

Table des matières

[Introduction 3](#_Toc168225814)

[I. Phase Initiale : Modélisation et Préparation de l'Automate 4](#_Toc168225815)

[1. Modélisation orientée objet de l’automate 4](#_Toc168225816)

[a) Concepts de base des automates finis 4](#_Toc168225817)

[b) Définition des classes et objets 4](#_Toc168225818)

[c) Fonction de lecture d’un automate 5](#_Toc168225819)

[2. Fonctionnalités principales du code 8](#_Toc168225820)

[a) Rendre un automate déterministe s’il ne l’est pas : Algorithme de déterminisation 8](#_Toc168225821)

[b) Rendre un automate complet s’il ne l’est pas : Algorithme de complétion 12](#_Toc168225822)

[c) Minimiser un automate qui ne l’est pas 15](#_Toc168225823)

[d) Affichage du graphe de transition de l’automate sous forme graphique 18](#_Toc168225824)

[II. Phase applicative 19](#_Toc168225825)

[1. Idée 19](#_Toc168225826)

[2. Étapes de Réalisation 19](#_Toc168225827)

[a) Définition des Règles Harmoniques et Rythmiques 19](#_Toc168225828)

[b) Classe Automate pour la Génération de Musique 20](#_Toc168225829)

[c) Génération de Fichiers MIDI 21](#_Toc168225830)

[d) Création de l’interface utilisateur 21](#_Toc168225831)

[e) Résultats Obtenus 23](#_Toc168225832)

[III. Conclusion et perspectives 24](#_Toc168225833)

[1. Bilan du projet 24](#_Toc168225834)

[2. Limites Rencontrées 24](#_Toc168225835)

[3. Perspectives et Améliorations Futures 24](#_Toc168225836)

[IV. Références Bibliographiques et Webographies 25](#_Toc168225837)

[1. Références Bibliographiques 25](#_Toc168225838)

[2. Webographies 25](#_Toc168225839)

# Introduction

La génération de musique algorithmique représente une intersection fascinante entre les arts et les sciences, combinant la créativité musicale avec les principes rigoureux des mathématiques et de l'informatique. Depuis des siècles, les musiciens et les compositeurs ont cherché à systématiser la composition musicale, et avec l'avènement de l'informatique moderne, cette exploration a pris une nouvelle dimension. La musique algorithmique, où les éléments musicaux sont générés par des algorithmes plutôt que composés directement par un humain, permet de créer de vastes variations musicales, parfois impossibles à réaliser manuellement. Les automates finis, en particulier, offrent un cadre puissant pour modéliser et générer des séquences musicales de manière structurée et prédictive, rendant cette approche particulièrement adaptée à l'expérimentation et à l'innovation musicale.

Ce mini-projet vise à concevoir et à implémenter un automate capable de générer des séquences musicales basées sur des règles définies. L'objectif principal est de démontrer comment les concepts des automates finis peuvent être appliqués à la création musicale, en explorant les possibilités offertes par la musique algorithmique. En développant cet automate, nous cherchons à :

* **Modéliser des motifs musicaux** en utilisant une approche orientée objet.
* **Développer des règles harmoniques et rythmiques** pour guider la génération musicale.
* **Implémenter des déclencheurs** internes et externes pour gérer les transitions et variations musicales.
* **Combiner créativité et technique** pour produire des séquences musicales variées et cohérentes.

# Phase Initiale : Modélisation et Préparation de l'Automate

## Modélisation orientée objet de l’automate

### Concepts de base des automates finis

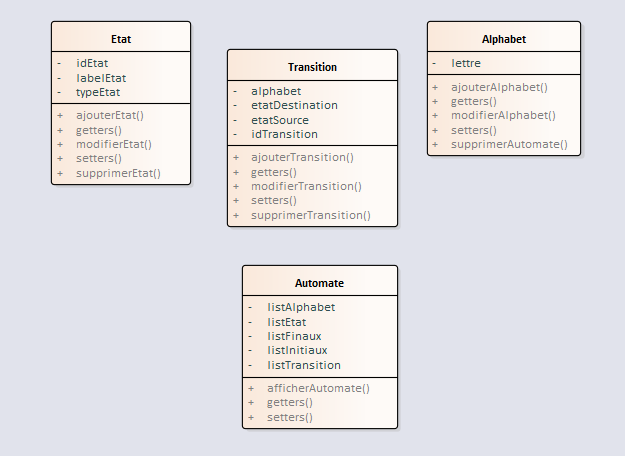
Les automates finis sont des modèles mathématiques utilisés pour représenter des systèmes avec un nombre fini d'états. Un automate fini Aut = (A, Q, I, T, E) est composé de :

* **Un ensemble d'états (Q)** : un ensemble fini d'états de l'automate.
* **Un alphabet (Σ)** : l'alphabet utilisé pour la construction de mots.
* **Un ensemble d’états initiaux (I)** : Un sous-ensemble de Q formant les états dits "initiaux", ou états "de départ"
* **Un ensemble d'états finaux (F)** : un sous-ensemble de Q formant les états dits "terminaux", ou états "finals", ou états "d'acceptation
* **Un ensemble de transition E** : un ensemble de triplets appelés "transitions" E ⊆ Q x A x Q

Ces concepts fondamentaux permettent de structurer un automate fini de manière à modéliser des processus complexes, tels que la génération de séquences musicales, en respectant des règles définies.

### Définition des classes et objets

Pour modéliser un automate de manière orientée objet, nous définissons plusieurs classes représentant les éléments de base de l'automate :



**Figure 1 : Vue d'ensembles des classes**

*Remarques :*

Chaque classe possède un constructeur et les attributs de chaque classe possèdent un getter et un setter.

### Fonction de lecture d’un automate

La fonction **lire\_automate** permet de lire et de créer un automate à partir de données fournies. Elle suit plusieurs étapes clés pour transformer les données d'entrée en un objet automate utilisable.

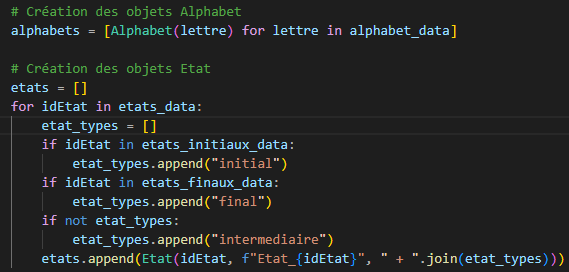
#### Extraction des données

Nous commençons par extraire les différentes parties des données fournies.



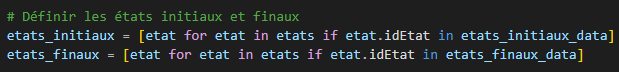
#### Création des objets Alphabet et Etat

Chaque symbole de l'alphabet est transformé en un objet **Alphabet** et chaque état est créé avec son type déterminé (initial, final, ou intermédiaire).



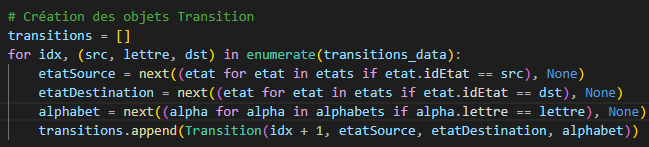
#### Définition des états initiaux et finaux

Nous définissons les listes des états initiaux et finaux à partir des objets état.



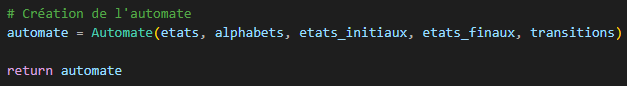
#### Création des objets Transition

Chaque transition est transformée en un objet **Transition**.

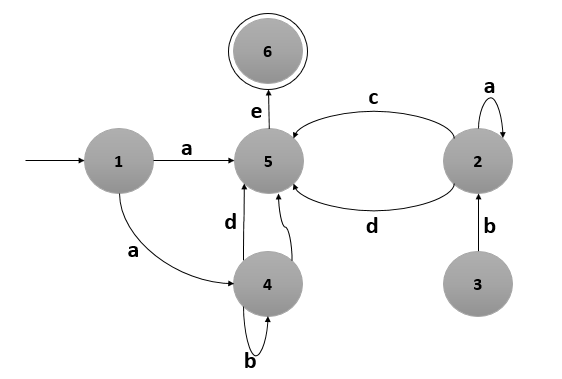


#### Création de l'objet Automate

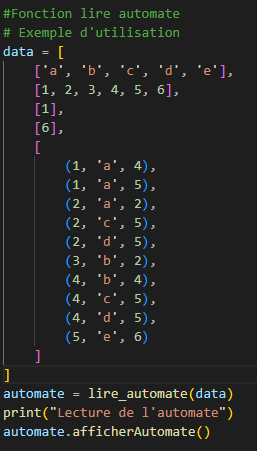
Enfin, nous créons l'automate avec les listes des états, des alphabets, des états initiaux, des états finaux, et des transitions.

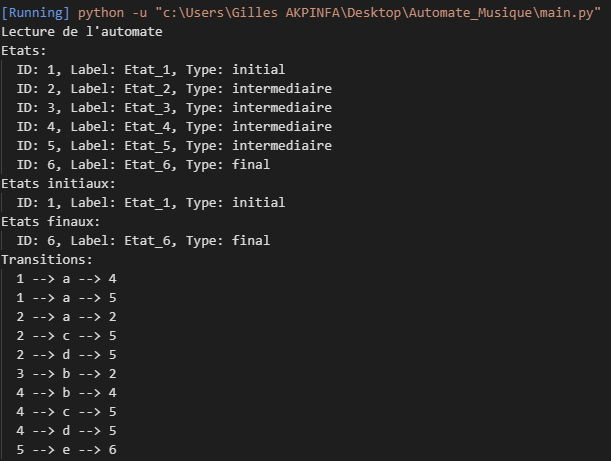


Soit l’automate suivant à lire :



**Figure 2 Exemple d'automate à lire**





## Fonctionnalités principales du code

### Rendre un automate déterministe s’il ne l’est pas : Algorithme de déterminisation

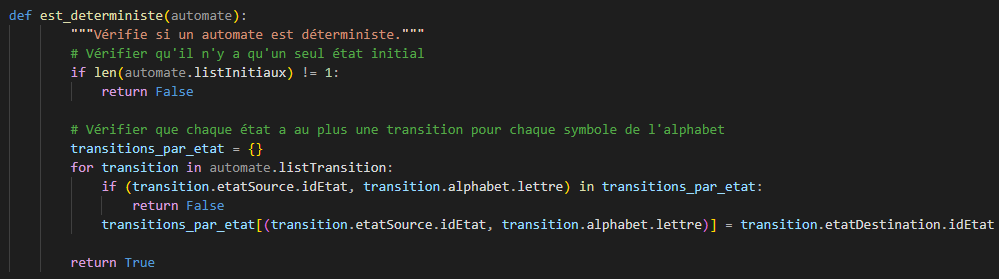
#### Vérifions d’abord si l’automate est déterministe

Pour qu’un automate fini soit déterministe :

* Chacun de ses états doit avoir au plus une transition pour chaque symbole de l'alphabet.
* Il ne doit y avoir qu'un seul état initial.

Codons d’abord une fonction qui permet de vérifier si un automate est déterministe. Pour cela, nous allons utiliser la définition précédente.

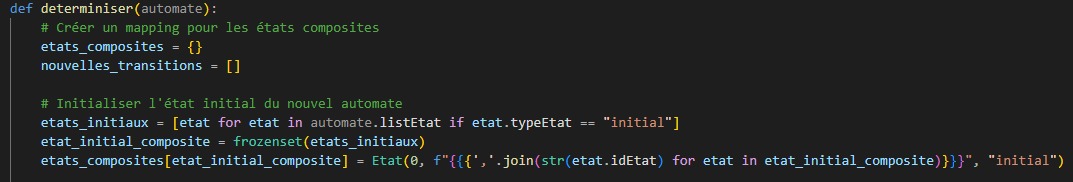
Le code python est le suivant :



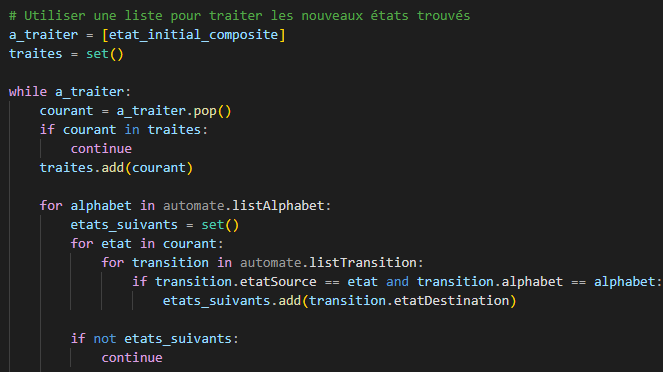
#### Détermination de l’automate non déterministe

Pour déterminiser un automate, nous allons utiliser l’algorithme de déterminisation. Voici ces différentes étapes.

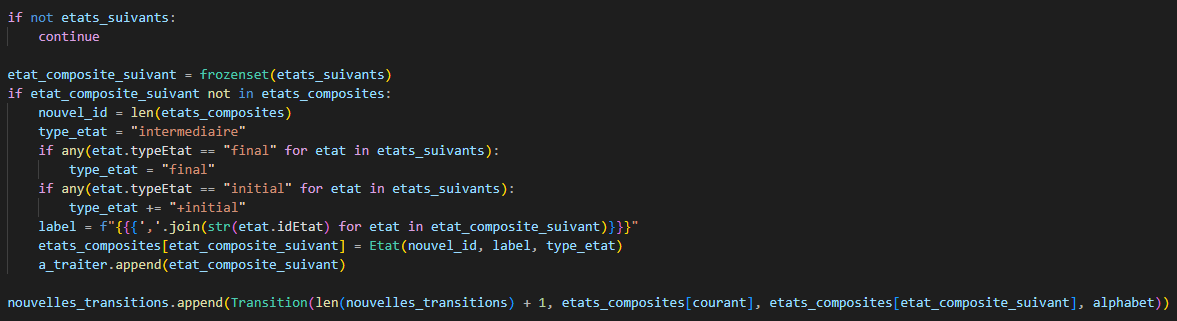
1. **État initial composite :** Nous commençons par créer un état initial composite qui est un ensemble des états initiaux de l'automate original.



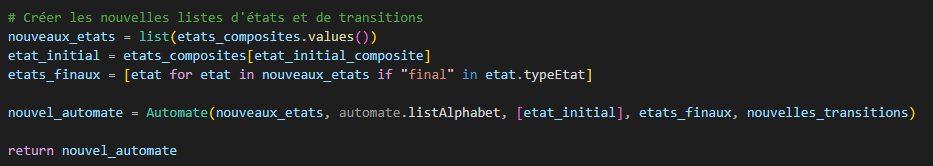
1. **Transitions par ensemble d'états :** Pour chaque état composite et chaque symbole de l'alphabet, nous déterminons les états suivants en utilisant les transitions de tous les états composant l'état courant.



1. **Nouveaux états et transitions :** Les nouveaux ensembles d'états générés sont ajoutés comme des nouveaux états, et les transitions entre ces états sont enregistrées.

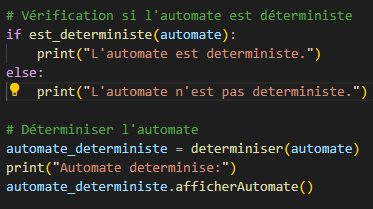


1. **Répétition jusqu'à convergence** : Le processus continue jusqu'à ce que tous les nouveaux ensembles d'états soient traités.

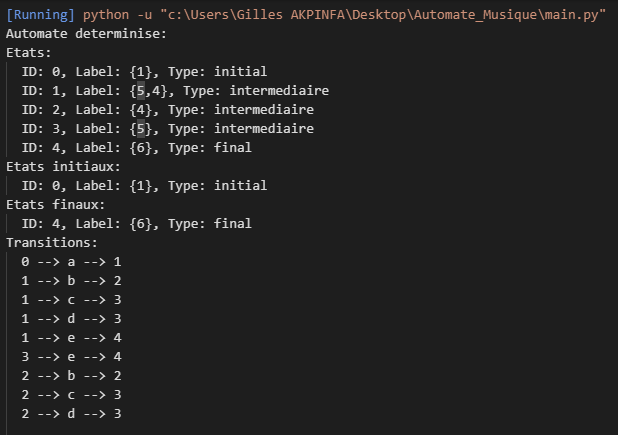


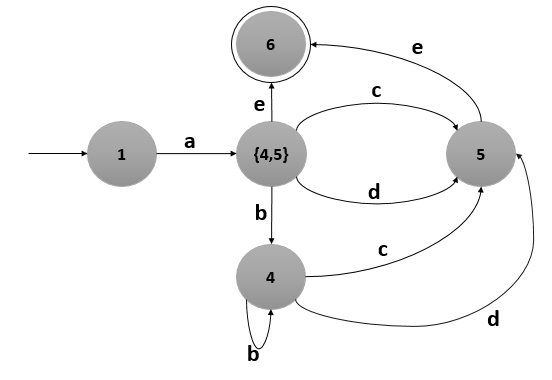
Exemple de fonctionnement

Soit l’automate de la figure… à déterminer :



Résultat obtenu est le suivant :





**Figure 3 Résultat obtenu après la déterminisation de l'automate de la figure 1**

### Rendre un automate complet s’il ne l’est pas : Algorithme de complétion

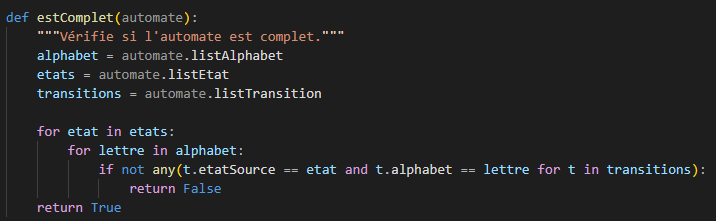
Un automate fini est dit complet si, pour chaque état et chaque symbole de l'alphabet, il existe une transition sortante.

Un automate non complet peut être transformé en automate complet en ajoutant des transitions manquantes vers un état puits. Dans cette section, nous décrivons l'algorithme de complétion et fournissons les fonctions nécessaires pour vérifier la complétion et compléter un automate si nécessaire.

#### Vérification de la Complétion

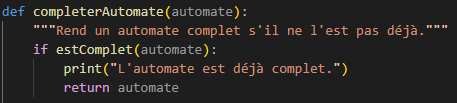
Pour chaque état et chaque symbole de l'alphabet, nous vérifions s'il existe au moins une transition sortante correspondante. Si une transition est manquante, l'automate n'est pas complet.

**Code pour la Vérification de la Complétion**

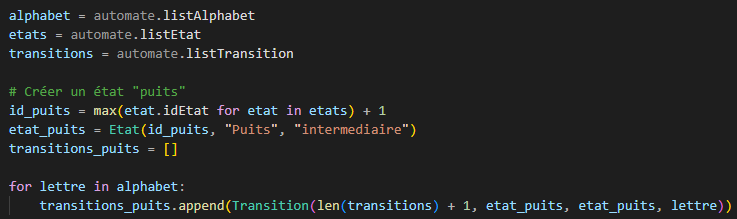


#### Algorithme et code de complétion

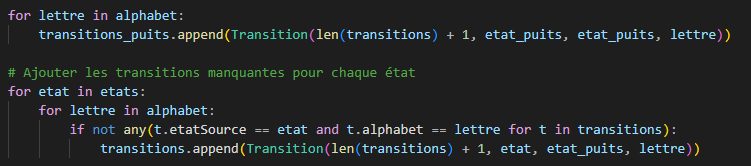
1. **Vérifier la complétion** : Nous commençons par vérifier si l'automate est déjà complet. Si c'est le cas, nous n'effectuons aucune modification et retournons l'automate tel quel.



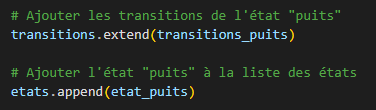
1. **Créer un état "puits"** : Nous créons un nouvel état "puits" et des transitions de cet état vers lui-même pour chaque symbole de l'alphabet.



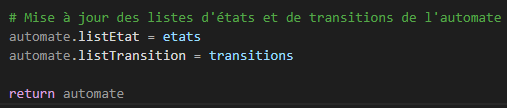
1. **Ajouter les transitions manquantes** : Pour chaque état de l'automate, nous ajoutons une transition vers l'état "puits" pour chaque symbole de l'alphabet manquant.



1. **Ajouter les transitions de l'état "puits"** : Nous ajoutons les transitions de l'état "puits" à lui-même pour chaque symbole de l'alphabet.

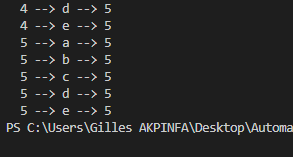


1. **Mettre à jour l'automate** : Nous ajoutons l'état "puits" à la liste des états et mettons à jour les listes d'états et de transitions de l'automate.



**Exemple d’utilisation**

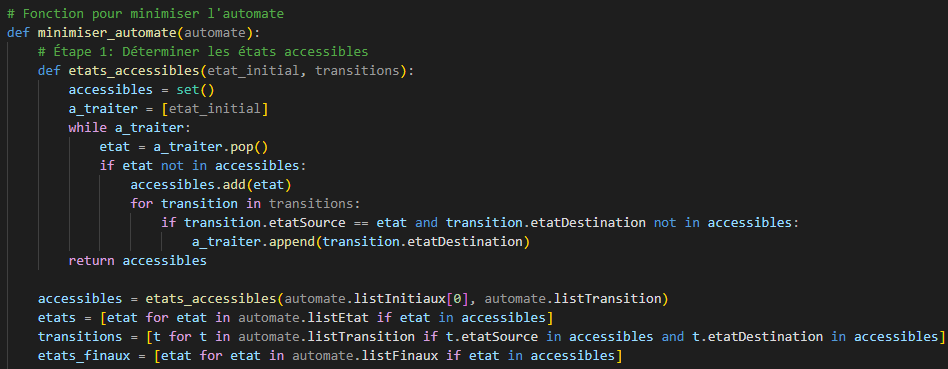
|  |  |
| --- | --- |
|  |  |



### Minimiser un automate qui ne l’est pas

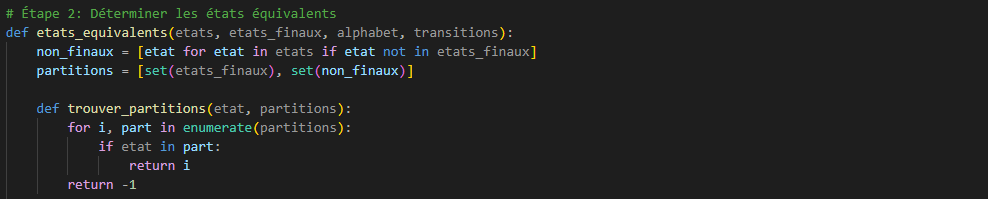
*Algorithme et code*   
  
**Détermination des états accessibles**

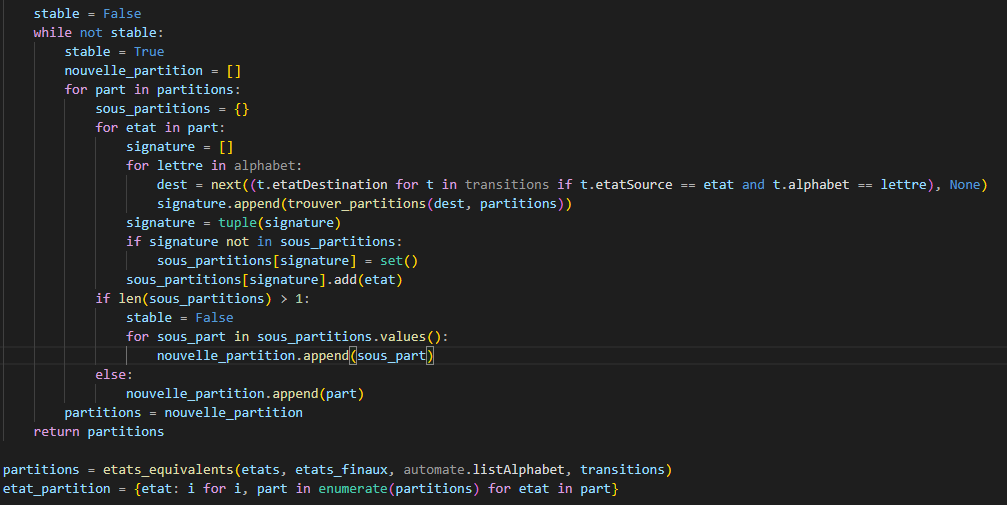
Ce bloc détermine les états accessibles à partir de l'état initial. Les états inaccessibles sont éliminés, ce qui simplifie l'automate en supprimant les états et transitions inutiles.



**Détermination des états équivalents**

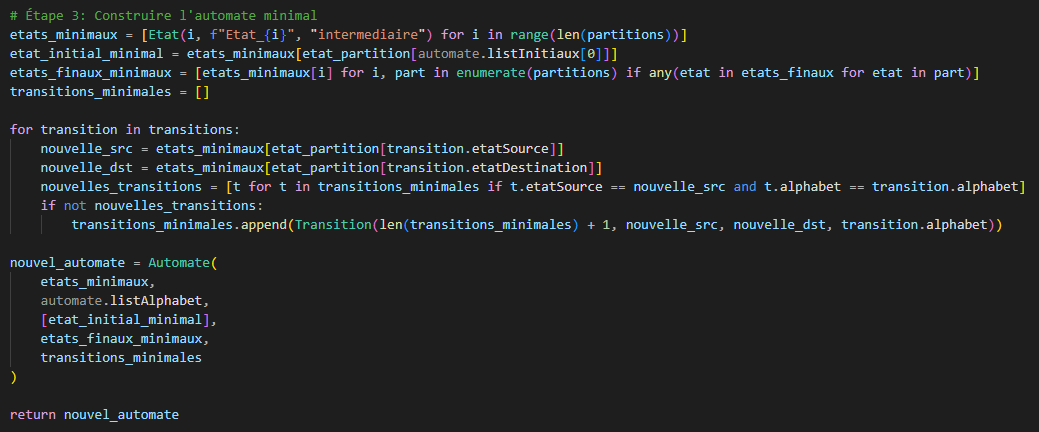
* Les états sont partitionnés en groupes d'états équivalents, où deux états sont considérés équivalents s'ils ne peuvent pas être distingués par aucune séquence d'entrées.
* L'algorithme affine progressivement les partitions jusqu'à ce qu'elles deviennent stables (aucun changement supplémentaire).





**Construction de l’automate minimal**

* Les partitions d'états équivalents sont utilisées pour créer un nouvel automate avec un nombre minimal d'états.
* Les nouvelles transitions sont définies en fonction des partitions.

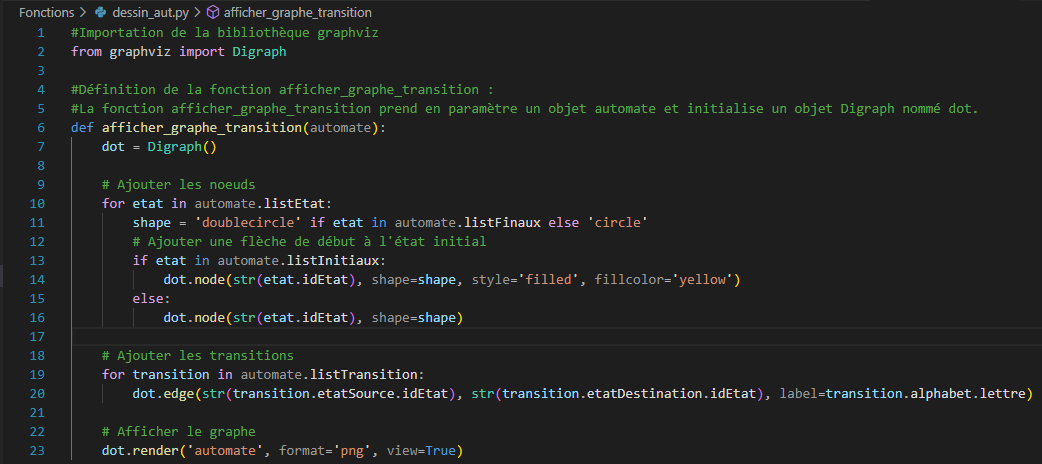


**Exemple d’utilisation**

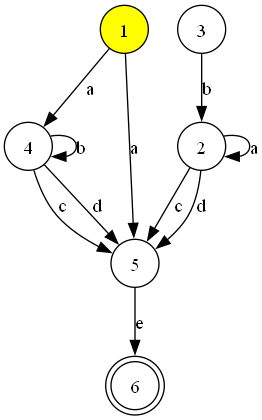
|  |  |
| --- | --- |
|  |  |

**Tableau 1 Résultat obtenu après minimisation de l'automate**

### Affichage du graphe de transition de l’automate sous forme graphique



Exemple d’affichage



**Figure 4 : Affichage graphique d'un automate**

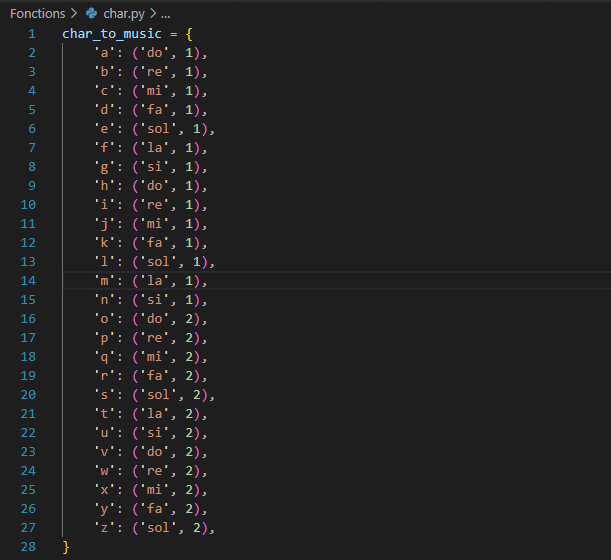
# Phase applicative

## Idée

Pour cette phase, mon idée est de générer de la musique automatiquement en utilisant les concepts des automates pour créer de la musique à partir du texte fourni par l'utilisateur. Le système convertit chaque caractère du texte en notes musicales. Voici les étapes de réalisation et les résultats obtenus durant la phase application.

## Étapes de Réalisation

### Définition des Règles Harmoniques et Rythmiques

Les caractères du texte sont mappés à des notes. Cette étape implique la création d'un dictionnaire (**char\_to\_music**) qui associe chaque caractère à une note et à une durée.  
  


### Classe Automate pour la Génération de Musique

La nouvelle version de la classe **Automate** inclut des fonctionnalités supplémentaires pour la génération de séquences musicales à partir du texte, ainsi que pour la gestion des états et des transitions dans ce contexte spécifique. Voici les ajouts et modifications effectués :

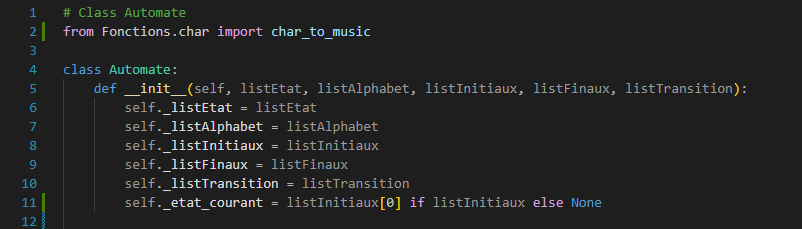
**Attributs et Initialisation**

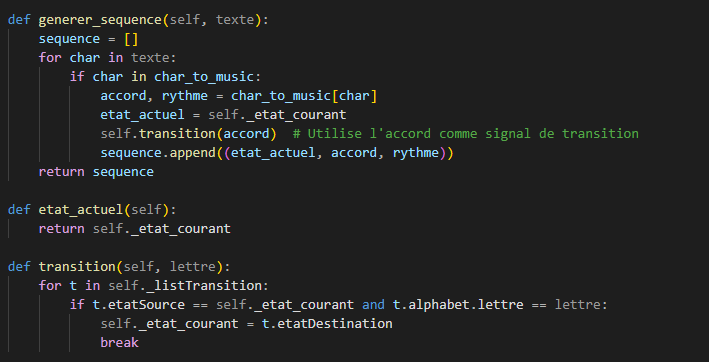
* Ajout de l'attribut **etat\_courant** pour suivre l'état actuel de l'automate.

**Méthodes**

1. **transition** : Cette méthode effectue une transition d'état en fonction d'un signal (ici, une note musicale).
2. **etat\_actuel** : Retourne l'état actuel de l'automate.
3. **generer\_sequence** : Génère une séquence musicale en fonction du texte fourni.

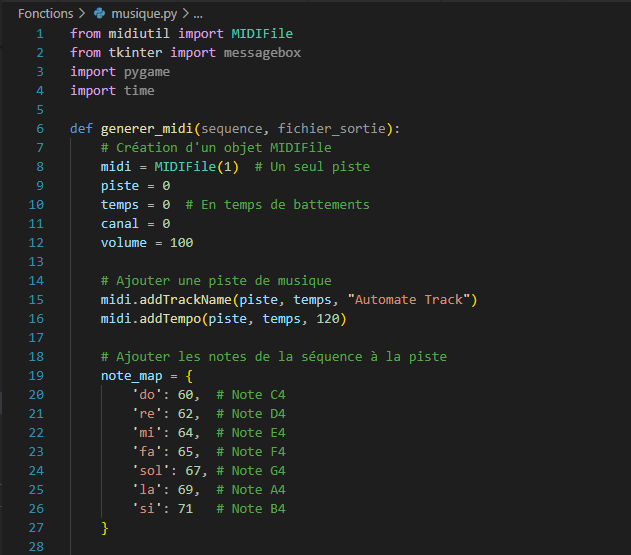
Voici ce qui a changé dans le code de base :

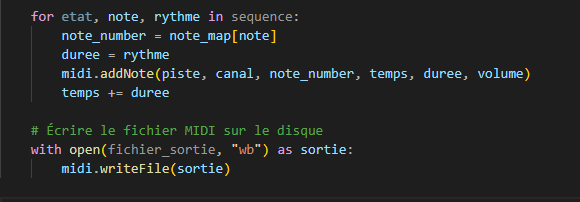




### Génération de Fichiers MIDI

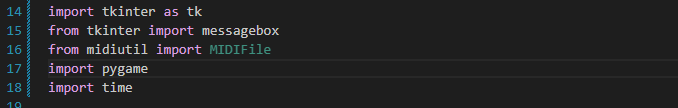
Une fonction **generer\_midi** a été ajoutée pour convertir la séquence musicale en un fichier MIDI.

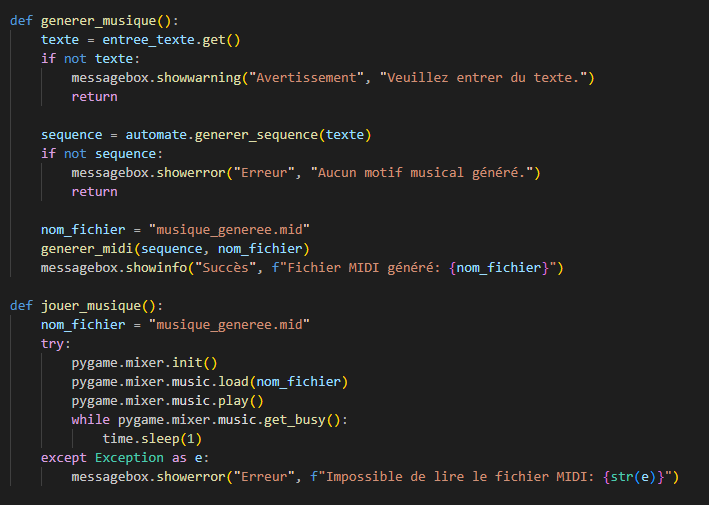




### Création de l’interface utilisateur

Une interface utilisateur (UI) a été créée en utilisant Tkinter. L'interface permet aux utilisateurs d'entrer du texte, de générer de la musique et de lire la musique directement.

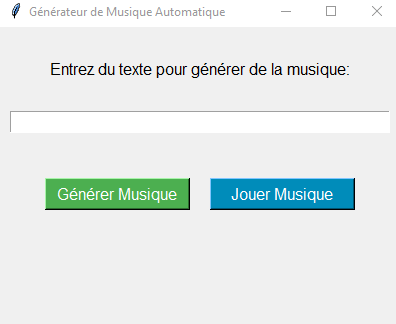






### Résultats Obtenus

1. **Génération de Séquences Musicales** : L'automate génère des séquences musicales en fonction du texte fourni par l'utilisateur. Chaque caractère est converti en une note et une durée.
2. **Création de Fichiers MIDI** : Les séquences générées sont converties en fichiers MIDI, ce qui permet de sauvegarder et de partager la musique.
3. **Lecture de Musique** : L'interface utilisateur permet de lire les fichiers MIDI générés directement, offrant une expérience utilisateur interactive et instantanée.



**Figure 5 : Interface Utilisateur**

# Conclusion et perspectives

## Bilan du projet

Le projet de génération de musique algorithmique via un automate a démontré comment les concepts théoriques des automates finis peuvent être appliqués à la création musicale. J’ai réussi à :

* Modéliser des motifs musicaux en utilisant une approche orientée objet.
* Développer des règles harmoniques et rythmiques pour guider la génération musicale.

## Limites Rencontrées

Malgré les succès, plusieurs limites ont été rencontrées :

* **Complexité des Règles Musicales** : La création de règles harmoniques et rythmiques précises est complexe et nécessite une expertise musicale approfondie.
* **Qualité Artistique** : Bien que l'automate puisse générer des séquences musicales, la qualité artistique et l'émotion véhiculée par la musique générée sont encore loin de rivaliser avec celles composées par des humains.
* **Déclencheurs Internes et Externes** : Des déclencheurs sophistiqués pour gérer les transitions et variations musicales en réponse à des conditions internes et des événements externes n'ont pas été implémentés dans cette version.

## Perspectives et Améliorations Futures

Pour améliorer et étendre ce projet, plusieurs pistes peuvent être envisagées :

* **Affinement des Règles Musicales** : Collaborer avec des musiciens professionnels pour affiner et enrichir les règles harmoniques et rythmiques, augmentant ainsi la qualité musicale des séquences générées.
* **Déclencheurs Internes et Externes** : Implémenter des déclencheurs internes et externes pour permettre une gestion plus dynamique et interactive des transitions musicales.
* **Expérience Utilisateur** : Développer une interface utilisateur conviviale permettant aux utilisateurs de créer, modifier et écouter des séquences musicales générées de manière intuitive.
* **Incorporation de l'Apprentissage Machine** : Utiliser des techniques d'apprentissage machine pour apprendre des modèles musicaux à partir de grandes bases de données de musique existante, ce qui pourrait améliorer la diversité et la qualité des séquences générées.

# Références Bibliographiques et Webographies

## Références Bibliographiques

* *Informatique Théorique et Applications*, Marc Chemillier
* *Monoïde libre et musique, première partie : les musiciens ont-ils besoin des mathématiques ?***,** Marc Chemillier
* *Rapport du projet d'Informatique Théorique II***,** Jérémy Turon & Salomé Coavoux
* *Learning Python,* par Mark Lutz

## Webographies

* + **Music FX - Google AI Test Kitchen,** [Music FX](https://aitestkitchen.withgoogle.com/fr/tools/music-fx)
  + [Python Documentation](https://docs.python.org/3/)
  + [Graphviz Documentation](https://graphviz.org/documentation/)
  + [Midiutil Documentation](https://midiutil.readthedocs.io/en/1.2.1/)