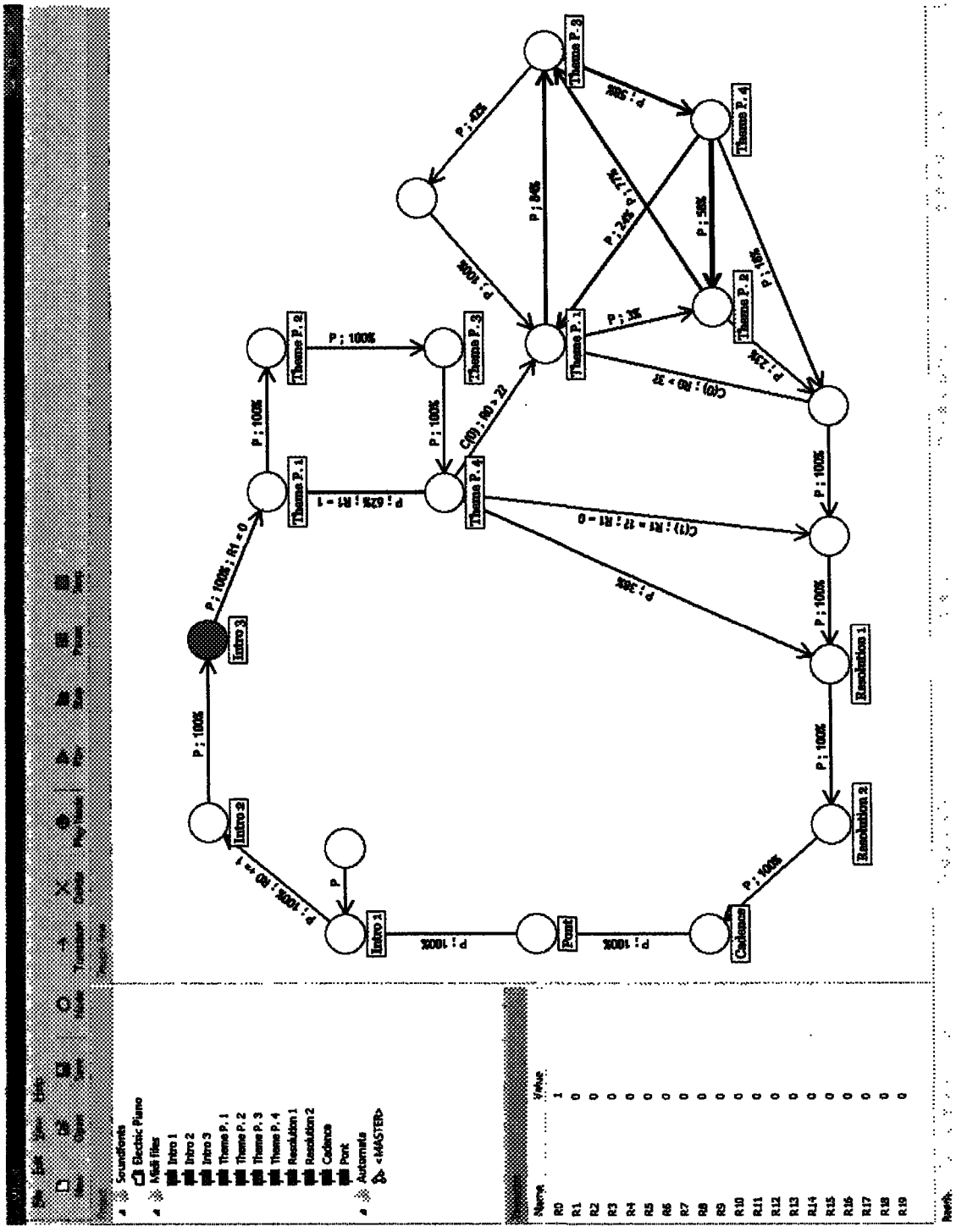


FIG. 4.5 – Le logiciel en mode prévisualisation/déboguage

figure 4.5. On remarque sur cette figure qu'une fenêtre affichant le contenu des registres a pris la place de la fenêtre de propriétés. Il s'agit en fait d'un débogueur qui permet de modifier le contenu des registres lorsque l'utilisateur suspend la simulation. Ces modifications seront prises en compte lors de la reprise de la simulation, au moment où le programme réinitialisera les registres avec les valeurs apparaissant dans la fenêtre de déboguage.

4.3 Mise en œuvre dans un jeu vidéo

Afin de tester notre outil dans un scénario d'utilisation typique, nous avons conçu un prototype de jeu vidéo très simple et composé la trame sonore pour celui-ci à l'aide d'*IMTool*. Cette section portera sur les différentes étapes de cet exercice et sera donc divisée en trois parties : la conception du prototype de jeu, la composition de la trame sonore et l'intégration de la trame sonore dans celui-ci.

4.3.1 Conception d'un prototype de jeu

La première étape de la conception d'un jeu est la production d'un *document de concept*. Dépendamment de la complexité du jeu, la taille de ce document peut varier entre un paragraphe et plusieurs dizaines de pages. Nous nous sommes limités à un seul paragraphe puisqu'il ne s'agissait que d'un prototype de jeu très simple destiné principalement à tester notre algorithme de musique interactive et à valider l'API du moteur.

> L'objectif du jeu est de protéger une cible attaquée par des vagues d'ennemis successives. Le joueur peut éliminer ces ennemis en *cliquant* dessus avec le stylet de la Nintendo DS. Lorsque tous les ennemis d'une même vague ont été éliminés, une nouvelle vague d'ennemis plus agressifs est générée.

Puisque dans ce concept de jeu, le stylet n'est utilisé que de façon superficielle et aurait parfaitement pû être remplacé par la souris d'un ordinateur conventionnel, il est nécessaire ici d'expliquer le choix d'utiliser la Nintendo DS comme plateforme d'implémentation. En effet, rappelons-nous qu'au chapitre 3, nous avons affirmé que notre algorithme était très efficace et ainsi mieux adapté que plusieurs autres pour l'utilisation en temps réel dans un jeu vidéo. Le fait d'utiliser la Nintendo DS nous permettra de justifier cette affirmation puisqu'il s'agit

d'une console ne disposant que de peu de puissance et qu'elle doit partager celle-ci entre tous les sous-systèmes d'un même jeu (graphiques, intelligence artificielle, musique, effets sonores, etc) tout en maintenant une vitesse respectable pour celui-ci. Idéalement, un jeu sur Nintendo DS devrait fonctionner à une vitesse de 60 images (« frames » en anglais) par seconde, soit la vitesse de rafraîchissement de l'écran de la console. En pratique, on peut être un peu plus tolérant.

4.3.2 Composition de la trame sonore

Revenons au document de concept du jeu. Sur celui-ci, aucune direction musicale n'est spécifiée pour le jeu. Vu la nature itérative de ces documents, cette situation est parfaitement normale. Toutefois, elle nous empêche de procéder à la composition de la trame sonore. Il nous faut donc y remédier :

> Initialement, pendant la première vague d'ennemis, la musique sera constituée d'une boucle musicale très simple. La musique change lorsqu'une vague d'ennemis est complètement détruite. Dès lors, on peut soit ajouter une couche supplémentaire à la musique en cours, soit complexifier celle-ci en y ajoutant des notes, des transitions aléatoires, ou en changeant carrément sa structure. On peut aussi combiner les méthodes précédentes pour produire un résultat complètement chaotique.

Ainsi donc, on peut passer à la première étape de la composition de la trame sonore qui consiste en la composition d'une ou plusieurs séquences MIDI initiales. Ces séquences initiales pourront faire partie ou non des séquences qui seront utilisée dans l'œuvre finale[5]. Ces séquences pourront ensuite être importées dans *IMTool* et nous permettront d'immédiatement commencer à utiliser *IMTool* pour spécifier la *méta-structure*[6] de notre composition ainsi que ses interactions avec notre prototype de jeu.

Après avoir importé les séquences MIDI dans *IMTool*, il nous faut maintenant choisir des Soundfonts pour effectuer le rendu de celles-ci. Nous avons choisi d'utiliser une Soundfont fournissant l'ensemble complet des instruments spécifiés par le standard General

[5]Souvent, les séquences utilisées dans l'œuvre finale sont produites en parallèle avec la spécification de l'automate et dérivent des séquences initiales.

[6]On retrouve souvent plusieurs niveaux hiérarchiques de structure dans les pièces musicales. Puisque chaque séquence MIDI possède sa propre structure interne et que l'automate décrit par *IMTool* est une structure d'ordre supérieur qui s'y superpose, on qualifie celui-ci de méta-structure.

MIDI : « Airfont 380 ». Cette Soundfont est disponible gratuitement sur Internet pour usage non commercial.

Ensuite, nous avons défini une correspondance entre les registres de l'automate et les interactions souhaitées entre la musique et l'état du jeu. Selon le document de concept, la seule interaction nécessaire consiste à modifier la musique lorsqu'une vague d'ennemis est éliminée. Nous avons donc choisi de faire correspondre le registre R_0 au nombre d'ennemis restants. De cette façon, la condition $R_0 = 0$ déclenche le changement de structure musicale souhaité à chaque nouvelle vague d'ennemis.

Enfin, nous avons créé les états et les transitions, toujours en suivant le document de concept que nous avons défini. La figure 4.6 illustre le résultat final. Notez qu'à des fins de clarté nous avons supprimé quelques transitions.

4.3.3 Intégration de la trame sonore

La trame sonore en main, il ne reste qu'à l'intégrer au prototype de jeu vidéo précédemment développé. À cet effet, nous avons développé une version optimisée du moteur d'*IMTool* éliminant les calculs inutiles, tel que le rendu à partir du format MIDI vers le format WAV, et les informations redondantes, telles que celles énumérées ci-dessous :

1. Le numéro d'identification des états
2. Le numéro d'identification de l'état initial
3. Les indicateurs d'états finaux
4. L'état de départ sur les transitions
5. La valeur de priorité sur les transitions

Nous considérons que ces informations sont redondantes puisqu'il est possible d'obtenir le même résultat en prétraitant l'automate dans un exporteur de la façon suivante :

1. Réassignation des numéros d'identification des états ; si l'automate possède N états, on leur assigne les numéros $0 \ldots N-1$.
2. Assignation du numéro d'identification 0 à l'état initial de l'automate.

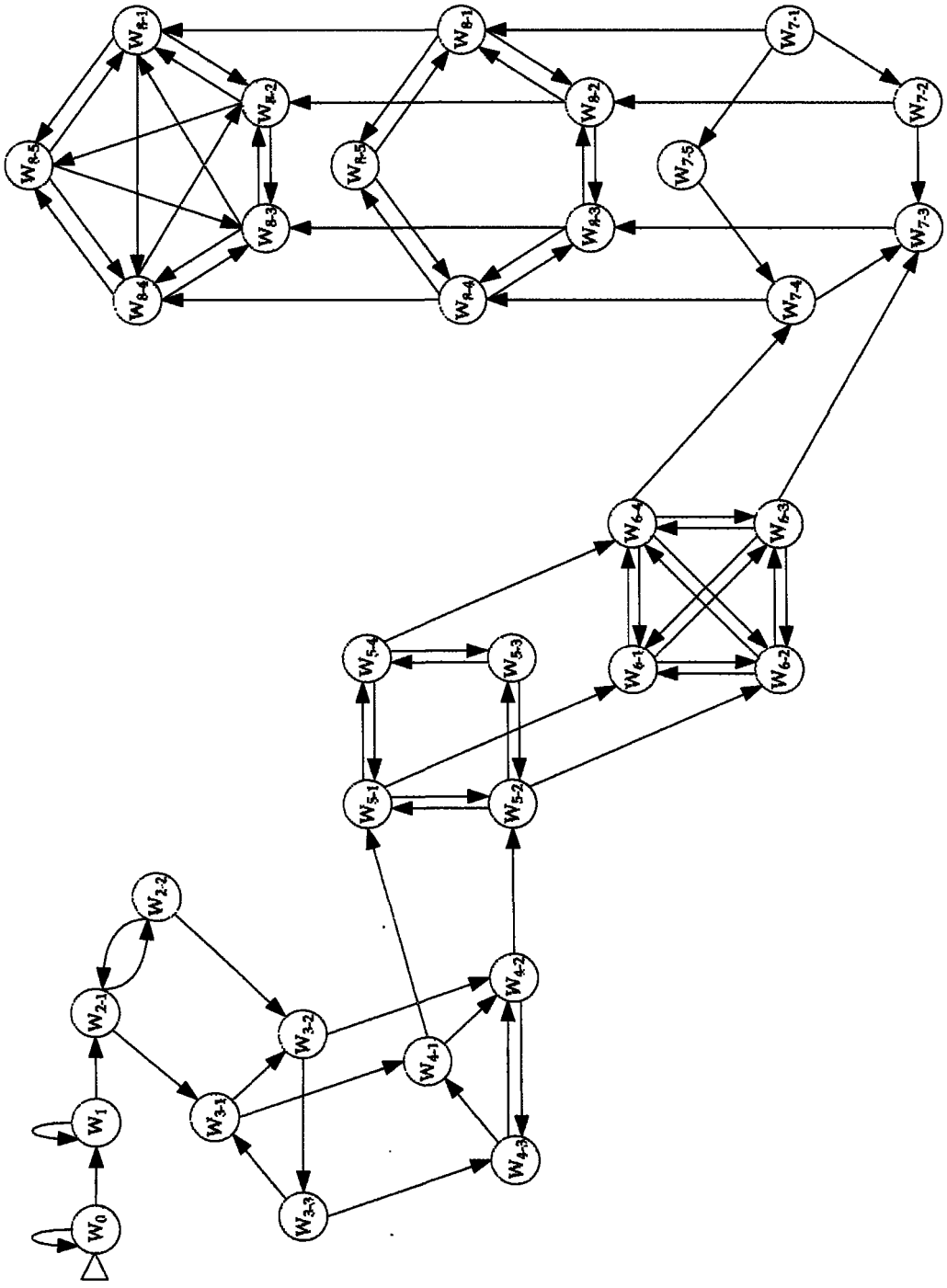


FIG. 4.6 – Automate résultant

3. Suppression des transitions inactivables (telles que définies à la section 3.2.1) incluant les transitions sortant des états finaux.
4. Fusion des tableaux d'états et de transitions pour produire un intercalage. La structure de donnée finale peut être visualisée à la figure 4.7.
5. Tri préalable des transitions par priorité.

En résumé, les différences majeures entre le moteur utilisé dans *IMTool* et celui utilisé dans notre prototype de jeu sont que ce dernier utilise directement le format WAV plutôt que le format MIDI, ce qui permet de l'utiliser sur des ordinateurs disposant de beaucoup moins de puissance de calcul qu'un PC moderne, par exemple la Nintendo DS. C'est grâce à *Fluidsynth* que cela a été rendu possible car sans celui-ci nous n'aurions pas pu convertir les fichiers MIDI en fichiers WAV. De plus, le moteur utilisé dans notre jeu utilise des structures de données optimisées occupant moins d'espace sur le disque et, de ce fait, permettant un chargement plus rapide des fichiers de musique interactive. Des essais ont démontré que ce nouveau format occupe jusqu'à 90% moins d'espace sur disque que le format original, basé sur XML, se charge en mémoire jusqu'à 32 fois plus rapidement sur la Nintendo DS[7].

Nombre d'états
1er pointeur vers les états
...
Nombre de conditions dans le 1er état
1er pointeur vers les conditions dans le 1er état
...
Nb. de destinations de la 1^{ere} condition du 1er état
1^{ere} destination[8] de la 1^{ere} condition du 1er état
...
...

FIG. 4.7 – Structure de données finale

[7]Les résultats exacts sont 76 secondes pour le chargement de vingt mille fichiers dans le format original et 2.35 secondes pour le chargement de vingt mille fichiers dans le format optimisé. Ces résultats sont reproductibles.

[8]La description d'une destination comprend sa probabilité, son état d'arrivée et sa fonction de mise-à-jour.

Un autre avantage de ce format est qu'il représente beaucoup mieux notre modèle tel que nous l'avons décrit au chapitre 3. Ainsi donc, dans ce format, chaque condition représente une fonction d'activation $c(X) \in C$ et la liste de destinations sert à réduire l'espace utilisé par les vecteurs stochastiques (puisque la majorité des probabilités de transitions sont égales à 0).

Avant de conclure, il convient de décrire l'interface entre le moteur de musique et la boucle principale du jeu. Celle-ci est composée de six fonctions : `Reset`, `LoadMusic`, `SetRegister`, `GetRegister`, `StartMusic`, et `StopMusic`. Leur rôle est le suivant :

`Reset` (ré)initialise le moteur de musique en remettant tous les registres à zéro.

`LoadMusic` charge un fichier contenant un automate étendu probabiliste ainsi que tous les segments musicaux nécessaires.

`SetRegister` modifie le contenu d'un registre.

`GetRegister` retourne le contenu d'un registre.

`StartMusic` démarre le moteur de musique.

`StopMusic` arrête le moteur de musique.

4.4 Discussion

Après avoir testé le jeu résultant, il nous apparaît clair que notre algorithme est assez efficace pour être utilisé en temps réel dans un jeu vidéo, même lorsque celui-ci est programmé sur une console disposant d'une puissance limitée. De plus, nous avons trouvé que l'interface d'*IMTool* est possède pour permettre, avec une quantité d'effort raisonnable, la création d'automates étendus probabilistes complexes (comme celui sur la figure 4.6). Ces résultats corroborent nos affirmations du chapitre 3.

Toutefois, il nous apparaît aussi que l'interface d'*IMTool* présente une limitation importante que nous illustrerons à l'aide d'un exemple : notre composition présentée à la figure 4.6. L'automate sur cette figure est formé d'un très grand nombre d'états et de transitions. Nous avons donc dû supprimer certaines transitions afin améliorer la clarté visuelle. Nous proposons ici deux solutions à cette limitation :

- L'introduction d'une représentation hiérarchique pour les automates étendus probabilistes où chaque état de l'automate aurait la possibilité de correspondre à l'état de départ d'un sous-automate.
- L'utilisation de flèches bidirectionnelles ($\leftrightarrow$) lorsque la réciproque $r \rightarrow q$ d'une transition $q \rightarrow r$ existe.

Remarquons au passage qu'il n'est toutefois pas nécessaire d'apporter de changements au modèle tel que nous l'avons présenté au chapitre 3. En effet, il est possible d'aplatir un automate représenté de façon hiérarchique en remplaçant de façon récursive les états qui correspondent à des sous-automates (*méta-états*) par la description de ces derniers. Ne reste qu'à déterminer s'il faut recopier les transitions sortant des méta-états sur tous les états de leurs sous-automates ou s'il ne faut évaluer ces transitions que lorsqu'un sous-automate atteint un état final. Nous proposons de combiner les deux solutions en appelant le premier type de transitions : « transitions prioritaires » et le deuxième type de transitions : « transitions normales ».

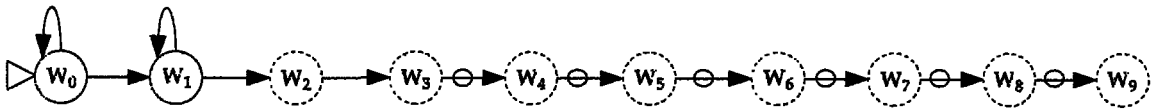


FIG. 4.8 – Structure globale de la pièce utilisée pour notre prototype

En appliquant la première solution, la figure 4.6 pourrait être rescindée en structure hiérarchique à deux niveaux. Le premier représenterait la structure globale de la pièce et pourrait être représenté comme sur la figure 4.8. Sur cette figure, les méta-états sont représentés par des traits pointillés et les transitions hautement prioritaires sont marquées d'un cercle. Chaque méta-état serait associé à un sous-automate décrivant sa composante respective. Ainsi le méta-état W_9 serait associé à un automate contenant les anciens états W_{9-1}, W_{9-2}, W_{9-3}, W_{9-4} et W_{9-5}. En appliquant la seconde solution, ce dernier pourrait être représenté comme sur la figure 4.9.

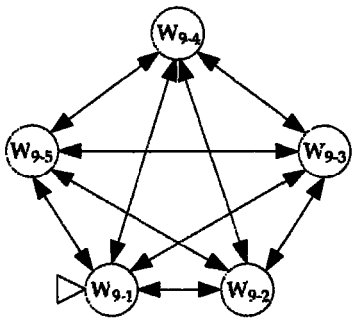


FIG. 4.9 – Utilisation de flèches bidirectionnelles pour améliorer la clarté visuelle

Conclusion

Ce travail a porté sur l'étude de différentes structures mathématiques et leurs applications possibles dans un cadre de génération et de composition de musique. Ceci avait pour but de trouver des pistes de solution au problème de la composition de musique pour les médias interactifs qui, à l'inverse des médias linéaires, n'ont pas de durée fixe ni d'ordre pré-déterminé pour leurs scènes. Cette propriété des médias interactifs, dont les jeux vidéo font partie, fait qu'il est très difficile de juger de la quantité de musique qu'il sera nécessaire de composer pour une production donnée car il faut en fait établir un juste milieu entre diversité et répétition. Ceci permet de ne pas fatiguer l'oreille du sujet avec de la musique trop répétitive mais laisse quand même la possibilité au compositeur d'établir des thèmes forts et mémorables qui permettront à l'auditeur d'identifier leur provenance sans trop de difficulté. Bien entendu, différents types de médias interactifs auront différentes priorités. Par exemple, un guide touristique interactif favorisera plus la diversité que la répétition. À l'inverse, les médias ludiques tels que les jeux vidéo favoriseront plus la répétition que la diversité afin de mieux plonger le joueur dans l'univers du concepteur du jeu. Puisque ces deux points de vue sont difficilement réconciliables, nous avons choisi de ne nous attarder qu'à ce dernier cas pour ce mémoire. Ceci s'est avéré la bonne décision car ce travail a donné lieu à une publication dans le cadre d'une conférence internationale sur les jeux vidéo [8].

Nous avons tout d'abord étudié plusieurs classes d'algorithmes ayant été appliqués à la génération de musique. Cela nous a permis de nous familiariser avec les différentes méthodes

ayant été utilisées par le passé pour générer de la musique algorithmiquement. Ces techniques peuvent ou non inclure l'utilisateur dans la boucle de génération. Entre autres, nous avons couvert différentes méthodes stochastiques dont les dés musicaux, les chaînes de Markov et les automates probabilistes car le hasard engendre la diversité. Nous avons aussi couvert un type particulier de grammaires, les L-systèmes, à cause de leur lien particulier avec les structures fractales. Ensuite, nous avons couvert les algorithmes génétiques car la recherche d'une solution musicale optimale est une technique importante en composition algorithmique. Enfin, nous avons couvert différentes classes d'automates, allant des automates cellulaires aux automates étendus, en passant par les automates finis et les automates probabilistes. Lors de cette étude, nous avons déterminé qu'aucun des algorithmes étudiés ne correspondaient parfaitement à nos critères. Toutefois, deux des modèles étudiés, les automates étendus et les algorithmes probabilistes, s'approchaient de très près de tous nos critères. Ceci nous a guidé dans la conception d'un nouveau modèle : les automates étendus probabilistes.

Nous avons ensuite étudié différents outils de composition afin de voir s'ils nous permettraient de mettre en application notre nouvel algorithme. Et, en cas de résultats positifs, s'ils seraient assez faciles à utiliser pour répondre à notre critère de facilité d'utilisation. Malheureusement, ceux que nous avons trouvé qui étaient assez flexibles pour qu'on puisse les modifier afin d'y intégrer notre algorithme présentaient généralement une courbe d'apprentissage trop élevée pour répondre à notre critère. À l'inverse, ceux qui étaient faciles à utiliser n'étaient pas modifiables. Ceci nous a incité à concevoir notre propre outil de composition afin de tester notre algorithme.

Nous avons ensuite introduit les automates étendus probabilistes. Comme leur nom l'indique, il s'agit d'une combinaison entre les automates étendus et les automates probabilistes, chacun venant complémenter les faiblesses de l'autre dans le cadre de la composition de musique. Leur fonctionnement est expliqué en détails au chapitre 3.

D'abord implémentés dans le logiciel *IMTool*, les automates étendus probabilistes ont été testés avec succès à l'intérieur d'un environnement de composition de musique. Ce logiciel permet à son utilisateur d'importer des séquences MIDI représentant de courts motifs

musicaux puis de chaîner ceux-ci dans un transducteur étendu probabiliste où chaque état est associé soit à une des séquences importées, soit à la séquence nulle. En ce sens, les transitions représentent un ensemble de règles et de probabilités régissant la progression de la musique du jeu. La tâche incombe au compositeur de définir la portée de ces règles de transition.

Afin de tester sa performance, nous avons ensuite implémenté notre modèle d'automates étendus probabilistes dans un moteur de jeu vidéo. Ces essais se sont avérés concluants en ce sens que l'utilisation de notre modèle a permis de composer de la musique qui établit un thème que le joueur peut reconnaître tout au long du jeu tout en introduisant de la diversité dans celui-ci. De plus, le modèle est assez simple pour pouvoir s'implémenter de manière efficace de façon à ne pas ralentir le jeu.

L'outil et le modèle présentés dans ce mémoire répondent donc aux objectifs que nous nous étions fixés au début de celui-ci. La flexibilité du modèle développé est extrêmement intéressante et il serait certainement intéressant de l'appliquer à d'autres contextes. Puisqu'il s'agit d'un modèle Turing-complet, il peut, par exemple, être utilisé partout où une machine de Turing peut être utilisée. Il peut aussi avantageusement remplacer les automates probabilistes et les chaînes de Markov cachées. On remarque en effet que notre modèle combine ces deux concepts, les automates étendus avec priorités se substituant aux machines de Turing, et que c'est là sa grande force.

Annexe A

Présentation du protocole MIDI

MIDI est un acronyme pour « Musical Instrument Digital Interface » et est d'abord et avant tout un protocole de communication qui a été développé afin de permettre aux instruments électroniques développés par différentes compagnies de s'interconnecter entre eux et d'échanger des données. Il a été introduit en 1982 et a permis, dès 1983, de connecter ensemble les deux premiers instruments MIDI : le Prophet-600 de *Sequential Circuits* et le JX3P de *Roland*. De ce fait, on peut dire que le protocole MIDI a grandement accéléré le développement de la communauté de la musique électronique car sans lui, ou tout du moins sans un protocole similaire, les musiciens utilisant des claviers (synthétiseurs) provenant de différents manufacturiers ne pourraient toujours pas interagir ensemble.

La spécification MIDI définit d'abord un connecteur physique et un format de messages pour l'interconnexion de périphériques et leur contrôle en « temps réel ». Une addition subséquente définit un format de stockage standardisé afin que les données d'une représentation (ou interprétation) d'une pièce puissent être stockées, échangées entre différents périphériques et relues à une date ultérieure. À la section A.1, nous parlerons du format de messages utilisé par le protocole MIDI et présenterons les principaux messages MIDI. Puis, aux sections A.2 et A.3, nous décrirons la norme General MIDI et son influence sur le développement des fichiers standards MIDI. Enfin, à la section A.4, nous décrirons la technologie SoundFont qui a permis aux musiciens de créer leurs propres instruments à partir de leurs propres échantillons. Dans cette même section, nous glisserons aussi un mot sur la

contrepartie standardisée aux Soundfonts : les sons téléchargeables (« Downloadable Sounds » ou DLS en anglais).

A.1 Les messages MIDI

La spécification MIDI originale définit un canal de communication fonctionnant à 31,5 Kbits/sec où tous les messages sont transmis en série. C'est pourquoi il a fallu trouver un format de messages compact et efficace pouvant être transmis en temps réel sans produire de sautillement (« jitter » en anglais) sur une telle ligne. À cet effet, les concepteurs de MIDI proposèrent d'utiliser des messages de 10 à 30 bits divisés en décuplets comprenant chacun 1 bit de départ, 8 bits de données et 1 bit d'arrêt. De plus, ils proposèrent d'omettre le premier décuplet d'un message lorsqu'il s'agit d'un message de canal répété plusieurs fois de file ; ceci afin d'augmenter le nombre de messages transmis par seconde et ainsi la fidélité de reproduction rythmique.

Un autre avantage de l'utilisation d'un tel format de messages est sa flexibilité. Par exemple, on peut transposer une pièce en temps réel ou corriger une fausse note sans reprendre toute la pièce à partir de zéro. Par contraste, même les meilleurs algorithmes de traitement de signaux qu'on connaît aujourd'hui ne peuvent effectuer une correction aussi fine que le remplacement d'une seule note.

Les principaux messages MIDI se nomment (en anglais) : Note on, Note off, Program change, Control change, Pitch wheel change, Channel pressure, Polyphonic key pressure et System exclusive.

Les deux premiers représentent, respectivement, le déclenchement et le relâchement d'une note. Ils prennent en paramètre le numéro de la note enfoncée ainsi que la vitesse à laquelle cette note a été enfoncée. Ce deuxième paramètre représente généralement un volume individuel pour chaque note (à ne pas confondre avec le volume global du synthétiseur). En ce sens, il est généralement ignoré par l'évènement Note off. De plus, lorsqu'il est égal à zéro, le message Note on est équivalent au message Note off.

Le message Program change représente un changement d'instrument sur le synthétiseur. Il prend en paramètre le numéro d'identification du nouvel instrument à utiliser. Dans la

spécification MIDI 1.0, ces numéros d'identification n'étaient pas standardisés, ce qui causait plusieurs problèmes d'incompatibilité entres les instruments. Toutefois, la norme General MIDI, que nous verrons à la section A.2, a complètement résolu ce problème.

Le message Control change permet de contrôler toute une variété de réglages dans un synthétiseur. Son premier paramètre est un numéro de « contrôleur » qui représente le réglage à modifier. Par exemple, le « contrôleur » zéro représente le numéro de banque. Modifier la valeur de ce « contrôleur » change le sens du message Program change et permet à celui-ci d'adresser plus de 128 instruments. Le deuxième paramètre du message Control change est, comme vous l'aurez compris, la nouvelle valeur pour le « contrôleur » sélectionné. Le standard définit près de 128 « contrôleurs » mais n'oblige pas les manufacturiers de claviers à les supporter tous. En fait, seule une dizaine est obligatoire.

Le message Pitch wheel change représente une variation dans la molette de modulation, ce qui permet de faire varier la fréquence des notes produites par le synthétiseur. Ses deux paramètres de sept bits (huit bits en fait mais le huitième vaut toujours zéro) n'en forment en fait qu'un seul de quatorze bits qui représente la variation de fréquence. La valeur 8192 représente la position centrale (donc aucun changement).

Les deux messages suivants, Channel pressure et Polyphonic key pressure, sont généralement regroupés par les manufacturiers de claviers MIDI sous le nom « d'aftertouch ». Ceci fait référence à la capacité de certains claviers à détecter la pression exercée par le claviériste lorsque celui-ci maintient une ou plusieurs touches enfoncées. De ce fait, il est légitime de se demander pourquoi le standard MIDI définit deux messages pour représenter une seule et même fonctionnalité. En fait, la raison est la même qui a poussé les concepteur du standard à rendre optionnels la majorité des « contrôleurs » du message Control change : il existe toute une variété de claviers dont certains sont plus sophistiqués que d'autres. Ainsi, le message Channel pressure est destiné aux claviers qui ne possèdent qu'un seul capteur de pression pour tout le clavier retournant la pression maximale exercée sur toutes les touches à un instant donné. À l'inverse, le message Polyphonic key pressure est destiné aux claviers plus sophistiqués qui possèdent un capteur de pression pour chaque touche.

Enfin, le dernier message, System exclusive, sert à représenter toutes les fonctionnalités supportées par un clavier mais non supportées par la norme MIDI. Il prend en paramètre le numéro d'identification du manufacturier du clavier (qui doit être enregistré auprès de la *MIDI Manufacturers Association*). De plus, ce message est toujours suivi par un flux de données brutes qui se termine obligatoirement par le message End of exclusive. Si le clavier reconnaît le numéro d'identification de manufacturier comme étant le sien, il devra traiter ce flux de données. Sinon, il devra l'ignorer.

Il existe aussi quelques messages MIDI plus ésotériques que nous n'avons pas décris qui sont (en anglais) : Song position pointer, Song select, Start, Continue, Stop, Active sensing et Reset. Toutefois, ceux que nous venons de décrire constituent généralement près de 95% du flux MIDI.

A.2 La norme General MIDI

Après la publication de la première version du standard MIDI, certaines de ses faiblesses devinrent évidentes. Notamment, il apparut que les numéros d'identifications des instruments n'étaient pas standardisés, ce qui causait des problèmes de compatibilité entre les différents instruments. Ceci pouvait par exemple mener à des situations absurdes lorsqu'un musicien essayait d'utiliser un instrument provenant d'un manufacturier A pour contrôler un autre instrument provenant d'un manufacturier B si les deux manufacturiers n'utilisaient pas la même convention pour la numérotation des instruments (ce qui était peu probable).

C'est pourquoi, en 1991, la *MIDI Manufacturers Association* adopta un addendum au standard MIDI qui ajouta quelques critères minimaux auxquels devraient satisfaire les manufacturiers d'instruments MIDI :

1. Supporter au minimum 24 voix (notes) simultanées dont 16 mélodiques et 8 percussives,
2. Supporter les 16 canaux MIDI simultanément (avec le canal 10 réservé pour les percussions),
3. Supporter la polyphonie (plusieurs notes en même temps) sur chaque canal individuel,
4. Répondre au paramètre de vélocité des messages Note on et Note off,

5. Supporter un ensemble minimal de 128 instruments standardisés,
6. Réserver le canal 10 pour les percussions,
7. Supporter un ensemble minimal de contrôleurs standardisés, et enfin
8. Supporter deux messages « systèmes exclusifs universels » permettant d'activer et de désactiver le mode General MIDI pour les synthétiseurs ayant des fonctionnalités additionnelles à offrir.

Le tableau A.1 présente la liste des instruments définis par la norme General MIDI. Le tableau A.2 présente la correspondance entre les notes jouées sur le canal 10 et les sons de percussions résultants. Enfin, le tableau A.3 présente la liste des contrôleurs définis par la norme General MIDI.

A.3 Les fichiers standards MIDI

Les fichiers standards MIDI ont été développés pour permettre aux musiciens de stocker leurs compositions dans un format commode que leurs synthétiseurs pourraient facilement comprendre. Ils sont organisés en une successions de blocs appelés CHUNKS. Il y a deux types de CHUNKS définis par le standard officiel : le CHUNK MThd qui correspond à l'entête du fichier et les CHUNKS MTrk qui correspondent aux pistes de données du fichiers. Si d'autres types de CHUNKS sont présents dans un fichier, un synthétiseur est autorisé à les ignorer.

L'entête du fichier contient des informations à propos du format du fichier, du nombre de pistes de données et de la résolution de temps utilisée dans les pistes de données du fichier. Le format du fichier peut être l'un des suivants :

1. Format 0 : un seul MTrk contenant un ou plusieurs canaux MIDI,
2. Format 1 : plusieurs MTrk joués simultanément contenant chacun un seul canal MIDI plus un MTrk contenant les informations de tempo,
3. Format 2 : plusieurs MTrk joués en séquence contenant chacun un ou plusieurs canaux MIDI (rarement utilisé).

La résolution de temps peut soit être spécifiée en nombre d'impulsions par quart de note (noire) ou par un nombre de trames et de sous-trames par secondes (time-code SMPTE).

	Pianos		Instruments à anches
000	Grand piano acoustique	064	Saxophone soprano
001	Piano acoustique	065	Saxophone alto
002	Grand piano électrique	066	Saxophone ténor
003	Piano bastringue (Honky-Tonk Piano)	067	Saxophone baryton
004	Piano électrique (Rhodes piano)	068	Hautbois
005	Piano - effet chorus	069	Cor anglais
006	Clavecin	070	Basson
007	Clavinet	071	Clarinette
	Percussions chromatiques		**Autres instruments à vent**
008	Celesta	072	Piccolo
009	Glockenspiel	073	Flûte
010	Boîte à musique	074	Flûte à bec
011	Vibraphone	075	Flûte de pan
012	Marimba	076	Bouteille - souffle
013	Xylophone	077	Shakuhachi
014	Cloches tubulaires	078	Sifflet
015	Tympanon (Dulcimer)	079	Ocarina
	Orgues (Organ)		**Synthé - Solo**
016	Orgue Hammond	080	Signal carré (Lead 1 (Square))
017	Orgue à percussion	081	Signal dents de scie (Lead 2 (Sawtooth))
018	Orgue - Rock	082	Orgue à vapeur (Lead 3 (Calliope lead))
019	Grandes orgues (Church organ)	083	Chiffer (Lead 4 (Chiff lead))
020	Harmonium (Reed Organ)	084	Charang (Lead 5 (Charang))
021	Accordéon	085	Voix solo (Lead 6 (Voice))
022	Harmonica	086	Signal dent de scie en quinte (Lead 7 (Fifths))
023	Accordéon tango (Bandonéon)	087	Basse et Solo (Lead 8 (Brass + lead))
	Guitares		**Synthé - Ensembles**
024	Guitare classique (Acoustic guitar (nylon))	088	Fantaisie (Pad 1 (New age))
025	Guitare sèche (Acoustic Guitar (steel))	089	Son chaleureux (Pad 2 (Warm))
026	Guitare électrique - Jazz	090	Polysynthé (Pad 3 (Polysynth))
027	Guitare électrique - son clair	091	Chœur (Pad 4 (Choir))
028	Guitare électrique - sourdine	092	Archet (Pad 5 (Bowed))
029	Guitare saturée (Overdriven guitar)	093	Métallique (Pad 6 (Metallic))
030	Guitare avec distorsion	094	Halo (Pad 7 (Halo))
031	Harmoniques de guitare	095	Balai (Pad 8 (Sweep))
	Basses		**Synthé - Effets**
032	Basse acoustique sans frettes	096	Pluie de glace
033	Basse électrique	097	Trames sonores
034	Basse électrique - médiator	098	Cristal
035	Basse sans frettes	099	Atmosphère
036	Basse - slap 1	100	Brillance
037	Basse - slap 2	101	Gobelins
038	Basse synthé 1	102	Échos
039	Basse synthé 2	103	Espace (Sci-Fi)
	Cordes et orchestre		**Instruments ethniques**
040	Violon	104	Sitar
041	Violon alto	105	Banjo
042	Violoncelle	106	Shamisen
043	Contrebasse	107	Koto
044	Cordes - trémolo	108	Kalimba
045	Cordes - pizzicato	109	Cornemuse
046	Harpe	110	Viole
047	Timbales	111	Shannaï
	Ensembles		**Percussions**
048	Ensemble acoustique à Cordes 1	112	Clochettes
049	Ensemble acoustique à Cordes 2	113	Agogo
050	Cordes synthé 1	114	Batterie métallique
051	Cordes synthé 2	115	Planchettes
052	Chœur - "Aah" (Choir Aahs)	116	Wood-block
053	Chœur - "Ooh" (Voice Oohs)	117	Tom mélodique
054	Voix synthétique	118	Tambour synthétique
055	Coup d'orchestre	119	Cymbale - inversée
	Cuivres (Brass)		**Effets sonores (bruitages)**
056	Trompette	120	Guitare - bruit de frette
057	Trombone	121	Respiration
058	Tuba	122	Rivage
059	Trompette en sourdine	123	Gazouilli
060	Cor d'harmonie (French horn)	124	Sonnerie de téléphone
061	Section de cuivres	125	Hélicoptère
062	Cuivres synthé	126	Applaudissements
063	Cuivres synthé	127	Coup de feu

TAB. A.1 – Liste des instruments General MIDI

35	Grosse caisse médium	59	Cymbale ride aigue
36	Grosse caisse haute	60	Bongo aigu
37	Coup de métronome	61	Bongo grave
38	Caisse claire 1	62	Congas aigu sourd
39	Claquement de main	63	Congas aigu ouvert
40	Caisse claire 2	64	Congas grave
41	Tom basse grave	65	Timbales aigu
42	Charley frappé	66	Timbales grave
43	Tom basse aigu	67	Cloche agogo aiguë
44	Charley au pied	68	Cloche agogo grave
45	Tom médium 4	69	Cabasa
46	Charley ouvert	70	Maracas
47	Tom médium 3	71	Sifflet aigu
48	Tom médium 2	72	Sifflet grave
49	Cymbale crash	73	Guiro court
50	Tom aigu	74	Guiro long
51	Ride	75	Claves
52	Cymbale china	76	Woodblock aigu
53	Cymbale ride/cup	77	Woodblock grave
54	Tambourin	78	Cuica assourdie
55	Cymbale splash	79	Cuica ouverte
56	Clochette de vache	80	Triangle tenu
57	Crash 2	81	Triangle libre
58	Vibra-slap		

TAB. A.2 – Correspondance notes vs. instruments percussifs sur le canal 10

1	Modulation
6	Data Entry MSB
7	Volume
10	Pan
11	Expression
38	Data Entry LSB
64	Sustain
100	RPN LSB
101	RPN MSB
121	Reset all controllers
123	All notes off

TAB. A.3 – Liste des contrôleurs General MIDI

Il est crucial de comprendre l'impact qu'a la résolution de temps sur l'interprétation d'un fichier MIDI. Par exemple, si celle-ci est de 16 impulsions par quart de note alors chaque delta de temps sera équivalent à une quadruple croche. Par contre, si celle-ci est spécifiée comme étant de 25 trames et 40 sous-trames par seconde alors chaque delta de temps sera équivalent à une milliseconde.

Quant aux pistes de données, ils consistent en une liste d'évènements MIDI, chacun précédé d'un delta de temps par rapport au dernier évènement dans la piste. De cette façon, un synthétiseur (ou un séquenceur) qui lit un fichier MIDI peut rejouer une séquence MIDI avec exactitude ou tout du moins de façon aussi précise que la résolution de temps spécifiée dans l'entête le permet. Les pistes de données peuvent aussi contenir des méta-évènements MIDI qui ajoutent certaines informations au fichier. Certains de ces méta-évènements sont utiles comme le marqueur de fin de piste (qui est d'ailleurs obligatoire) et l'évènement tempo, d'autres moins comme la signature de temps et la signature de clef qui n'ont absolument aucun effet sur le rendu d'une séquence MIDI mais peuvent être utilisés par les logiciels de notation de musique pour transcrire un fichier MIDI en partition. Remarquons que ces deux derniers évènements ne seront jamais présents dans une séquence MIDI enregistrée « live » à partir d'un instrument MIDI et qu'il faut manuellement les ajouter.

A.4 La technologie SoundFont

Les Soundfonts sont un format de « sons téléchargeables » développé par *Creative Labs* et *E-mu Technologies*. Ceci signifie qu'il s'agit de collections d'instruments qui peuvent être envoyées à un synthétiseur. En ce sens, elles sont destinées à être utilisées avec des synthétiseurs et/ou des cartes de sons qui utilisent un modèle de synthèse par table d'ondes.

Elles sont structurées en trois niveaux : les préréglages, les instruments et les échantillons. Les préréglages constituent le plus haut niveau et correspondent aux changements de programme (instruments) MIDI. Ceux-ci sont formés d'un ou plusieurs instruments superposés auxquels on applique différents paramètres tels que l'intervalle de notes MIDI affectées par cet instrument, l'intervalle de vélocité, l'atténuation, la position spatiale, les enveloppes de

volume et de modulation, les oscillateurs à basse fréquence, le chœur et la réverbération. Ces instruments sont à leur tour constitués d'échantillons auxquels on peut appliquer les mêmes paramètres qu'on applique aux instruments dans les préréglages mais à un niveau supérieur. Enfin, les échantillons sont constitués d'échantillons auxquels on applique les paramètres suivants : début de la boucle, fin de la boucle, note de base et correction de fréquence.

BIBLIOGRAPHIE

[1] AMES, C. The markov process as a compositional model : A survey and tutorial. *Leonardo Music Journal 22*, 2 (1989), 175–187.

[2] AUDIOKINETIC. Wwise 2007.1. PC, 2007.

[3] BENITO, J. M., AND HERNÁNDEZ, P. Modelling segregation through Cellular Automata : A theoretical answer. Tech. rep., Instituto Valenciano de Investigaciones Económicas, S.A. (Ivie), 2007.

[4] BILES, J. A. Genjam : A genetic algorithm for generating jazz solos. In *Proceedings of the 1994 International Computer Music Conference* (1994), pp. 131–137.

[5] BURRASTON, D., EDMONDS, E., LIVINGSTONE, D., AND MIRANDA, E. Cellular Automata in MIDI based Computer Music. In *Proceedings of the 2004 International Computer Music Conference* (2004), pp. 71–78.

[6] CENTRE NATIONAL DE RESSOURCES TEXTUELLES ET LEXICALES. Le portail lexical du centre national de ressources textuelles et lexicales.

[7] CHENG, K.-T., AND KRISHNAKUMAR, A. S. Automatic functional test generation using the extended finite state machine model. In *Proceedings of the 30th international conference on Design automation conference* (1993), pp. 86–91.

[8] CHIRICOTA, Y., AND GILBERT, J.-M. IMTool : an open framework for interactive music composition. *Proceedings of the 2007 conference on Future Play* (2007), 181–188.

[9] CHOMSKY, N. *Aspects of the Theory of Syntax.* MIT Press, 1965.

[10] COHEN, A. Film music : Perspectives from cognitive psychology. In *Music and Cinema*, J. Buhler, C. Flinn, and D. Neumeyer, Eds. 2000, pp. 360–377.

[11] CONWAY, J. H. The game of life, 1970.

[12] COOK, P. R. *Real Sound Synthesis for Interactive Applications.* AK Peters, Ltd., 2002.

[13] DELERUE, O., ASSAYAG, G., AND AGON, C. Etude et réalisation d'opérateurs rythmiques dans OpenMusic, un environnement de programmation appliqué à la composition musicale. In *Actes des Journées d'Informatique Musicales JIM'98, La Londe, les Maures* (1998).

[14] DUBUIS, E. Graphical user interfaces : Mess them up ! In *Winter School of Computer Graphics 1996* (1996).

[15] FIRELIGHT TECHNOLOGIES. FMOD Designer. PC & Mac, 2007.

[16] FOATA, D., AND FUCHS, A. *Processus stochastiques : Processus de Poisson, chaînes de Markov et martingales.* Dunod, 2002.

[17] FOLEY, J. D., VAN DAM, A., FEINER, S. K., AND HUGHES, J. F. *Computer Graphics : Principles and Practice in C, 2nd Edition.* Addison-Wesley, 1995.

[18] FREE SOFTWARE FOUNDATION. Gnu lesser general public license v2 (gnu lgpl).

[19] GOLDBERG, D. E. Real-coded genetic algorithms, virtual alphabets, and blocking. Tech. rep., University of Illinois at Urbana-Champaign, 1990.

[20] GOLDBERG, D. E., AND HOLLAND, J. H. Genetic Algorithms and Machine Learning. *Machine Learning 3*, 2-3 (1988), 95–99.

[21] GOUDE, K., AND O'KEEFE, S. A Cellular Automata Model for Dictyostelium Discoideum. Tech. rep., University of York, 2005.

[22] GRAME COMPUTER MUSIC RESEARCH LAB. MidiShare. Open source.

[23] GREMLIN INTERACTIVE. Northstar. C64, 1988.

[24] GRIFFEATH, D. Cyclic random competition : a case history in experimental mathematics. *Notices of the American Mathematical Society 35* (1988), 1472–1480.

[25] GRUDIN, J. The case against user interface consistency. *Communications of the ACM 32*, 10 (1989), 1164–1173.

[26] HANAPPE, P., GREEN, J., LETZ, S., NENTWIG, M., AND SCHMITT, A. Fluidsynth. Open source.

[27] HEDGES, S. A. Dice music in the eighteenth century. *Music & Letters 59*, 2 (1978), 180–187.

[28] HIDALGO, J. L., CAMAHORT, E., ABAD, F. J., AND DOMINGO, A. Ml-system : Generating complex geometries using l-systems. In *Proceedings of the International Digital Games Conference - GAMES 2006* (September 2006 2006), pp. 225–228.

[29] HILBERT, D. Über die stetige abbildung einer linie auf ein flächenstück. *Math. Ann. 38* (1891), 459–460.

[30] HUANG, C.-M., AND LO, C.-M. An EFSM-Based Multimedia Synchronization Model and the Authoring System. *IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS 14* (1996).

[31] ICKING, W., Ed. *Der allezeit fertige Polonoisen und Menuettencomponist.* 1995. Transcription partielle. Disponible en ligne. `http://icking-music-archive.org/scores/kirnberger/menuet.pdf`.

[32] ISAACSON, D. L., AND MADSEN, R. W. *Markov Chains : Theory and Applications.* John Wiley & Sons, Inc., 1976.

[33] KELLOGG, W. A. The dimensions of consistency. In *Coordinating User Interfaces for Consistency*, J. Nielsen, Ed. Morgan Kauffmann, 1989.

[34] KIRNBERGER, J. P. *Der allezeit fertige Polonoisen und Menuettencomponist.* 1757. Disponible en ligne. `http://www.musikwissenschaft.uni-mainz.de/Musikinformatik/schriftenreihe/nr15/Kirnframe.html`.

[35] KOZA, J. R. *Genetic Programming : On the Programming of Computers by Means of*

Natural Selection. MIT Press, Cambridge, MA, USA, 1992.

[36] KRISHNAKUMAR, A. S. Reachability and Recurrence in Extended Finite State Machines : Modular Vector Addition Systems. In *Proceedings of the 5th International Conference on Computer Aided Verification* (1993), Springer.

[37] LANGSTON, P. Six techniques for algorithmic music composition. In *Proceedings of the 15th International Computer Music Conference (ICMC)* (1989).

[38] LIPSCOMB, S. D., AND TOLCHINSKY, D. E. The Role of Music Communication in Cinema. *Musical Communication* (2005).

[39] LUCASFILM GAMES. Ballblazer. Atari, 1984.

[40] MANOUSAKIS, S. Musical l-systems. Master's thesis, Conservatoire Royal de La Haye, 2006.

[41] MCALPINE, K., MIRANDA, E. R., AND HOGGAR, S. Making music with algorithms : A case-study system. *Computer Music Journal 23*, 2 (1999), 19–30.

[42] MCCLURE, M. Self-Similar Structure in Hilbert's Space-Filling Curve. *Mathematics Magazine 76*, 1 (2003), 40–47.

[43] MIRANDA, E. R. Evolving cellular automata music : From sound synthesis to composition. In *Proceedings of the Workshop on Artificial Life Models for Musical Applications / European Conference on Artificial Life* (2001).

[44] MOORE, E. F. Gedanken-experiments on sequential machines, Automata Studies. *Annals of Mathematical Studies 34* (1956), 129–153.

[45] NINTENDO. The Legend of Zelda : Ocarina of Time. Nintendo 64 & Nintendo GameCube, 1998.

[46] NINTENDO. New Super Mario Bros. Nintendo DS, 2006.

[47] NOGUCHI, H. Mozart - Musical Game in C K. 516f, 1997. Disponible en ligne. `http://www.asahi-net.or.jp/~rb5h-ngc/s/k516f.htm`.

[48] ÖREN, T., AND YILMAZ, L. Quality principles for the ergonomics of human-computer interfaces of modeling and simulation software. In *Proceedings of the 2005 International Conference on Human-Computer Interface Advances for Modeling and Simulation (SIMCHI'05), January 23–25, 2005, New Orleans, Louisiana* (2005), pp. 5–11.

[49] PHILLIPS, D. Computer music journal reviews : Notes from the metalevel : Introduction to algorithmic music composition. *Computer Music Journal 29*, 3 (2005), 91–93.

[50] PRUSINKIEWICZ, P. Score generation with l-systems. In *Proceedings of the 1986 International Computer Music Conference* (1986), pp. 455–457.

[51] PRUSINKIEWICZ, P., AND LINDENMAYER, A. *The algorithmic beauty of plants*. Springer-Verlag, New York, 1990.

[52] PUCKETTE, M. The Patcher. In *Proceedings of the 1988 International Computer Music Conference* (1988), pp. 420–429.

[53] PUCKETTE, M. Pure data : another integrated computer music environment. In *Proceedings of the Second Intercollege Computer Music Concerts, Tachikawa* (1996), pp. 37–41.

[54] PUCKETTE, M. Max at Seventeen. *Computer Music Journal 26*, 4 (2002), 31–43.

[55] SHNEIDERMAN, B. Direct manipulation : A step beyond programming languages. *IEEE Computer 16*, 8 (1983), 57–69.

[56] SHNEIDERMAN, B. Direct manipulation for comprehensible, predictable and controllable user interfaces. In *Intelligent User Interfaces* (1997), pp. 33–39.

[57] SIPSER, M. *Introduction to the Theory of Computation.* PWS Publishing Company, 1997.

[58] SONY COMPUTER ENTERTAINMENT OF AMERICA. The Mark of Kri. PlayStation 2, 2002.

[59] SPECTOR, L., AND ALPERN, A. Induction and recapitulation of deep musical structure. In *Proceedings of International Joint Conference on Artificial Intelligence, IJCAI'95 Workshop on Music and AI* (Montreal, Quebec, Canada, 20-25 Aug. 1995).

[60] SQUARE. Final fantasy. Nintendo Entertainment System, 1987.

[61] SWEETSER, P. How to build evolutionary algorithms for games. In *AI Game Programming Wisdom 2*, S. Rabin, Ed. Charles River Media, 2004.

[62] TAUBE, H. K. *Notes from the Metalevel : Introduction to Algorithmic Music Composition.* Routledge, 2004.

[63] THORN EMI, AND CREATIVE SPARKS. Black hawk. C64, 1984.

[64] TOUGNE, L. Recognition of figures with two-dimensional cellular automata. Tech. rep., Ecole Normale SupÃ©rieure de Lyon, 1994.

[65] TRACTINSKY, N. Does aesthetics matter in human-computer interaction? In *Mensch & Computer 2005 : Kunst und Wissenschaft — Grenzüberschreitungen der interaktiven ART, Munich* (2005), C. Stary, Ed., Oldenbourg Verlag, pp. 29–42.

[66] TURAKAINEN, P. Generalized Automata and Stochastic Languages. In *Proceedings of the American Mathematical Society* (1969), vol. 21, pp. 303–309.

[67] WHITLEY, D. A genetic algorithm tutorial. *Statistics and Computing 4*, 2 (1994), 65–85.

[68] WOLFRAM, S. *A New Kind of Science.* Wolfram Media, 2002.

[69] WORTH, P., AND STEPNEY, S. Growing music : musical interpretations of l-systems. In *EvoMUSART workshop, EuroGP 2005, Lausanne, Switzerland, March 2005* (2005), pp. 545–550.

[70] YVON, F., AND DEMAILLE, A. *Théorie des Langages : Notes de Cours*, 2005.