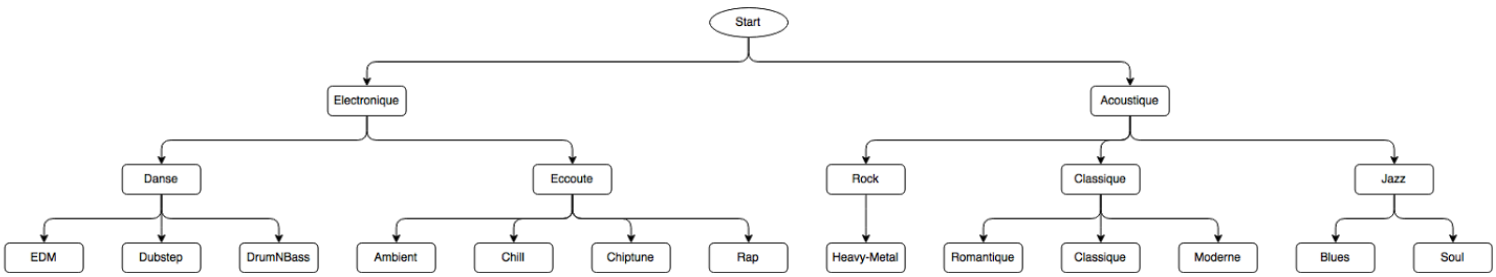
[IA01] TP3 Rendu final

Dans le cadre de l'UV IA01, nous avons programmé un système expert permettant la reconnaissance de genres musicaux.
Nous avons utilisé différents sites d'expertise pour créer notre système. Cependant, dans la mesure où la musique possède de nombreuses caractéristiques et qu'il est difficile d'établir une liste exhaustive de celles-ci, nous nous sommes aidé de notre propre appréciation des différents morceaux pour composer nos règles.

1. Sites d'expertise

http://www.koop.org/library/genres-definitions
http://persongified.tumblr.com/post/29757011469/the-key-differences-between-soul-music-and-blues
http://www.diffen.com/difference/Blues_vs_Jazz
https://fr.wikipedia.org/wiki/Chiptune
https://fr.wikipedia.org/wiki/Blues

2. Base de règles



R1 : Si (nombre instrument acoustique = faible) et si (son = électronique) alors (musique=electronique).
R2 : Si (nombre instrument acoustique = normal) et si (son = acoustique) alors (musique=acoustique).
R3 : Si (nombre instrument acoustique = solo) et si (son = acoustique) alors (musique = acoustique).
R3 : Si (musique= électronique) et que (club = oui) et que (important=beat) alors (style = dansant).
R4 : Si (musique= électronique) et que (club = non) alors (style = écoute).
R5 : Si (musique= acoustique), que (voix = oui) et que (paroles = chantées) et que (tempo > 5) et que (intruments = {guitare électrique U guitare acoustique U batterie}), et que (structure = strophique) alors (genre = rock).
R6 : Si (genre = rock), si (effets = saturation) et que (cris = oui), et que (tempo > 7) alors (sous-genre = heavy metal).
R7 : Si (musique = acoustique) et que (son = électrique) et que (intruments = {trompette U saxophone U batterie U synthé U contrebasse U guitare basse U clap}), que (rythme = triolet) et que (tempo < 5) alors (genre = jazz).

R8 : Si (genre = jazz), que (voix = oui), que (paroles = chantées), et que (intruments = {trompette U saxophone U batterie U synthé U contrebasse U guitare basse U harmonica) alors (sous-genre = blues).
R9 : Si (genre = jazz), que (voix = oui), que (paroles = chantées) et que (important = voix), et que (intruments = {trompette U saxophone U batterie U synthé U contrebasse U guitare basse U clap) alors (sous-genre = soul).
R10 : Si (musique= acoustique), que (instruments = {piano U violon U violoncelle U alto U flûte U hautbois U clarinette U trompette U trombone U vibraphone U grosse casse U cymbales}), que (voix = non) et (savante = oui) alors (genre = classique).
R11 : Si (genre = classique) et que (tonalité = non) alors (sous-genre = moderne).
R12 : Si (genre = classique) et que (instruments = clavecin) alors (sous-genre = baroque).
R13 : Si (genre = classique) et que (important = sentiments) alors (sous-genre = romantique).
R14 : S'il (orchestre = oui) alors (instruments = {violon U violoncelle U alto U flute U hautbois U clarinette U trompette U trombone U vibraphone U grosse caisse U cymbales})
R15 : Si (style = écoute), (voix = non) et que (bruit = 8-bit) alors (genre = chiptune).
R16 : Si (style = écoute) et si (voix = non) et si (drop = non), si (ambiance = calme) et (bruit = sourd) alors (genre = ambient)
R17 : Si (style = ecoute) et (voix = oui) et (ambiance = calme) et (bruit = sourd) et (mélodie = répétitive) alors (genre = chill)
R18 : Si (style = ecoute) et (voix oui) et (paroles = parlées) alors (genre = rap)
R19 : Si (style = danse) et (tempo > 8) et (batterie = forte) et (basse = forte) et (bruit = sourd) alors (genre = drumNbass)
R20 : Si (style = danse) et (rythme = brouille) et (drop = oui) et (mélodie = inexistante) et (voix = non) et (tempo > 7) alors (genre = dubstep)
R21 : Si (style = danse) et (important = beat) et (drop = oui) et (voix = oui) et (paroles = chantées) alors (genre = EDM)
R22 : Si (orchestre = oui) alors (nombre d'instruments acoustique = normal)

Quelques précisions :

La catégorie "Élément important" dans l'interface fait référence à un élément remarquable dans la musique. Exemple, si le beat est très présent dans la musique, alors il faut l'ajouter dans cette catégorie.

Savante : il s'agit de musiques qui mettent un accent particulier sur les questions formelles de style et qui invitent à la déconstruction technique et détaillée et exigent une attention plus pointue de l'auditeur. Cela implique des considérations structurelles et théoriques avancées. La musique classique fait donc partie de la catégorie des musiques savantes.

3. Représentation LISP

Pour mettre en place le système expert, nous avons choisi de représenter la base de faits sous la forme du a-list :

```
(setq *base_faits* '((carac1 valeur1 valeur2) (carac2 valeur 1) ...))
```

Chaque fait étant une des sous-listes.

Pour chaque règle, nous les avons représenté sous la forme suivante :

```
(setq R '( (premisses) (but) ))
```

Et chaque prémisse est représentée de la même façon que la base de faits. Le but est aussi représenté de la même manière qu'un fait.

Grâce à cette représentation, lors de l'appel du moteur avant, il est simple d'ajouter des faits dans la base de faits dans la mesure où il suffit de rajouter le but dans la base de faits et accéder au champ où il faut rajouter le fait ne nécessite qu'un assoc.

La base de faits au départ contient toutes les caractéristiques possibles avec les champs de valeur vide :

```
(setq *base_faits* '( (carac1) (carac2) (carac3) ...) )
```

Nous avons choisi d'utiliser une interface pour la saisie des faits plutôt que de poser les questions les unes après les autres car dans notre situation, il est difficile de faire un système de questions qui sont posées les unes après les autres : en effet, on ne peut orienter réellement les questions à poser dans la mesure où la musique possède toujours de nombreuses caractéristiques.

Pour le moteur d'inférence, nous avons dû définir différentes fonctions auxiliaires dont les plus notables sont les fonctions "question", "applicable" et "interface".

Dans le cadre des questions, nous avons établi un modèle pour chaque question :

```
(setq Q1 '( (question à afficher) (carac) (liste_carac) ) )
```

Pour chaque caractéristique, nous lui avons associé comme valeur toutes les valeurs possibles que celle-ci peut prendre.

ex : `(setq son '(acoustique electronique electrique))`

La "liste_carac" est une variable que nous allons modifié lors de l'appel de la question et nous y stockons la réponse entrée par l'utilisateur.

Fonction *question*

```
(defun question(Q faits)
	(let ((saisie))
		(format t (caar Q))
		(format t "~& Entrez Q pour sortir")
		(loop
			(print "Saisie :")
			(setq saisie (read))
			(cond
				((OR(equal saisie 'q) (equal saisie 'Q)) (return-from NIL
(caddr Q)))
					(t
					(if (member saisie (eval (caadr Q))) (setf (caddr Q)
(append (caddr Q) (list saisie)))
							(print "error")
							)
					)
				)
			)
```

```
                (setf (cdr (assoc (caadr Q) faits)) (cdaddr Q))
        )
```

Pour pouvoir attendre la saisie de l'utilisateur, nous utilisons la fonction read qui attend que l'utilisateur saisisse une réponse. Ensuite, nous vérifions que l'entrée de l'utilisateur est bien une entrée possible et existante dans notre système expert grâce la valeur de la caractéristique (cf au-dessus). Ensuite, tant que l'utilisateur a des valeurs à rentrer, il continue. Pour s'arrêter, il faut saisir 'q'. Au bout de la fonction, on modifie la base de faits entrée argument dans la fonction.

Fonction *nbr_instruments*

```
(defun nbr_instruments(*faits*)
    (if (< (list-length (cdr (assoc 'instruments *faits*))) 3)
        (if (= (list-length (cdr (assoc 'instruments *faits*))) 1)
               (setf (cdr (assoc 'nbInstruments *faits*)) (list 'solo))
               (setf (cdr (assoc 'nbInstruments *faits*)) (list 'peu))
               )
        (setf (cdr (assoc 'nbInstruments *faits*)) (list 'normal))
    )
)
```

Cette fonction nbr_instruments est une fonction qui permet de mettre a jour lla caractéristique NbrInstruments : en effet, cette musique dépend de la caractéristique instruments car elle dépend du nbr d'instruments entrés. On utilise la longueur de la liste instruments.

Fonction *applicable*

```
(defun applicable(regle base_faits)
       (let ((ok t) (compteur 0))
              (dolist (x (car regle) ok)
                     (if (equal (car x) 'instruments)
                                   (dolist (y (cdr (assoc 'instruments base_faits)))
                                          (if (not (member y x)) (setq ok nil))
                                                 )
                                   (if (equal 'tempo (car x))
                                          (cond
                                                 ((equal (cadr x) '<) (if (> (tempo_value
*faits*) (caaddr x)) (setq ok NIL)))
                                                 ((equal (cadr x) '>) (if (< (tempo_value
*faits*) (caaddr x)) (setq ok NIL)))
                                          )
                                          (if (not (member x base_faits :test #'equal))
(setq ok NIL))
                                   )
                     )
              )
              ok
       )
```

```
)
```

Cette fonction prend en arguments la base de règles et la base de faits et retourne un booléen “ok” qui reste à T tant qu’il n’y a pas de contradiction. Il y a 3 cas à étudier:

- Instruments

Dans le cas où la prémisses est un champ instrument, nous vérifions qu’il n’y a pas d’instruments dans la base de faits qui n’est pas présent dans la prémisses de la règle. Si c’est le cas, la règle ne sera pas applicable (setq ok NIL)

- Tempo

Dans le cas du tempo, le fait dans la base de règle est représenté de façon différente que dans la base de faits : (tempo ‘> 8). Ainsi, pour vérifier la règle, nous devons voir dans la prémisses de quel signe il s’agit ainsi que de quelle valeur il s’agit. On changera donc la valeur de “ok” en fonction.

- Autres

Dans tous les autres cas, nous vérifions que la prémisse est présente dans la base de fait.

Pour entrer les valeurs dans la base de faits, nous utilisons une fonction interface qui fait appel à notre fonction question. L’interface est représenté à la manière d’un menu à choix multiples, chaque choix correspondant à un fait à modifier.

Moteur avant largeur :

```
(defun moteur_avant_largeur(*regles* *faits*)      ;largeur
   (let ((bf *faits*) (br *regles*) (ok NIL))
           (while (null (cdr (assoc 'genre bf)))
                   (setq ok NIL)
                   (dolist (r br)
                           (if (applicable (eval r) *faits*)
                                   (progn
                                           (setq ok t)
                                           (setf (cdr (assoc (enonce_but (eval r)) bf)) (but (eval
r)))
                                           (setq br (remove r br))         ;supprimer regle de br
                                           )
                                   )
                           )
                   (if (equal ok NIL) (return-from moteur_avant_largeur (format t "Le moteur n'a pas
pu trouver de genre correspondant dans la base de donnees~&~&")))
           )

           (if (member (cadr (assoc 'genre bf)) gSousgenres)
                   (if (not (null (cdr (assoc 'sous-genre *faits*))))
                           (return-from moteur_avant_largeur (format t "~& ~& Le sous-genre de la
musique est : ~a ~& ~& ~&"(cadr (assoc 'sous-genre *faits*))))
                           (dolist (r br)
                                   (if (and (applicable (eval r) *faits* ) (member (assoc 'genre bf)
(car (eval r)) :test #'equal))
                                           (progn
                                                   (setf (cdr (assoc (enonce_but (eval r)) bf)) (but
(eval r)))
                                                   (return-from moteur_avant_profondeur (format t "~&
~& Le sous-genre de la musique est : ~a ~& ~& ~&"(car (but (eval r)))))
                                           )
                                   )
```

```
                        )
                )
        )
        (format t "~& ~& Le genre de la musique est : ~a ~& ~& ~&"(cadr (assoc 'genre bf)))
    )
)
```

Notre moteur avant prend en paramètres la base de règle et la base de faits. Nous créons des copies de ces bases avec des variables locales dans la fonction pour éviter de détruire ces bases et de les changer par inattention.
Tant que nous n'avons pas trouvé de genre associé à la musique écoutée (tant que le cdr de genre est null) alors on met une variable OK à NIL qui nous permettra de vérifier s'il reste des règles applicables ou non. On vérifie ensuite les règles une à une dans la base de règle pour voir si elles sont applicables (cf. fonction précédente). Si elle l'est, alors on met OK à vrai car il existe au moins une règle applicable. On met le résultat de la règle dans la base de fait et on enlève la règle que l'on a appliqué de notre base de règle. On continue tant que nous n'avons pas trouvé de genre ou que OK est à NIL. Si OK est à NIL alors on arrête le moteur.

Le moteur va ensuite vérifier si le genre trouvé contient des sous-genres grâce à la liste gSousgenres. Si oui, alors soit la règle associée au sous-genre a été appliquée : on va donc afficher le sous-genre en question; soit on reparcourt la base de règle et on cherche une règle qui est à la fois applicable mais aussi dont le genre correspond. La première règle vérifiant ces conditions donne le sous-genre de la musique.
Dans le cas contraire, on donne juste le genre de la musique et on arrête le moteur. Le moteur est donc un moteur d'inférence avant en largeur.

Comparaison avec le moteur avant en profondeur :

```
(defun moteur_avant_profondeur(*regles* *faits*)       ;profondeur
        (let ((bf *faits*) (br *regles*) (ok NIL) (result NIL))
                (while (null (cdr (assoc 'genre bf)))
                        (setq ok NIL)
                        (dolist (r br)
                                (if (applicable (eval r) *faits*)
                                        (progn
                                                (setq ok t)
                                                (setf (cdr (assoc (enonce_but (eval r)) bf)) (but
(eval r)))
                                                (setq br (remove r br))        ;supprimer regle de
br
                                                (return-from                moteur_avant_profondeur
(moteur_avant_profondeur br bf))
                                                )
                                        )
                                )
                        (if (equal ok NIL) (return-from moteur_avant_profondeur (format t "Le
moteur n'a pas pu trouver de genre correspondant dans la base de donnees~&~&")))
                )

                (if (member (cadr (assoc 'genre bf)) gSousgenres)
                        (dolist (r br )
```

```
                                (if (and (applicable (eval r) *faits* ) (member (assoc 'genre bf)
(car (eval r)) :test #'equal))
                                                        (return-from moteur_avant_profondeur (format t "~&
~& Le sous-genre de la musique est : ~a ~& ~& ~&"(car (but (eval r)))))
                                )
                        )
                )
                (format t "~& ~& Le genre de la musique est : ~a ~& ~& ~&"(cadr (assoc 'genre bf)))
        )
)
```

Au lieu d'appliquer toutes les règles applicables d'un coup, dès qu'une règle est applicable alors on relance le moteur avant sur la nouvelle base de faits avec la nouvelle conclusion qui vient d'être ajoutée. On retourne le résultat de l'appel récursif.

4. Scénario d'utilisation

Tout d'abord, il faut exécuter la fonction (diapason) sur l'interface Lisp. Ensuite il faut rentrer les réponses aux questions que vous choisissez dans le menu. Il n'est pas nécessaire de répondre à toutes les questions, juste à celles dont la caractéristique vous semble présente dans la musique entendue. Attention, toutes les questions sont à choix unique SAUF celles des instruments, où il faudra rentrer les instruments entendus et appuyer sur q lorsque vous aurez fini.
Lorsque vous pensez avoir répondu à toutes les questions possibles, il suffira de choisir l'option Recherche.

Voici la liste des instruments que le programme pourra reconnaître :

guitare_acoustique	vent
violon	percussions
mandoline	saxophone
banjo	trompette
harmonica	contrebasse
guitare_basse	violoncelle
castagnettes	clarinette
clap	hautbois
cajon	trombone
batterie	vibraphone
guitare_electrique	grosse_caisse
melodica	cymbales
piano	clavecin
chorale	alto
synthe	flute

5. Jeux d'essai

```
(setq *base_faits* '((musique) (son acoustique) (nbInstruments) (instruments piano)
(melodie) (orchestre) (rythme) (danse) (genre) (bruit) (effets) (langue) (voix non)
(sous-genre)
       (drop) (tonalite) (important sentiment) (langue) (basse) (club) (paroles) (style)
(savante oui) (ambiance) (tempo)
                        )
)

(diapason)
```

R3 (musique acoustique) -> R11 (genre classique) -> R14 (sous-genre romantique)

```
(setq *base_faits* '((musique) (son electronique) (nbInstruments) (instruments) (melodie)
(orchestre) (rythme) (danse) (genre) (bruit) (effets) (langue) (voix oui) (sous-genre)
       (drop oui) (tonalite) (important beat) (langue) (basse) (club oui) (paroles
chantees) (style) (savante) (ambiance) (tempo)
                        )
)
```

R1 (musique electronique) -> R4 (style danse) -> R22 (genre EDM)

```
(setq *base_faits* '((musique) (son acoustique) (nbInstruments) (instruments trompette
saxophone batterie contrebasse) (melodie) (orchestre) (rythme triolet) (danse) (genre)
(bruit) (effets) (langue) (voix) (sous-genre)
       (drop) (tonalite) (important) (langue) (basse) (club) (paroles) (style) (savante)
(ambiance) (tempo 4)
                        )
)
```

R2 (musique acoustique) -> R8 (genre jazz)